Before using this information and the product it supports, read the information in Appendix C, “Notices,” on page 497.
## Contents

**Preface** .............................................. xvii
- Who should read this book ................................ xvii
- What this book contains .................................. xvii

**Publications** ........................................... xix
- Release information ...................................... xix
- Base information ......................................... xix
- Web security information ................................. xx
- Developer references ................................... xx
- Technical supplements .................................. xxi
- Related publications ..................................... xxi
  - IBM Global Security Kit ................................ xxi
  - IBM Tivoli Directory Server ............................. xxi
  - IBM DB2 Universal Database ............................ xxi
  - IBM WebSphere Application Server ................... xxi
  - IBM Tivoli Access Manager for Business Integration ........................................... xxi
  - IBM Tivoli Access Manager for WebSphere Business Integration Brokers .................. xiii
  - IBM Tivoli Access Manager for Operating Systems ..................................... xiii
  - IBM Tivoli Identity Manager ................................ ..................... xiv
- Accessing publications online ........................... xxv
- Accessibility ............................................ xxv
- Contacting software support .............................. xxv
- Conventions used in this book ............................ xxv
  - Typeface conventions ................................... xxv
  - Operating system differences ........................... xxvi

**Chapter 1. IBM Tivoli Access Manager WebSEAL overview** ..................................... 1
- Introducing IBM Tivoli Access Manager and WebSEAL ........................................ 1
- Understanding the Tivoli Access Manager security model .................................. 2
  - The protected object space ................................ 3
  - Defining and applying ACL and POP policies ................................ 3
    - The access control list (ACL) .................................. 4
    - Protected object policies (POP) ................................. 4
    - Explicit and inherited policy ................................... 5
  - Policy administration: The Web Portal Manager .................................. 5
- Protecting the Web space with WebSEAL .................................................. 5
- Planning and implementing the security policy .......................................... 6
  - Identifying content types and levels of protection ................................... 7
- Understanding WebSEAL authentication ..................................................... 8
  - The goals of authentication ........................................ 8
  - Authenticated and unauthenticated access to resources ................................ 9
  - The WebSEAL session/credentials cache structure .................................. 10
- Understanding WebSEAL junctions ............................................................ 11
  - WebSEAL junctions and Web site scalability ........................................ 12
    - Replicated front-end WebSEAL servers .................................. 13
    - Supporting back-end servers .................................... 13
    - Replicated back-end servers .................................. 13

**Chapter 2. WebSEAL server configuration** .................................................. 15
- Server instance configuration ............................................ 16
  - Server instance configuration overview ................................ 16
    - Planning a server instance configuration .................................. 16
    - Example server instance configuration values ................................ 20
  - Unique configuration file for each instance ....................................... 20
  - Interactive configuration overview ........................................... 20
  - Command line configuration overview ........................................... 20

© Copyright IBM Corp. 1999, 2003
Preventing vulnerability caused by cross-site scripting ........................................... 56
  Background ........................................................................................................... 56
Configuring URL string filtering ............................................................................... 56
Replicated front-end WebSEAL servers ..................................................................... 58
Platform for Privacy Preferences (P3P) ..................................................................... 59
  Compact policy overview ......................................................................................... 59
  Compact policy declaration ....................................................................................... 59
  Junction header preservation ................................................................................... 60
  Default compact policy in the P3P header ................................................................. 61
  P3P header configuration ......................................................................................... 62
  Specifying a custom P3P compact policy ................................................................. 67
Troubleshooting ......................................................................................................... 68
Suppressing server identity ......................................................................................... 69
Handling BASE HREF tags ......................................................................................... 69
Enabling HTTP TRACE method .................................................................................. 69

Chapter 3. WebSEAL server administration ................................................................. 71
Server tasks .................................................................................................................. 72
  Start a WebSEAL server ............................................................................................ 72
  Stop a WebSEAL server ............................................................................................ 72
  Restart a WebSEAL server ....................................................................................... 73
  Display WebSEAL server status .............................................................................. 73
Managing the Web space ............................................................................................ 74
  WebSEAL represented in the protected object space ................................................ 74
  Root directory of the Web document tree ............................................................... 74
  Configuring directory indexing ............................................................................... 75
  Windows: File naming for CGI programs ................................................................. 76
  Executable UNIX files on the WebSEAL server host system .................................... 77
  Configuring Web document caching ....................................................................... 77
    Conditions affecting Web document caching ......................................................... 78
    Flushing all caches ................................................................................................. 78
    Controlling caching for specific documents ......................................................... 78
  Specifying document MIME types for URL filtering .............................................. 79
HTTP data compression ............................................................................................. 80
  Compression based on MIME-type ........................................................................... 80
  Compression based on user agent type .................................................................... 81
  Compression policy in POPs ...................................................................................... 82
  Data compression limitation ...................................................................................... 82
  Data compression configuration ............................................................................... 82
HTTP error message pages ......................................................................................... 84
  Default error message pages .................................................................................... 84
  Modifying existing HTTP error pages .................................................................... 84
  Creating new HTTP error pages .............................................................................. 86
  Enabling the time of day error page ........................................................................ 86
  Specifying error page location ................................................................................ 87
  Backwards compatibility ......................................................................................... 87
Managing custom account management pages ......................................................... 88
  Custom page parameters and values ....................................................................... 88
  Custom HTML page descriptions .......................................................................... 89
  Macro support for account management pages ...................................................... 89
  Modifying pages from prior versions of Tivoli Access Manager ............................... 91
Backup and restore ...................................................................................................... 92
  Backup WebSEAL data ............................................................................................. 92
  Restore WebSEAL data ............................................................................................ 93
  Extract archived WebSEAL data .............................................................................. 93
  Problem determination tools for WebSEAL ............................................................. 94

Chapter 4. Serviceability and logging ........................................................................ 95
Logging WebSEAL serviceability messages .............................................................. 96
  Serviceability messages in UTF-8 format ............................................................... 97
## Chapter 5. WebSEAL security policy

### WebSEAL-specific ACL policies
- /WebSEAL/host-instance_name/file
- /WebSEAL/host-instance_name

### WebSEAL ACL permissions
- Default /WebSEAL ACL policy
- Valid characters for ACL names

### Configuring three strikes login policy
- Account lock policy with load-balanced WebSEAL servers
- Syntax for three strikes login commands

### Configuring password strength policy
- Password strength policy set by the pdadmin utility
- Syntax for password strength policy commands
- Default policy parameter values
- Valid and invalid password examples
- Specific user and global settings

### Authentication strength policy
- Overview of authentication strength
- Authentication strength configuration
- Establish an authentication strength policy
- Specify authentication levels
- Specify the authentication strength login form
- Create a protected object policy
- Specify network-based access restrictions
- Attach a protected object policy to a protected resource
- Enforce user identity match across authentication levels

### Quality of protection POP policy
- Handling unauthenticated users (HTTP / HTTPS)
- Processing a request from an anonymous client
- Forcing user login
- Applications of unauthenticated HTTPS

## Chapter 6. Authentication

### Overview of the authentication process
- Supported session data types
- Supported authentication methods

### Managing session state
- Session state overview
- GSKit and WebSEAL session cache overview
- Configuring the GSKit SSL session ID cache
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting the cache entry timeout value</td>
<td>133</td>
</tr>
<tr>
<td>Setting the maximum concurrent entries value</td>
<td>133</td>
</tr>
<tr>
<td>Configuring the WebSEAL session/credentials cache</td>
<td>133</td>
</tr>
<tr>
<td>Setting the maximum concurrent entries value</td>
<td>133</td>
</tr>
<tr>
<td>Setting the cache entry lifetime timeout value</td>
<td>134</td>
</tr>
<tr>
<td>Setting the cache entry inactivity timeout value</td>
<td>134</td>
</tr>
<tr>
<td>Credentials cache limitation</td>
<td>134</td>
</tr>
<tr>
<td>Maintaining state with session cookies</td>
<td>135</td>
</tr>
<tr>
<td>Session cookie conditions</td>
<td>135</td>
</tr>
<tr>
<td>Session cookies with basic authentication headers</td>
<td>135</td>
</tr>
<tr>
<td>Enabling and disabling session ID cookies</td>
<td>136</td>
</tr>
<tr>
<td>Enabling and disabling same sessions</td>
<td>137</td>
</tr>
<tr>
<td>Same session limitation with Netscape 4.7x</td>
<td>138</td>
</tr>
<tr>
<td>Determining valid session ID data types</td>
<td>138</td>
</tr>
<tr>
<td>Authentication configuration overview</td>
<td>140</td>
</tr>
<tr>
<td>Authentication module parameters</td>
<td>140</td>
</tr>
<tr>
<td>Authentication conversion library</td>
<td>141</td>
</tr>
<tr>
<td>Default configuration for WebSEAL authentication</td>
<td>141</td>
</tr>
<tr>
<td>Configuring multiple authentication methods</td>
<td>142</td>
</tr>
<tr>
<td>Prompting for login</td>
<td>142</td>
</tr>
<tr>
<td>Configuring account expiry notification</td>
<td>142</td>
</tr>
<tr>
<td>Logout and change password commands</td>
<td>143</td>
</tr>
<tr>
<td>pkmslogout</td>
<td>143</td>
</tr>
<tr>
<td>pkmspasswd</td>
<td>143</td>
</tr>
<tr>
<td>Post password change processing</td>
<td>144</td>
</tr>
<tr>
<td>Basic authentication</td>
<td>145</td>
</tr>
<tr>
<td>Enabling and disabling basic authentication</td>
<td>145</td>
</tr>
<tr>
<td>Setting the realm name</td>
<td>145</td>
</tr>
<tr>
<td>Configuring the basic authentication mechanism</td>
<td>145</td>
</tr>
<tr>
<td>Configuration conditions</td>
<td>146</td>
</tr>
<tr>
<td>Multi-byte UTF-8 logins not supported</td>
<td>146</td>
</tr>
<tr>
<td>Forms authentication</td>
<td>147</td>
</tr>
<tr>
<td>Enabling and disabling forms authentication</td>
<td>147</td>
</tr>
<tr>
<td>Configuring the forms authentication mechanism</td>
<td>147</td>
</tr>
<tr>
<td>Configuration conditions</td>
<td>148</td>
</tr>
<tr>
<td>Customizing HTML response forms</td>
<td>148</td>
</tr>
<tr>
<td>Client-side certificate authentication</td>
<td>149</td>
</tr>
<tr>
<td>Overview of client-side certificate authentication</td>
<td>149</td>
</tr>
<tr>
<td>Required certificate authentication mode</td>
<td>149</td>
</tr>
<tr>
<td>Optional certificate authentication mode</td>
<td>149</td>
</tr>
<tr>
<td>Delayed certificate authentication mode</td>
<td>149</td>
</tr>
<tr>
<td>Certificate authentication configuration</td>
<td>151</td>
</tr>
<tr>
<td>Enable certificate authentication</td>
<td>151</td>
</tr>
<tr>
<td>Specify the certificate authentication mechanism</td>
<td>152</td>
</tr>
<tr>
<td>Specify the certificate login form</td>
<td>152</td>
</tr>
<tr>
<td>Specify the certificate login error page</td>
<td>152</td>
</tr>
<tr>
<td>Disable SSL session IDs for session tracking</td>
<td>153</td>
</tr>
<tr>
<td>Enable and configure the Certificate SSL ID cache</td>
<td>153</td>
</tr>
<tr>
<td>Set the timeout for Certificate SSL ID cache</td>
<td>153</td>
</tr>
<tr>
<td>Specify an error page for incorrect protocol</td>
<td>154</td>
</tr>
<tr>
<td>Disable certificate authentication</td>
<td>155</td>
</tr>
<tr>
<td>Disable the Certificate SSL ID cache</td>
<td>155</td>
</tr>
<tr>
<td>HTTP header authentication</td>
<td>156</td>
</tr>
<tr>
<td>Enable HTTP header authentication</td>
<td>156</td>
</tr>
<tr>
<td>Specify header types</td>
<td>157</td>
</tr>
<tr>
<td>Specify the HTTP header authentication mechanism</td>
<td>157</td>
</tr>
<tr>
<td>Disable HTTP header authentication</td>
<td>158</td>
</tr>
<tr>
<td>IP address authentication</td>
<td>159</td>
</tr>
<tr>
<td>Enabling and disabling IP address authentication</td>
<td>159</td>
</tr>
<tr>
<td>Configuring the IP address authentication mechanism</td>
<td>159</td>
</tr>
<tr>
<td>Token authentication</td>
<td>160</td>
</tr>
</tbody>
</table>
### Chapter 8. WebSEAL Key Management

WebSEAL Key Management Overview .................................................. 227
Managing client-side and server-side certificates ................................ 228
GSKit key database file types .............................................................. 228
Chapter 9. Client single sign-on solutions ........................................ 233

Windows desktop single sign-on .................................................. 234
  Windows desktop single sign-on concepts ................................ 234
  SPNEGO protocol and Kerberos authentication ......................... 234
  User registry and platform support ......................................... 235
  Compatibility with other authentication methods ....................... 235
  Limitations ............................................................................. 236

Windows desktop single sign-on configuration .............................. 237
  Step 1: Configure WebSEAL server into Active Directory domain ..... 237
  Step 2: Map Kerberos principal to Active Directory user ............. 238
  Step 3: Install Kerberos runtime client (UNIX only) ................... 240
  Step 4: Configure Kerberos client (UNIX only) ......................... 240
  Step 5: Verify authentication of Web server principal (UNIX only) 241
  Step 6: Verify WebSEAL authentication using the keytab file (UNIX only) 242
  Step 7: Enable SPNEGO for WebSEAL ...................................... 242
  Step 8: Add service name and keytab file entries (UNIX only) ....... 242
  Step 9: Restart WebSEAL ....................................................... 243
  Step 10: Configure the Internet Explorer client ......................... 243
  Troubleshooting tips .................................................................. 243

Cross-domain single sign-on ......................................................... 245
  Customizing single sign-on authentication .................................. 245
  Authentication process flow for CDSSO with CDMF .................... 245

Configuring CDSSO authentication ................................................. 247
  CDSSO conditions and requirements ........................................ 247
  Resolving machine names ....................................................... 247
  CDSSO configuration summary ................................................. 248
    Configuring default CDSSO token create functionality .............. 248
    Configuring default CDSSO token consume functionality ........... 248
  1. Enabling and disabling CDSSO authentication ....................... 249
  2. Configuring the single sign-on authentication mechanism .......... 249
  3. Encrypting the authentication token data .............................. 250
  4. Configuring the token time stamp ........................................ 251
  5. Configuring the token label name ........................................ 251
  Creating the CDSSO HTML link .............................................. 251
  Protecting the authentication token ........................................ 252
  UTF-8 encoding of tokens for cross domain single sign-on .......... 252
  Enabling compatibility with tokens prior to Version 4.1 ............. 252
  Enable backwards compatibility for Version 4.1 tokens ............... 253
  Specify extended attributes to add to token ............................ 253
  Specify extended attributes to extract from token .................... 254

E-community single sign-on ......................................................... 256
  E-community features and requirements .................................... 257
  E-community process flow ..................................................... 258
  Understanding the E-community cookie .................................... 261
  Understanding the "vouch for" request and reply ....................... 262
    The "vouch for" Request .................................................. 262
    The "vouch for" Reply .................................................... 262
  Understanding the "vouch for" token .................................... 263

Configuring E-community single sign-on ..................................... 263
  E-community conditions and requirements ............................... 263
    Resolving machine names .................................................. 264
  E-community configuration summary ...................................... 264
    Configuring default token create functionality on the "vouch for" server .................................................. 265
    Configuring default token consume functionality on the receiving server .................................................. 265

X  IBM Tivoli Access Manager for e-business: WebSEAL Administration Guide
1. Enabling and disabling e-community authentication ........................................... 266
2. Specifying an e-community name ................................................................. 266
3. Configuring the single sign-on authentication mechanism ......................... 266
4. Encrypting the "vouch for" token ................................................................. 267
   E-community Domain Keys ............................................................................. 267
5. Configuring the "vouch for" token label name .............................................. 268
6. Specifying the master authentication server (MAS) ...................................... 268
7. Specifying the "vouch for" URL ..................................................................... 269
8. Configure token and ec-cookie lifetime values ........................................... 270
Enabling unauthenticated access ......................................................................... 270
UTF-8 encoding of tokens for e-community single sign-on .............................. 271
Enabling compatibility with tokens prior to Version 4.1 ............................... 271
Enable backwards compatibility for Version 4.1 tokens ............................... 271
Specify extended attributes to add to token .................................................... 272
Specify extended attributes to extract from token ......................................... 273

Chapter 10. WebSEAL junctions ............................................................................. 275
WebSEAL junctions overview ............................................................................. 276
   Junction database location and format ......................................................... 276
   Applying coarse-grained access control: summary ........................................ 276
   Applying fine-grained access control: summary ........................................... 276
   Guidelines for creating WebSEAL junctions .................................................. 276
   Additional references for WebSEAL junctions .............................................. 277
Managing junctions with Web Portal Manager .................................................. 278
   Create a junction using Web Portal Manager ............................................. 278
   List junctions using Web Portal Manager .................................................... 278
   Delete junctions using Web Portal Manager ................................................. 278
Using pdadmin to create junctions ..................................................................... 279
Configuring a basic WebSEAL junction ............................................................. 280
   Creating TCP type junctions ....................................................................... 280
   Creating SSL type junctions ....................................................................... 280
      Verifying the back-end server certificate ................................................. 280
      Examples of SSL junctions ...................................................................... 281
      Disabling SSL protocol versions for junctions ........................................... 281
   Adding back-end servers to a junction .......................................................... 281
Mutually authenticated SSL junctions ................................................................. 282
   WebSEAL validates back-end server certificate .......................................... 282
   Distinguished name (DN) matching ............................................................ 282
   WebSEAL authenticates with client certificate ............................................. 283
   WebSEAL authenticates with BA header ..................................................... 283
Handling client identity information across junctions ........................................ 284
   Using -b supply .......................................................................................... 284
   Using -b ignore .......................................................................................... 284
   Using -b gso ............................................................................................... 284
   Using -b filter ............................................................................................. 284
Creating TCP and SSL proxy junctions ............................................................... 285
WebSEAL-to-WebSEAL junctions over SSL ......................................................... 286
Modifying URLs to back-end resources ............................................................ 287
   Understanding path types used in URLs ...................................................... 288
Filtering URLs in responses .............................................................................. 288
   Standard URL filtering rules for WebSEAL ................................................ 288
   Modifying absolute URLs with script filtering ......................................... 290
   Filtering changes the Content-Length header .......................................... 290
   Limitation with unfiltered server-relative links ......................................... 291
Processing URLs in requests .............................................................................. 292
   Handling server-relative URLs with junction cookies (-j) ......................... 292
   Handling server-relative URLs with junction mapping ................................ 293
   Processing root junction requests ............................................................... 294
Handling cookies from servers across multiple -j junctions ............................ 295
   Part 1: -j junctions modify Set-Cookie Path attributes ............................ 295
   Part 2: -j junctions modify Set-Cookie Name attributes ........................... 296
Preserving cookie names ............................................................. 296
Additional junction options. ....................................................... 297
Forcing a new junction (–f) ........................................................ 297
Supplying client identity in HTTP headers (–c) ............................... 298
–c syntax ................................................................................. 298
Supplying client IP addresses in HTTP headers (–r) ......................... 299
Limiting the size of WebSEAL-generated HTTP headers .................. 299
Passing session cookies to junctioned portal servers (–k) ................. 300
Supporting case-insensitive URLs (–i) .......................................... 300
Stateful junction support (–s, –u) .................................................. 301
Specifying back-end server UUIDs for stateful junctions (–u) ............. 302
Example: .................................................................................. 303
Junctioning to Windows file systems (–w) ....................................... 304
Example: .................................................................................. 304
ACLs and POPs must attach to lower-case object names .................. 305
Specifying UTF-8 encoding for HTTP header data ............................ 305
Technical notes for using WebSEAL junctions. ................................. 307
Mounting multiple servers at the same junction ............................... 307
Exceptions to enforcing permissions across junctions ....................... 307
Certificate authentication across junctions ...................................... 307
Handling domain cookies ............................................................ 308
WebSEAL returns HTTP/1.1 ....................................................... 308
Junctioned application with Web Portal Manager .............................. 308
Using query_contents with third-party servers ............................... 309
Installing query_contents components ........................................ 309
Installing query_contents on third-party UNIX servers .................... 309
Installing query_contents on third-party Win32 servers ................... 310
Testing the configuration .............................................................. 311
Customizing query_contents ...................................................... 311
Customizing the doc root directory .............................................. 312
Additional functionality .............................................................. 312
Securing query_contents ............................................................ 312
Troubleshooting ....................................................................... 313

Chapter 11. Single sign-on solutions across junctions. ....................... 315
Configuring BA headers for single sign-on solutions ......................... 316
Single sign-on (SSO) concepts ..................................................... 316
Supplying client identity in BA headers ......................................... 316
Supplying client identity and generic password ............................... 316
Limitations .............................................................................. 317
Forwarding original client BA header information ............................ 318
Removing client BA header information ........................................ 318
Supplying user names and passwords from GSO ............................ 318
Using global sign-on (GSO). ..................................................... 320
Mapping the authentication information ........................................ 321
Configuring a GSO-enabled WebSEAL junction ............................. 322
Examples of GSO-enabled WebSEAL junctions ............................... 322
Configuring the GSO cache ......................................................... 322
Configuring single sign-on to IBM WebSphere (LTPA) ..................... 324
Configuring an LTPA junction ..................................................... 324
Configuring the LTPA cache ....................................................... 325
Technical notes for LTPA single sign-on ........................................ 325
Configuring single sign-on forms authentication. .......................... 326
Background and goals .................................................................. 326
Forms single sign-on process flow ................................................. 326
Requirements for application support .......................................... 328
Creating the configuration file for forms single sign-on .................... 328
The [forms-ssologin-pages] stanza .............................................. 329
The custom login page stanza ..................................................... 329
Using regular expressions ........................................................... 330
The argument stanza ................................................................. 331
**Chapter 12. Application integration** ................................................. 335

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabling forms single sign-on</td>
<td>332</td>
</tr>
<tr>
<td>Example configuration file for IBM HelpNow</td>
<td>332</td>
</tr>
<tr>
<td>Supporting CGI programming</td>
<td>336</td>
</tr>
<tr>
<td>UTF-8 environment variables for CGI programs</td>
<td>336</td>
</tr>
<tr>
<td>Windows: Supporting WIN32 environment variables</td>
<td>337</td>
</tr>
<tr>
<td>Supporting back-end server-side applications</td>
<td>338</td>
</tr>
<tr>
<td>Junction “best practices” for application integration</td>
<td>339</td>
</tr>
<tr>
<td>Supplying complete HOST header information with <code>-v</code></td>
<td>339</td>
</tr>
<tr>
<td>Supporting standard absolute URL filtering</td>
<td>339</td>
</tr>
<tr>
<td>Building a custom personalization service</td>
<td>341</td>
</tr>
<tr>
<td>Configuring WebSEAL for a personalization service</td>
<td>341</td>
</tr>
<tr>
<td>Personalization service example</td>
<td>341</td>
</tr>
<tr>
<td>Maintaining session state between client and back-end applications</td>
<td>343</td>
</tr>
<tr>
<td>Background to user session management</td>
<td>343</td>
</tr>
<tr>
<td>Enabling user session id management</td>
<td>343</td>
</tr>
<tr>
<td>Inserting credential data into the HTTP header</td>
<td>344</td>
</tr>
<tr>
<td>Terminating user sessions</td>
<td>345</td>
</tr>
<tr>
<td>Using Administration API to terminate single user sessions</td>
<td>345</td>
</tr>
<tr>
<td>Using pdadmin to terminate all user sessions</td>
<td>345</td>
</tr>
<tr>
<td>Providing access control to dynamic URLs</td>
<td>347</td>
</tr>
<tr>
<td>Dynamic URL components</td>
<td>347</td>
</tr>
<tr>
<td>Mapping ACL and POP objects to dynamic URLs</td>
<td>347</td>
</tr>
<tr>
<td>Updating WebSEAL for dynamic URLs</td>
<td>349</td>
</tr>
<tr>
<td>Resolving dynamic URLs in the object space</td>
<td>349</td>
</tr>
<tr>
<td>ACL and POP Evaluation</td>
<td>350</td>
</tr>
<tr>
<td>Configuring limitations on POST requests</td>
<td>350</td>
</tr>
<tr>
<td>Summary and technical notes</td>
<td>351</td>
</tr>
<tr>
<td>Dynamic URL example: The Travel Kingdom</td>
<td>353</td>
</tr>
<tr>
<td>The application</td>
<td>353</td>
</tr>
<tr>
<td>The interface</td>
<td>353</td>
</tr>
<tr>
<td>Web space structure</td>
<td>353</td>
</tr>
<tr>
<td>The security policy</td>
<td>354</td>
</tr>
<tr>
<td>Dynamic URL to object space mappings</td>
<td>354</td>
</tr>
<tr>
<td>Secure clients</td>
<td>354</td>
</tr>
<tr>
<td>Account and group structure</td>
<td>354</td>
</tr>
<tr>
<td>Access control</td>
<td>354</td>
</tr>
<tr>
<td>Conclusion</td>
<td>355</td>
</tr>
</tbody>
</table>

**Chapter 13. Authorization decision information retrieval** .......................... 357

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of ADI retrieval</td>
<td>358</td>
</tr>
<tr>
<td>Retrieving ADI from the WebSEAL client request</td>
<td>359</td>
</tr>
<tr>
<td>Example: Retrieving ADI from the request header</td>
<td>360</td>
</tr>
<tr>
<td>Example: Retrieving ADI from the request query string</td>
<td>360</td>
</tr>
<tr>
<td>Example: Retrieving ADI from the request POST body</td>
<td>361</td>
</tr>
<tr>
<td>Retrieving ADI from the user credential</td>
<td>362</td>
</tr>
<tr>
<td>Supplying a failure reason across a junction</td>
<td>363</td>
</tr>
<tr>
<td>Dynamic ADI retrieval</td>
<td>364</td>
</tr>
<tr>
<td>Deploying the attribute retrieval service</td>
<td>365</td>
</tr>
</tbody>
</table>

**Chapter 14. Attribute retrieval service reference** ................................ 367

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic configuration</td>
<td>368</td>
</tr>
<tr>
<td>Configuration files.</td>
<td>368</td>
</tr>
<tr>
<td>amwebars.conf</td>
<td>368</td>
</tr>
<tr>
<td>ContainerDescriptorTable.xml</td>
<td>368</td>
</tr>
<tr>
<td>ProviderTable.xml</td>
<td>368</td>
</tr>
<tr>
<td>ProtocolTable.xml</td>
<td>368</td>
</tr>
<tr>
<td>Descriptions of amwebars.conf configuration parameters</td>
<td>368</td>
</tr>
<tr>
<td>Table locations</td>
<td>368</td>
</tr>
</tbody>
</table>
Logging .................................................. 369
Limitation of client and session number ............... 370
Miscellaneous options ................................ 370
Protocol modules to load at initialization ............... 370
Editing the data tables ................................ 371
ProviderTable ........................................... 371
  Provider sub-elements ................................ 371
Example ProviderTable ................................ 371
ContainerDescriptorTable ............................... 372
  ContainerDescriptor sub-elements .................... 372
Attribute mapping ...................................... 373
  Example ContainerDescriptorTable ................... 373
ProtocolTable .......................................... 374
  Protocol sub-elements ................................ 374
Example ProtocolTable ................................ 374
Creating custom protocol plug-ins ....................... 375
Overview ................................................ 375
Creating the protocol plug-in .......................... 375

Appendix A. WebSEAL configuration file reference ........ 377
Configuration file name and location ..................... 377
Guidelines for configuring stanzas ....................... 378
  General guidelines .................................... 378
  Default values ....................................... 379
Strings .................................................. 379
  Defined strings ...................................... 379
Lists .................................................... 379
File names .............................................. 380
Integers ............................................... 380
Boolean values ........................................ 381
Change configuration settings ............................ 381
Stanza organization .................................... 382
Server configuration ................................... 384
User registry ............................................ 393
  LDAP .................................................. 393
  Active Directory ..................................... 400
  IBM Lotus Domino ...................................... 403
Secure Socket Layer .................................... 406
Authentication .......................................... 413
  Authentication methods .............................. 414
  Authentication libraries ............................. 419
Reauthentication ....................................... 423
Authentication failover ................................ 424
Cross-domain single sign-on ............................. 428
e-community single sign-on .................. 431
Quality of protection .................................. 435
Session .................................................. 437
Content .................................................. 440
  Content management .................................. 441
Account management .................................... 443
Automatic redirect ...................................... 446
Local CGI ............................................... 447
Icons .................................................... 449
Content caching ........................................ 451
Content compression .................................... 452
Content MIME types ................................... 453
Content encodings ..................................... 454
Junctions ................................................ 455
  Junction management ................................ 456
Document filtering ..................................... 460
Event handler filtering ................................ 461
Scheme filtering .............................................. 462
MIME types and header filtering .......................... 463
Script filtering ................................................. 464
Credential refresh ............................................. 466
Header names .................................................. 466
Global Sign-On cache ........................................ 467
Lightweight Third Party Authentication cache .......... 469
Logging .......................................................... 471
Auditing ........................................................ 474
Policy database ................................................. 477
Entitlement services ........................................... 479
Policy server .................................................... 480
Platform for Privacy Preferences (P3P) ..................... 481

Appendix B. WebSEAL junction reference ................. 491
Using pdadmin to create junctions ........................ 491
The junction commands ...................................... 492
Create a new junction for an initial server ............... 493
Add a server to an existing junction ...................... 495

Appendix C. Notices ........................................... 497
Trademarks ..................................................... 499

Index ............................................................ 501
Preface

Welcome to the IBM® Tivoli® Access Manager for e-business WebSEAL Administration Guide.

IBM Tivoli Access Manager WebSEAL is the resource security manager for Web-based resources in a Tivoli Access Manager secure domain. WebSEAL is a high performance, multi-threaded Web server that applies fine-grained security policy to the protected Web object space. WebSEAL can provide single sign-on solutions and incorporate back-end Web application server resources into its security policy.

This administration guide provides a comprehensive set of procedures and reference information for managing the resources of your secure Web domain. This guide also provides you with valuable background and concept information for the wide range of WebSEAL functionality.

IBM® Tivoli® Access Manager (Tivoli Access Manager) is the base software that is required to run applications in the IBM Tivoli Access Manager product suite. It enables the integration of IBM Tivoli Access Manager applications that provide a wide range of authorization and management solutions. Sold as an integrated solution, these products provide an access control management solution that centralizes network and application security policy for e-business applications.

Note: IBM Tivoli Access Manager is the new name of the previously released software entitled Tivoli SecureWay® Policy Director. Also, for users familiar with the Tivoli SecureWay Policy Director software and documentation, the management server is now referred to as the policy server.

Who should read this book

This guide is for system administrators responsible for configuring and maintaining an Tivoli Access Manager WebSEAL environment.

Readers should be familiar with the following:
• PC and UNIX® operating systems
• Database architecture and concepts
• Security management
• Internet protocols, including HTTP, TCP/IP, File Transfer Protocol (FTP), and Telnet
• Lightweight Directory Access Protocol (LDAP) and directory services
• A supported user registry
• Authentication and authorization

If you are enabling Secure Sockets Layer (SSL) communication, you also should be familiar with SSL protocol, key exchange (public and private), digital signatures, cryptographic algorithms, and certificate authorities.

What this book contains

• Chapter 1: IBM Tivoli Access Manager WebSEAL overview
This chapter introduces you to important WebSEAL concepts and functionality such as: organizing and protecting your object space, authentication, credentials acquisition, and WebSEAL junctions.

- **Chapter 2: WebSEAL server configuration**
  This chapter is a technical reference for WebSEAL configuration tasks including: using the WebSEAL configuration file, configuring communication parameters, managing worker thread allocation, and configuring cryptographic hardware.

- **Chapter 3: WebSEAL server administration**
  This chapter is a technical reference for WebSEAL administration tasks including: managing the Web space and using custom account management pages.

- **Chapter 4: Serviceability and logging**
  This chapter describes WebSEAL support for serviceability, logging, and auditing.

- **Chapter 5: WebSEAL security policy**
  This chapter provides detailed technical procedures for customizing security policy on WebSEAL including: ACL and POP policies, quality of protection, step-up authentication policy, network-based authentication policy, three-strikes login policy, and password strength policy.

- **Chapter 6: WebSEAL authentication**
  This chapter provides configuration instructions for setting up WebSEAL to manage a variety of authentication methods including: user name and password, client-side certificates, SecurID token passcode, special HTTP header data, and multiplexing proxy agents.

- **Chapter 7: Advanced WebSEAL authentication**
  This chapter provides detailed technical procedures for setting up WebSEAL for advanced authentication methods including: switch user configuration, server-side request caching, reauthentication, and automatic redirection.

- **Chapter 8: WebSEAL key management**
  This chapter provides detailed technical procedures for setting up WebSEAL key management including: server-side and client-side certificate management, and configuring VeriSign certificate status checking.

- **Chapter 9: Cross domain single sign-on solutions**
  This chapter discusses cross domain single sign-on solutions including: CDSSO (cross-domain single sign-on) and e-community.

- **Chapter 10: WebSEAL junctions**
  This chapter is a technical reference for setting up and using WebSEAL junctions.

- **Chapter 11: Single sign-on solutions across junctions**
  This chapter discusses single sign-on solutions for the internal side of a WebSEAL proxy configuration—between the WebSEAL server and the back-end junctioned application server.

- **Chapter 12: Application integration**
  This chapter discusses a variety of WebSEAL capabilities for integrating third-party application functionality.

- **Chapter 13: Authorization decision information retrieval**
  This chapter discusses various mechanisms for obtaining authorization decision information (ADI) from WebSEAL to support the evaluation of authorization rules on protected resources.

- **Chapter 14: Attribute retrieval service reference**
This chapter discusses the administration and configuration of the attribute retrieval service.

- Appendix A: WebSEAL configuration file reference
- Appendix B: WebSEAL junction reference

Publications

Review the descriptions of the Tivoli Access Manager library, the prerequisite publications, and the related publications to determine which publications you might find helpful. After you determine the publications you need, refer to the instructions for accessing publications online.

Additional information on the IBM Tivoli Access Manager for e-business product itself can be found at:


The Tivoli Access Manager library is organized into the following categories:

- “Release information”
- “Base information”
- “Web security information” on page xx
- “Developer references” on page xx
- “Technical supplements” on page xx

Release information

- IBM Tivoli Access Manager for e-business Read This First (GI11-4155-00)
  Provides information for installing and getting started using Tivoli Access Manager.
- IBM Tivoli Access Manager for e-business Release Notes (GI11-4156-00)
  Provides late-breaking information, such as software limitations, workarounds, and documentation updates.

Base information

- IBM Tivoli Access Manager Base Installation Guide (SC32-1362-00)
  Explains how to install and configure the Tivoli Access Manager base software, including the Web Portal Manager interface. This book is a subset of IBM Tivoli Access Manager for e-business Web Security Installation Guide and is intended for use with other Tivoli Access Manager products, such as IBM Tivoli Access Manager for Business Integration and IBM Tivoli Access Manager for Operating Systems.
- IBM Tivoli Access Manager Upgrade Guide (SC32-1369-00)
  Explains how to upgrade from Tivoli SecureWay Policy Director Version 3.8 or previous versions of Tivoli Access Manager to Tivoli Access Manager Version 5.1.
- IBM Tivoli Access Manager Base Administration Guide (SC32-1360-00)
  Describes the concepts and procedures for using Tivoli Access Manager services. Provides instructions for performing tasks from the Web Portal Manager interface and by using the pdadmin command.
Web security information

- **IBM Tivoli Access Manager for e-business Web Security Installation Guide** (SC32-1361-00)
  Provides installation, configuration, and removal instructions for the Tivoli Access Manager base software, including the Web Portal Manager interface, as well as the Web Security components. This book is a superset of IBM Tivoli Access Manager Base Installation Guide.

- **IBM Tivoli Access Manager for e-business WebSEAL Administration Guide** (SC32-1359-00)
  Provides background material, administrative procedures, and technical reference information for using WebSEAL to manage the resources of your secure Web domain.

- **IBM Tivoli Access Manager for e-business IBM WebSphere Application Server Integration Guide** (SC32-1368-00)
  Provides installation, removal, and administration instructions for integrating Tivoli Access Manager with IBM WebSphere® Application Server.

- **IBM Tivoli Access Manager for e-business IBM Tivoli Identity Manager Provisioning Fast Start Guide** (SC32-1364-00)
  Provides an overview of the tasks related to integrating Tivoli Access Manager and Tivoli Identity Manager and explains how to use and install the Provisioning Fast Start collection.

- **IBM Tivoli Access Manager for e-business BEA WebLogic Server Integration Guide** (SC32-1366-00)
  Provides installation, removal, and administration instructions for integrating Tivoli Access Manager with BEA WebLogic Server.

- **IBM Tivoli Access Manager for e-business IBM WebSphere Edge Server Integration Guide** (SC32-1367-00)
  Provides installation, removal, and administration instructions for integrating Tivoli Access Manager with the IBM WebSphere Edge Server application.

- **IBM Tivoli Access Manager for e-business Plug-in for Web Servers Integration Guide** (SC32-1365-00)
  Provides installation instructions, administration procedures, and technical reference information for securing your Web domain using the plug-in for Web servers.

Developer references

- **IBM Tivoli Access Manager for e-business Authorization C API Developer Reference** (SC32-1355-00)
  Provides reference material that describes how to use the Tivoli Access Manager authorization C API and the Tivoli Access Manager service plug-in interface to add Tivoli Access Manager security to applications.

- **IBM Tivoli Access Manager for e-business Authorization Java Classes Developer Reference** (SC32-1350-00)
  Provides reference information for using the Java™ language implementation of the authorization API to enable an application to use Tivoli Access Manager security.

- **IBM Tivoli Access Manager for e-business Administration C API Developer Reference** (SC32-1357-00)
Provides reference information about using the administration API to enable an application to perform Tivoli Access Manager administration tasks. This document describes the C implementation of the administration API.

- **IBM Tivoli Access Manager for e-business Administration Java Classes Developer Reference** (SC32-1356-00)
  Provides reference information for using the Java language implementation of the administration API to enable an application to perform Tivoli Access Manager administration tasks.

- **IBM Tivoli Access Manager for e-business Web Security Developer Reference** (SC32-1358-00)
  Provides administration and programming information for the cross-domain authentication service (CDAS), the cross-domain mapping framework (CDMF), and the password strength module.

### Technical supplements

- **IBM Tivoli Access Manager for e-business Command Reference** (SC32-1354-00)
  Provides information about the command line utilities and scripts provided with Tivoli Access Manager.

- **IBM Tivoli Access Manager Error Message Reference** (SC32-1353-00)
  Provides explanations and recommended actions for the messages produced by Tivoli Access Manager.

- **IBM Tivoli Access Manager for e-business Problem Determination Guide** (SC32-1352-00)
  Provides problem determination information for Tivoli Access Manager.

- **IBM Tivoli Access Manager for e-business Performance Tuning Guide** (SC32-1351-00)
  Provides performance tuning information for an environment consisting of Tivoli Access Manager with the IBM Tivoli Directory server as the user registry.

### Related publications

This section lists publications related to the Tivoli Access Manager library.

The Tivoli Software Library provides a variety of Tivoli publications such as white papers, datasheets, demonstrations, redbooks, and announcement letters. The Tivoli Software Library is available on the Web at: [http://www.ibm.com/software/tivoli/library/](http://www.ibm.com/software/tivoli/library/)

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available, in English only, from the *Glossary* link on the left side of the Tivoli Software Library Web page [http://www.ibm.com/software/tivoli/library/](http://www.ibm.com/software/tivoli/library/)

### IBM Global Security Kit

Tivoli Access Manager provides data encryption through the use of the IBM Global Security Kit (GSKit) Version 7.0. GSKit is included on the IBM Tivoli Access Manager Base CD for your particular platform, as well as on the IBM Tivoli Access Manager Web Security CDs, the IBM Tivoli Access Manager Web Administration Interfaces CDs, and the IBM Tivoli Access Manager Directory Server CDs.

The GSKit package provides the iKeyman key management utility, `gsk7ikm`, which is used to create key databases, public-private key pairs, and certificate requests. The following document is available on the Tivoli Information Center Web site in the same section as the IBM Tivoli Access Manager product documentation:
• Secure Sockets Layer Introduction and iKeyman User’s Guide (SC32-1363-00)
  Provides information for network or system security administrators who plan to enable SSL communication in their Tivoli Access Manager environment.

IBM Tivoli Directory Server
IBM Tivoli Directory Server, Version 5.2, is included on the IBM Tivoli Access Manager Directory Server CD for the desired operating system.

Note: IBM Tivoli Directory Server is the new name for the previously released software known as:
  • IBM Directory Server (Version 4.1 and Version 5.1)
  • IBM SecureWay Directory Server (Version 3.2.2)


If you plan to use IBM Tivoli Directory Server as your user registry, see the information provided at:


IBM DB2 Universal Database
IBM DB2® Universal Database™ Enterprise Server Edition, Version 8.1 is provided on the IBM Tivoli Access Manager Directory Server CD and is installed with the IBM Tivoli Directory Server software. DB2 is required when using IBM Tivoli Directory Server, z/OS™, or OS/390® LDAP servers as the user registry for Tivoli Access Manager.

DB2 information is available at:

http://www.ibm.com/software/data/db2/

IBM WebSphere Application Server
IBM WebSphere Application Server, Advanced Single Server Edition 5.0, is included on the IBM Tivoli Access Manager Web Administration Interfaces CD for the desired operating system. WebSphere Application Server enables the support of both the Web Portal Manager interface, which is used to administer Tivoli Access Manager, and the Web Administration Tool, which is used to administer IBM Tivoli Directory Server. IBM WebSphere Application Server Fix Pack 2 is also required by Tivoli Access Manager and is provided on the IBM Tivoli Access Manager WebSphere Fix Pack CD.

For information about IBM WebSphere Application Server, see:


IBM Tivoli Access Manager for Business Integration
IBM Tivoli Access Manager for Business Integration, available as a separately orderable product, provides a security solution for IBM MQSeries®, Version 5.2, and IBM WebSphere® MQ for Version 5.3 messages. IBM Tivoli Access Manager for Business Integration allows WebSphere MQSeries applications to send data with privacy and integrity by using keys associated with sending and receiving applications. Like WebSEAL and IBM Tivoli Access Manager for Operating Systems, IBM Tivoli Access Manager for Business Integration, is one of the resource managers that use the services of IBM Tivoli Access Manager.
Additional information on IBM Tivoli Access Manager for Business Integration can be found at:


The following documents associated with IBM Tivoli Access Manager for Business Integration Version 5.1 are available on the Tivoli Information Center Web site:

- IBM Tivoli Access Manager for Business Integration Administration Guide (SC23-4831-01)
- IBM Tivoli Access Manager for Business Integration Problem Determination Guide (GC23-1328-00)
- IBM Tivoli Access Manager for Business Integration Release Notes (GI11-0957-01)
- IBM Tivoli Access Manager for Business Integration Read This First (GI11-4202-00)

**IBM Tivoli Access Manager for WebSphere Business Integration Brokers**
IBM Tivoli Access Manager for WebSphere Business Integration Brokers, available as a separately orderable product, provides a security solution for WebSphere Business Integration Message Broker, Version 5.0 and WebSphere Business Integration Event Broker, Version 5.0. IBM Tivoli Access Manager for WebSphere Business Integration Brokers operates in conjunction with Tivoli Access Manager to secure JMS publish/subscribe applications by providing password and credentials-based authentication, centrally-defined authorization, and auditing services.

Additional information on IBM Tivoli Access Manager for WebSphere Integration Brokers can be found at:


The following documents associated with IBM Tivoli Access Manager for WebSphere Integration Brokers, Version 5.1 are available on the Tivoli Information Center Web site:

- IBM Tivoli Access Manager for WebSphere Business Integration Brokers Administration Guide (SC32-1347-00)
- IBM Tivoli Access Manager for WebSphere Business Integration Brokers Release Notes (GI11-4154-00)
- IBM Tivoli Access Manager for WebSphere Business Integration Brokers Read This First (GI11-4153-00)

**IBM Tivoli Access Manager for Operating Systems**
IBM Tivoli Access Manager for Operating Systems, available as a separately orderable product, provides a layer of authorization policy enforcement on UNIX systems in addition to that provided by the native operating system. IBM Tivoli Access Manager for Operating Systems, like WebSEAL and IBM Tivoli Access Manager for Business Integration, is one of the resource managers that use the services of IBM Tivoli Access Manager.

Additional information on IBM Tivoli Access Manager for Operating Systems can be found at:

The following documents associated with IBM Tivoli Access Manager for Operating Systems Version 5.1 are available on the Tivoli Information Center Web site:

- IBM Tivoli Access Manager for Operating Systems Installation Guide (SC23-4829-00)
- IBM Tivoli Access Manager for Operating Systems Administration Guide (SC23-4827-00)
- IBM Tivoli Access Manager for Operating Systems Problem Determination Guide (SC23-4828-00)
- IBM Tivoli Access Manager for Operating Systems Release Notes (GI11-0951-00)
- IBM Tivoli Access Manager for Operating Systems Read Me First (GI11-0949-00)

**IBM Tivoli Identity Manager**

IBM Tivoli Identity Manager Version 4.5, available as a separately orderable product, when used with the Fast Start Provisioning support provided with IBM Tivoli Access Manager.

Additional information on IBM Tivoli Identity Manager can be found at:


The following documents associated with IBM Tivoli Identity Manager Version 4.5 are available on the Tivoli Information Center Web site:

- IBM Tivoli Identity Manager Release Notes (GI11-4212-00)
- IBM Tivoli Identity Manager Server Installation Guide on UNIX using WebSphere (SC32-1147-02)
- IBM Tivoli Identity Manager Server Installation Guide on Windows 2000 using WebSphere (SC32-1148-01)
- IBM Tivoli Identity Manager Server Installation Guide on UNIX using WebLogic (SC32-1334-00)
- IBM Tivoli Identity Manager Server Installation Guide on Windows 2000 using WebLogic (SC32-1335-00)
- IBM Tivoli Identity Manager Policy and Organization Administration Guide (SC32-1149-01)
- IBM Tivoli Identity Manager End User Guide (SC32-1152-01)
- IBM Tivoli Identity Manager Server Configuration Guide (SC32-1150-02)
- IBM Tivoli Identity Manager Server Troubleshooting Guide (SC32-1151-01)
- IBM Tivoli Identity Manager Access Manager Agent for Windows Installation Guide (SC32-1165-03)
- IBM Tivoli Identity Manager Lotus Notes Agent Installation Guide (SC32-1157-03)
- IBM Tivoli Identity Manager Sybase Agent for Windows Installation Guide (SC32-1161-03)
- IBM Tivoli Identity Manager Oracle Agent for Windows Installation Guide (SC32-1155-03)
- IBM Tivoli Identity Manager Windows 2000 Agent Installation Guide (SC32-1153-03)
- IBM Tivoli Identity Manager Windows NT Agent Installation Guide (SC32-1154-03)
- IBM Tivoli Identity Manager AIX Agent Installation Guide (SC32-1162-03)
- IBM Tivoli Identity Manager Exchange 2000 Agent Installation Guide (SC32-1156-03)
- IBM Tivoli Identity Manager Novell Network Agent Installation Guide (SC32-1158-03)
- IBM Tivoli Identity Manager Universal Provisioning Agent Installation Guide (SC32-1159-03)
Accessing publications online

The publications for this product are available online in Portable Document Format (PDF) or Hypertext Markup Language (HTML) format, or both in the Tivoli software library: [http://www.ibm.com/software/tivoli/library](http://www.ibm.com/software/tivoli/library)

To locate product publications in the library, click the **Product manuals** link on the left side of the library page. Then, locate and click the name of the product on the Tivoli software information center page.

Product publications include release notes, installation guides, user’s guides, administrator’s guides, and developer’s references.

**Note:** To ensure proper printing of PDF publications, select the **Fit to page** check box in the Adobe Acrobat Print window (which is available when you click **File → Print**).

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You also can use the keyboard instead of the mouse to operate all features of the graphical user interface.

Contacting software support


If you need additional help, contact software support by using the methods described in the *IBM Software Support Guide* at the following Web site: [http://techsupport.services.ibm.com/guides/handbook.html](http://techsupport.services.ibm.com/guides/handbook.html)

The guide provides the following information:

- Registration and eligibility requirements for receiving support
- Telephone numbers, depending on the country in which you are located
- A list of information you should gather before contacting customer support

Conventions used in this book

This reference uses several conventions for special terms and actions and for operating system-dependent commands and paths.

**Typeface conventions**

The following typeface conventions are used in this reference:

- **Bold** Lowercase commands or mixed case commands that are difficult to distinguish from surrounding text, keywords, parameters, options, names of Java classes, and objects are in **bold**.
- **Italic** Variables, titles of publications, and special words or phrases that are emphasized are in **italic**.
Monospace
Code examples, command lines, screen output, file and directory names that are difficult to distinguish from surrounding text, system messages, text that the user must type, and values for arguments or command options are in monospace.

Operating system differences
This book uses the UNIX convention for specifying environment variables and for directory notation. When using the Windows command line, replace $variable with %variable% for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. If you are using the bash shell on a Windows system, you can use the UNIX conventions.
Chapter 1. IBM Tivoli Access Manager WebSEAL overview

IBM® Tivoli® Access Manager for e-business (Tivoli Access Manager) is a robust and secure centralized policy management solution for e-business and distributed applications. IBM Tivoli Access Manager WebSEAL is a high performance, multi-threaded Web server that applies fine-grained security policy to the Tivoli Access Manager protected Web object space. WebSEAL can provide single sign-on solutions and incorporate back-end Web application server resources into its security policy.

This overview chapter introduces you to the main capabilities of the WebSEAL server.

Topic Index:
- “Introducing IBM Tivoli Access Manager and WebSEAL” on page 1
- “Understanding the Tivoli Access Manager security model” on page 2
- “Protecting the Web space with WebSEAL” on page 5
- “Planning and implementing the security policy” on page 6
- “Understanding WebSEAL authentication” on page 8
- “Understanding WebSEAL junctions” on page 11

Introducing IBM Tivoli Access Manager and WebSEAL

IBM Tivoli Access Manager:

IBM Tivoli Access Manager is a complete authorization and network security policy management solution that provides unsurpassed end-to-end protection of resources over geographically dispersed intranets and extranets.

In addition to its state-of-the-art security policy management feature, Tivoli Access Manager provides authentication, authorization, data security, and centralized resource management capabilities. You use Tivoli Access Manager in conjunction with standard Internet-based applications to build highly secure and well-managed intranets.

At its core, Tivoli Access Manager provides:
- Authentication framework
  Tivoli Access Manager provides a wide range of built-in authenticators and supports external authenticators.
- Authorization framework
  The Tivoli Access Manager authorization service, accessed through the Tivoli Access Manager authorization API, provides permit and deny decisions on requests for protected resources located in the secure domain.

With Tivoli Access Manager, businesses can securely manage access to private internal network-based resources while leveraging the public Internet’s broad connectivity and ease of use. Tivoli Access Manager, in combination with a corporate firewall system, can fully protect the Enterprise intranet from unauthorized access and intrusion.
IBM Tivoli Access Manager WebSEAL:

IBM Tivoli Access Manager WebSEAL is the resource manager responsible for managing and protecting Web-based information and resources.

WebSEAL is a high performance, multi-threaded Web server that applies fine-grained security policy to the Tivoli Access Manager protected Web object space. WebSEAL can provide single sign-on solutions and incorporate back-end Web application server resources into its security policy.

WebSEAL normally acts as a reverse Web proxy by receiving HTTP/HTTPS requests from a Web browser and delivering content from its own Web server or from junctioned back-end Web application servers. Requests passing through WebSEAL are evaluated by the Tivoli Access Manager authorization service to determine whether the user is authorized to access the requested resource.

WebSEAL provides the following features:
- Supports multiple authentication methods
  Both built-in and plug-in architectures allow flexibility in supporting a variety of authentication mechanisms.
- Accepts HTTP and HTTPS requests
- Integrates and protects back-end server resources through WebSEAL junction technology
- Manages fine-grained access control for the local and back-end server Web space
  Supported resources include URLs, URL-based regular expressions, CGI programs, HTML files, Java servlets, and Java class files.
- Performs as a reverse Web proxy
  WebSEAL appears as a Web server to clients and appears as a Web browser to the junctioned back-end servers it is protecting.
- Provides single sign-on capabilities

Understanding the Tivoli Access Manager security model

The security policy for a Tivoli Access Manager secure domain is maintained and governed by two key security structures:
- **User registry**
  The user registry (such as LDAP, Lotus Domino, or Microsoft Active Directory) contains all users and groups who are allowed to participate in the Tivoli Access Manager secure domain.
- **Master authorization (policy) database**
  The authorization database contains a representation of all resources in the domain (the protected object space). The security administrator can dictate any level of security by applying rules, known as access control list (ACL) policies and protected object policies (POP), to those resources requiring protection.

The process of authentication proves the identity of a user to WebSEAL. A user can participate in the secure domain as authenticated or unauthenticated. Only users with an entry in the user registry can become authenticated users. Using ACLs and POPs, the security administrator can make certain public resources available to unauthenticated users. Other resources can be made available only to certain authenticated users.
When a user successfully authenticates to WebSEAL, a set of identification information—known as a credential—is created for that user. The credential contains the user identity, any group memberships, and any special (“extended”) security attributes.

The Tivoli Access Manager authorization service enforces security policies by comparing a user’s authentication credentials with the policy permissions assigned to the requested resource. The resulting recommendation is passed to the resource manager (for example, WebSEAL), which completes the response to the original request. The user credential is essential for full participation in the secure domain.

**The protected object space**

The protected object space is a hierarchical portrayal of resources belonging to a Tivoli Access Manager secure domain. The virtual objects that appear in the hierarchical object space represent the actual physical network resources.

- **System resource** – the actual physical file or application.
- **Protected object** – the logical representation of an actual system resource used by the authorization service, the Web Portal Manager, and other Tivoli Access Manager management utilities.

Policy templates can be attached to objects in the object space to provide protection of the resource. The authorization service makes authorization decisions based on these templates.

The following object space categories are used by Tivoli Access Manager:

- **Web objects**
  Web objects represent any resource that can be addressed by an HTTP URL. This includes static Web pages and dynamic URLs that are converted to database queries or some other type of application. The WebSEAL server is responsible for protecting Web objects.

- **Tivoli Access Manager management objects**
  Management objects represent the management activities that can be performed through the Web Portal Manager. The objects represent the tasks necessary to define users and set security policy. Tivoli Access Manager supports delegation of management activities and can restrict an administrator’s ability to set security policy to a subset of the object space.

- **User-defined objects**
  User-defined objects represent customer-defined tasks or network resources protected by applications that access the authorization service through the Tivoli Access Manager authorization API.

**Defining and applying ACL and POP policies**

Security administrators protect Tivoli Access Manager system resources by defining rules, known as ACL and POP policies, and applying these policies to the object representations of those resources in the protected object space.

The Tivoli Access Manager authorization service performs authorization decisions based on the policies applied to these objects. When a requested operation on a protected object is permitted, the application responsible for the resource implements this operation.

One policy can dictate the protection parameters of many objects. Any change to the rule will affect all objects to which the template is attached.
The access control list (ACL)
An access control list policy, or ACL policy, is the set of rules (permissions) that specifies the conditions necessary to perform certain operations on that resource. ACL policy definitions are important components of the security policy established for the secure domain. ACL policies, like all policies, are used to stamp an organization’s security requirements onto the resources represented in the protected object space.

An ACL policy specifically controls:
1. What operations can be performed on the resource
2. Who can perform these operations

An ACL policy is made up of one or more entries that include user and group designations and their specific permissions or rights. An ACL can also contain rules that apply to unauthenticated users.

Protected object policies (POP)
ACL policies provide the authorization service with information to make a "yes" or "no" answer on a request to access a protected object and perform some operation on that object.

POP policies contain additional conditions on the request that are passed back to Tivoli Access Manager Base and the resource manager (such as WebSEAL) along with the "yes" ACL policy decision from the authorization service. It is the responsibility of Tivoli Access Manager and the resource manager to enforce the POP conditions.

The following tables list the available attributes for a POP:

<table>
<thead>
<tr>
<th>Enforced by Tivoli Access Manager Base</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP Attribute</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Name of the policy. This becomes the &lt;pop-name&gt; in the pdadmin pop commands.</td>
</tr>
<tr>
<td>Description</td>
<td>Descriptive text for the policy. This appears in the pop show command.</td>
</tr>
<tr>
<td>Warning Mode</td>
<td>Provides administrators a means to test ACL and POP policies.</td>
</tr>
<tr>
<td>Audit Level</td>
<td>Specifies type of auditing: all, none, successful access, denied access, errors.</td>
</tr>
<tr>
<td>Time-of-Day Access</td>
<td>Day and time restrictions for successful access to the protected object.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enforced by Resource Manager (such as WebSEAL)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POP Attribute</td>
<td></td>
</tr>
<tr>
<td>Quality of Protection</td>
<td>Specifies degree of data protection: none, integrity, privacy.</td>
</tr>
<tr>
<td>IP Endpoint Authentication Method Policy</td>
<td>Specifies authentication requirements for access from members of external networks.</td>
</tr>
<tr>
<td>Document Cache Control</td>
<td>Specify caching instructions for the handling of specific documents.</td>
</tr>
</tbody>
</table>
Explicit and inherited policy
Policy can be explicitly applied or inherited. The Tivoli Access Manager protected object space supports inheritance of ACL and POP attributes. This is an important consideration for the security administrator who manages the object space. The administrator needs to apply explicit policies only at points in the hierarchy where the rules must change.

Policy administration: The Web Portal Manager
The Web Portal Manager is a Web-based graphical application used to manage security policy in a Tivoli Access Manager secure domain. The \texttt{pdadmin} command line utility provides the same administration capabilities as the Web Portal Manager, plus some commands not supported by the Web Portal Manager.

From the Web Portal Manager (or \texttt{pdadmin}), you can manage the user registry, the master authorization policy database, and the Tivoli Access Manager servers. You can also add and delete users/groups and apply ACL and POP policies to network objects.

Protecting the Web space with WebSEAL
When WebSEAL enforces security in a secure domain, each client must provide proof of its identity. In turn, Tivoli Access Manager security policy determines whether that client is permitted to perform an operation on a requested resource. Because access to every Web resource in a secure domain is controlled by WebSEAL, WebSEAL’s demands for authentication and authorization can provide comprehensive network security.

In security systems, authorization is distinct from authentication. Authorization determines whether an authenticated client has the right to perform an operation on a specific resource in a secure domain. Authentication can validate the identity of a client, but says nothing about the client’s right to perform operations on a protected resource.

In the Tivoli Access Manager authorization model, authorization policy is implemented independently of the mechanism used for user authentication. Users can authenticate their identity using either public/private key, secret key, or customer-defined mechanisms.

Part of the authentication process involves the creation of a credential that describes the identity of the client. Authorization decisions made by an authorization service are based on user credentials.

The resources in a secure domain receive a level of protection as dictated by the security policy for the domain. The security policy defines the legitimate participants of the secure domain and the degree of protection surrounding each resource requiring protection.

The authorization process consists of the following basic components:

- A \textbf{resource manager} is responsible for implementing the requested operation when authorization is granted. WebSEAL is a resource manager.

  A component of the resource manager is a \textbf{policy enforcer} that directs the request to the authorization service for processing.
Note: Traditional applications bundle the policy enforcer and resource manager into one process. Examples of this structure include WebSEAL and third-party applications.

- An authorization service performs the decision-making action on the request.

The following diagram illustrates the complete authorization process:

![Diagram of the Tivoli Access Manager authorization process]

Figure 1. The Tivoli Access Manager authorization process

1. An authenticated client request for a resource is directed to the resource manager and intercepted by the policy enforcer process. The resource manager can be WebSEAL (for HTTP, HTTPS access) or a third-party application.
2. The policy enforcer process uses the Tivoli Access Manager authorization API to call the authorization service for an authorization decision.
3. The authorization service performs an authorization check on the resource, represented as an object in the protected object space. Base POP policies are checked first. Next the ACL policy attached to the object is checked against the client’s credentials. Then, POP policies enforced by the resource manager are checked.
4. The decision to accept or deny the request is returned as a recommendation to the resource manager (through the policy enforcer).
5. If the request is finally approved, the resource manager passes the request on to the application responsible for the resource.
6. The client receives the results of the requested operation.

Planning and implementing the security policy

A corporate security policy identifies:

- The Web resources requiring protection
- The level of protection

Tivoli Access Manager uses a virtual representation of these Web resources, called the protected object space. The protected object space contains objects that represent actual physical resources in your network.
You implement security policy by applying the appropriate security mechanisms to the objects requiring protection.

Security mechanisms include:

- **Access control list (ACL) policies**
  
  ACL policies identify user types that can be considered for access and specify the operations permitted on the object.

- **Protected object policies (POP)**
  
  A POP specifies additional conditions governing the access to the protected object, such as privacy, integrity, auditing, and time-of-day access.

- **Extended attributes**
  
  Extended attributes are additional values placed on an object, ACL, or POP that can be read and interpreted by third-party applications (such as an external authorization service).

The core component of Tivoli Access Manager is the Tivoli Access Manager authorization service—which permits or denies access to protected objects (resources) based on the user’s credentials and the access controls placed on the objects.

To successfully implement the security policy, you must logically organize the different content types (as described in “Identifying content types and levels of protection” on page 7) and apply the appropriate ACL and POP policies. Access control management can be very complex and is made much easier by careful categorization of the content types.

## Identifying content types and levels of protection

As the security administrator of your Web space, you must correctly identify the types of content available to a variety of user types. Some content must be highly protected and available only to specific users; other content is for general public view. Each security scenario demands different protection requirements and the associated WebSEAL configuration.

It is your responsibility to:

- Know your Web content
- Identify the types of users requiring access to this content
- Understand the strengths and weaknesses of the available WebSEAL configuration options for securing this content

Protection of Web content falls into three broad categories:

1. **Public content – access requires no protection**
   - Unauthenticated clients access using HTTP
   - Unauthenticated credential used for access control to resources
   - Basic WebSEAL configuration requirements

2. **Public content – access requires privacy (encryption)**
   - Unauthenticated clients access using HTTPS
   - Encryption required to protect sensitive data required by application server (such as credit card numbers and user account information)
   - Unauthenticated credential used for access control to resources
   - WebSEAL configuration needs to stipulate privacy

3. **Private content – access requires authentication**
• Authenticated clients access using HTTP or HTTPS
• Administrator determines the need for encryption
• Authenticated credential used for access control to resources; clients must have account defined in user registry
• WebSEAL configuration is complex and all options must be considered carefully to determine impact of security policy

Understanding WebSEAL authentication

Authentication is the method of identifying an individual process or entity attempting to login to a secure domain. When both server and client require authentication, the exchange is known as mutual authentication.

WebSEAL can enforce a high degree of security in a secure domain by requiring each client to provide proof of its identity.

The following conditions apply to WebSEAL authentication:
• WebSEAL supports a standard set of authentication methods.
  You can customize WebSEAL to support other authentication methods.
• The WebSEAL server process is independent of the authentication method.
• WebSEAL requires a client identity. From this identity, WebSEAL builds an authenticated (or unauthenticated) credential that can be used by the Tivoli Access Manager authorization service to permit or deny access to resources.

This flexible approach to authentication allows security policy to be based on business requirements and not physical network topology.

The goals of authentication

Although WebSEAL is independent of the authentication process, WebSEAL requires the results of authentication—the client identity. The authentication process results in the following actions:

1. The authentication method results in a client identity
   Client authentication is successful only if the user has an account defined in the Tivoli Access Manager user registry or is processed successfully by a Cross-domain Authentication Service (CDAS). Otherwise the user is designated as unauthenticated.
   Method-specific identity information, such as passwords, tokens, and certificates, represent physical identity properties of the user. This information can be used to establish a secure session with the server.

2. WebSEAL uses the identity to acquire credentials for that client
   WebSEAL matches the client identity with a registered Tivoli Access Manager user. WebSEAL then builds the credentials appropriate to this user. This is known as credentials acquisition.
   The credential represents a user’s privileges in the secure domain, describes the user in a specific context, and is valid only for the lifetime of that session.
   Credential data includes the user name, any group memberships, and any special “extended” security attributes.
   If a user is not a member of the user registry (“anonymous”), WebSEAL builds an unauthenticated credential for that user. Remember that an ACL can contain special rules governing unauthenticated users.
   These credentials are available to the authorization service that permits or denies access to requested objects in the WebSEAL protected object space.
Credentials can be used by any Tivoli Access Manager service that requires information about the client. Credentials allow Tivoli Access Manager to securely perform a multitude of services such as authorization, auditing, and delegation.

Tivoli Access Manager distinguishes the authentication of the user from the acquisition of credentials. A user’s identity is always constant. However, credentials—which define the groups or roles in which a user participates—are variable. Context-specific credentials can change over time. For example, when a person is promoted, credentials must reflect the new responsibility level.

See Chapter 6, “Authentication,” on page 129 for further information about support for specific authentication methods.

**Authenticated and unauthenticated access to resources**

In a Tivoli Access Manager secure domain environment, the identity of a user is proven to WebSEAL through the process of authentication. In general, a user can participate in the secure domain as authenticated or unauthenticated.

In either case, the Tivoli Access Manager authorization service requires a user credential to make authorization decisions on requests for resources in the secure domain. WebSEAL handles authenticated user credentials differently from unauthenticated user credentials.

The credential for an unauthenticated user is simply a generic passport that allows the user to participate in the secure domain and access resources that are available to unauthenticated users.

The credential for an authenticated user is a unique passport that describes a specific user who belongs to the Tivoli Access Manager user registry (or is processed successfully by a CDAS). The authenticated user credential contains the user identity, any group memberships, and any special (“extended”) security attributes.

The process flow for authenticated users:
- A user makes a request for a resource protected by WebSEAL. The protection on the resource requires that the user be authenticated. WebSEAL prompts the user to log in.
- Successful authentication can occur only if the user is a member of the Tivoli Access Manager user registry or is handled by a CDAS operation.
- A WebSEAL session ID is created for the user.
- A credential for this user is built from information contained in the registry about this user (such as group memberships).
- The session ID and credential, plus other data, is stored as an entry in the WebSEAL session/credentials cache.
- As WebSEAL processes this request (and future requests during this session), it keeps the credential information available.
- Whenever an authorization check is required, the Tivoli Access Manager authorization service uses the credential information during the decision-making process.
- When the user logs off, the cache entry for that user is removed and the session is terminated.

The process flow for unauthenticated users:
A user makes a request for a resource protected by WebSEAL. The protection on the resource does not require that the user be authenticated. WebSEAL does not prompt the user to log in.

WebSEAL builds an unauthenticated credential for the user.

No entry is created in the WebSEAL session/credentials cache.

The user can access resources that contain the correct permissions for the unauthenticated type category of user.

If the user requires access to a resource not available to unauthenticated users, WebSEAL prompts the user to log in.

A successful log in changes the user’s status to authenticated.

If log in is unsuccessful, a 403 "Forbidden" message is returned. The user can still continue to access other resources that are available to unauthenticated users.

The WebSEAL session/credentials cache structure

The WebSEAL session cache is also known as the WebSEAL credentials cache. The cache can be represented as an internal table where WebSEAL stores information about all sessions established by authenticated users.

Each user session is represented by an entry in the cache table.

Each cache entry contains the following types of information:

• **Session ID**
  
The session ID is a unique identifier that is sent with each request made by that user. The session ID identifies the specific cache entry for that user.

• **Cache data**
  
The most important data stored in the cache entry is the user credential. The credential is required whenever the user requests protected resources. The authorization service uses the credential information to permit or deny access to the resource.

  WebSEAL can mark, or "flag", a cache entry to support certain functionality. For example, when session inactivity reauthentication is enabled, a cache entry is "flagged" when the session inactivity value has expired.

• **Timestamps**
  
The creation timestamp for the cache entry becomes the reference point for the session lifetime value. The "last active" timestamp for the cache entry becomes the reference point for the session inactivity timer.

The user credential contains:

• User name

• Group memberships

• Extended attributes

Extended attributes allow you to store customized data in the user credential. An example of a credential extended attribute is the `tagvalue_user_session_id` attribute. The value of this attribute can be inserted in an HTTP header to allow a back-end junctioned server to maintain session state with the user.
Understanding WebSEAL junctions

Tivoli Access Manager provides authentication, authorization, and management services for a network. In a Web-based network, these services are best provided by one or more front-end WebSEAL servers that integrate and protect Web resources and applications located on back-end Web servers.

The connection between a WebSEAL server and a back-end Web application server is known as a WebSEAL junction. A WebSEAL junction is a TCP/IP connection between a front-end WebSEAL server and a back-end server.

The back-end server can be another WebSEAL server or, more commonly, a third-party Web application server. The back-end server Web space is "connected" to the WebSEAL server at a specially designated junction (mount) point in the WebSEAL Web space.

![Diagram of WebSEAL junctions](image)

**Figure 2. Junctions connect WebSEAL with back-end servers**

A junction allows WebSEAL to provide protective services on behalf of the back-end server. WebSEAL can perform authentication and authorization checks on all requests before passing those requests on to the back-end server. If the back-end server requires fine-grained access control on its objects, you must perform additional configuration steps to describe the third-party Web space to the Tivoli Access Manager security service (see "Using query_contents with third-party servers" on page 309).

Junctions provide a scalable, secure environment that allows load balancing, high availability, and state management capabilities—all performed transparently to clients. As an administrator, you can benefit from this centralized management of the Web space.

WebSEAL junctions provide the added value of logically combining the Web space of a back-end server with the Web space of the WebSEAL server. Junctions between cooperating servers result in a single, unified, distributed Web space that is seamless and transparent to users.

The client never needs to know the physical location of a Web resource. WebSEAL translates logical URL addresses into the physical addresses that a back-end server expects. Web objects can be moved from server to server without affecting the way the client accesses those objects.
A unified Web space simplifies the management of all resources for the system administrator. Additional administrative benefits include scalability, load balancing, and high availability.

Most commercial Web servers do not have the ability to define a logical Web object space. Instead, their access control is connected to the physical file and directory structure. WebSEAL junctions can transparently define an object space that reflects organizational structure rather than the physical machine and directory structure commonly encountered on standard Web servers.

WebSEAL junctions also allow you to create single sign-on solutions. A single sign-on configuration allows a user to access a resource, regardless of the resource’s location, using only one initial login. Any further login requirements from back-end servers are handled transparently to the user.

WebSEAL junctions are an important tool for making your Web site scalable. Junctions allow you to respond to increasing demands on a Web site by attaching additional servers.

**WebSEAL junctions and Web site scalability**

WebSEAL junctions are used to create a scalable Web site. As the demands on the Web site grow, more servers can easily be added to expand the capabilities of the site.

Additional servers can be added for the following reasons:
• To extend the site with additional content
• To duplicate existing content for load balancing, failover capability, and high availability

**Replicated front-end WebSEAL servers**
Junction support for back-end servers starts with at least one front-end WebSEAL server. Replicated front-end WebSEAL servers provide the site with load balancing during periods of heavy demand. The load balancing mechanism is handled by a mechanism such as IBM Network Dispatcher or Cisco Local Director.

Front-end replication also provides the site with fail-over capability—if a server fails for some reason, the remaining replica servers will continue to provide access to the site. Successful load balancing and failover capability results in high availability for users of the site.

When you replicate front-end WebSEAL servers, each server must contain an exact copy of the Web space and the junction database.

Account information for authentication resides in a user registry that is independent of the front-end servers.

**Supporting back-end servers**
Web site content can be served by the WebSEAL server itself, back-end servers, or a combination of both. WebSEAL junction support for back-end servers allows you to scale the Web site through additional content and resources.

Each unique back-end server must be junctioned to a separate junction (mount) point. As the demand for additional content grows, more servers can be added through junctions. This scenario provides a solution for networks that have a large existing investment in third-party Web servers.

Junctions provide a unified, logical object space. This Web space is transparent to the user and allows for centralized management.

**Replicated back-end servers**
To extend scalability features to a back-end server configuration, you can replicate the back-end servers. As is the case with replicated front-end servers, replicated back-end servers must contain Web spaces that are mirror images of each other.

WebSEAL load balances across the replicated servers using a “least-busy” scheduling algorithm. This algorithm directs each new request to the server with the fewest connections already in progress.

WebSEAL also correctly fails-over when a server is down and starts re-using that server after it has been restarted.

If the back-end application requires its state to be maintained over several pages, stateful junctions can be used to ensure that each session returns to the same back-end server.
Chapter 2. WebSEAL server configuration

This chapter contains information that describes configuration tasks you can perform to customize the WebSEAL server for your network.

Topic Index:
- “Server instance configuration” on page 16
- “Communication protocol configuration” on page 28
- “Cryptographic hardware for encryption and key storage” on page 32
- “Quality of protection levels” on page 38
- “Configuring authorization database updates and polling” on page 40
- “Managing worker thread allocation” on page 41
- “Multi-locale support with UTF-8” on page 44
- “Multi-locale messages” on page 53
- “Handling invalid character encoding in URL query strings” on page 55
- “Preventing vulnerability caused by cross-site scripting” on page 56
- “Replicated front-end WebSEAL servers” on page 58
- “Platform for Privacy Preferences (P3P)” on page 59
- “Suppressing server identity” on page 69
Server instance configuration

This section contains the following topics:

- “Server instance configuration overview”
- “Server instance configuration tasks” on page 24

Server instance configuration overview

Read each topic in this overview before configuring a WebSEAL server instance.

This section contains the following topics:

- “Planning a server instance configuration”
- “Example server instance configuration values” on page 20
- “Unique configuration file for each instance” on page 21
- “Interactive configuration overview” on page 21
- “Command line configuration overview” on page 21
- “Silent configuration overview” on page 23

Planning a server instance configuration

To configure a WebSEAL instance, you must decide how to deploy the server for your environment, and you must collect some information about the Tivoli Access Manager deployment.

Unless stated otherwise, each of the following settings is required.

- **Administrative user ID and password**
  This is the Tivoli Access Manager administrative user. By default, this is the sec_master user. You must have administrative user permissions to configure a WebSEAL server instance.

- **Host name**
  The name by which the system is known on the network. Typically this is expressed as a fully qualified domain name. During interactive installations, you can alternatively provide just the system name.

  Fully qualified domain name:
  diamond.subnet2.ibm.com

  System name:
  diamond

- **Instance name**
  A unique name that identifies the WebSEAL server. Multiple WebSEAL servers can be installed on one computer system. Each must have a unique name. Names can consist of alphanumeric characters ([A-Z][a-z][0–9]) plus the following characters: underscore ( _ ), hyphen ( - ), and period ( . ). No other characters are valid. Names must not exceed 20 characters in length.

  Example names: web1, web2, web_3, web-4, web.5

  The initial WebSEAL server, which is configured during installation and configuration of WebSEAL, is assigned by default an instance name of **default**. However, this name could have been modified by the administrator during initial installation and configuration.

  The choice of instance name will be viewable after configuration. For example, the file name for the configuration file for the WebSEAL server instance is created as follows:

  `webseald-instance_name.conf`
The instance name also affects how the server is listed during a **pdadmin server list** command. For this command, the server name is constructed as follows:

```
instance_name-webseald-hostname
```

For example, an `instance_name` of `web1` installed on a host named `diamond` would have a server name:

```
web1-webseald-diamond
```

- **Listening port**
  This is the port through which the WebSEAL server communicates with the Tivoli Access Manager policy server. The default port number is 7234. This port number must be unique for every WebSEAL server instance.
  The default port is typically used by the default (first) WebSEAL server instance. The interactive installation automatically increments to the next available port.
  You can modify the port number if necessary.
  When installing using the command line or from a response file, specify another port. Any port number above 1024 is valid. Select a port that is not used for any other purpose. A common configuration selection is to increment the port number by one.

- **HTTP protocol and HTTP port**
  Specifies whether to accept request across the HTTP protocol. If HTTP requests are accepted, the administrator must assign a port number. The default port number is 80. This port is used by the default (first) instance.
  When not using a logical network interface, specify another port number. Select a port that is not used for any other purpose. A common configuration selection is to increment the port number by one. For example, 81.
  When using a logical network interface, you can use the same port number (for example, 80).

- **HTTPS protocol and HTTPS port**
  Specifies whether to accept request across the HTTPS protocol. If HTTPS requests are accepted, the administrator must assign a port number. The default port number is 443. This port is used by the default (first) instance.
  When not using a logical network interface, specify another port number. Select a port that is not used for any other purpose. A common configuration selection is to increment the port number by one. For example, 444.
  When using a logical network interface, you can use the same port number (for example, 443).

- **Logical network interface and IP address**
  This setting is optional. You can choose to use a logical network interface for the WebSEAL server instance. This means that the WebSEAL server receives a unique IP address. Use of this feature requires network hardware support for more than one IP address.
  When the networking hardware supports more than one IP address, you can specify a separate IP address for each WebSEAL server instance.
  It is not necessary to specify a separate IP address. All WebSEAL server instances can share one IP address. With this configuration, however, each WebSEAL server must listen on unique ports for HTTP and HTTPS access.
  For example, the configuration settings for two WebSEAL instances that shared an IP address might be:

<table>
<thead>
<tr>
<th>Instance</th>
<th>IP address</th>
<th>HTTP port</th>
<th>HTTPS port</th>
</tr>
</thead>
</table>

*Table 1. WebSEAL instances sharing an IP address*
Table 1. WebSEAL instances sharing an IP address (continued)

<table>
<thead>
<tr>
<th>Instance</th>
<th>IP address</th>
<th>HTTP port</th>
<th>HTTPS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>1.2.3.4</td>
<td>80</td>
<td>443</td>
</tr>
<tr>
<td>web1</td>
<td>1.2.3.4</td>
<td>81</td>
<td>444</td>
</tr>
</tbody>
</table>

For example, the configuration settings for two WebSEAL instances with unique IP addresses might be:

Table 2. WebSEAL instances with unique IP addresses

<table>
<thead>
<tr>
<th>Instance</th>
<th>IP address</th>
<th>HTTP port</th>
<th>HTTPS port</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>1.2.3.4</td>
<td>80</td>
<td>443</td>
</tr>
<tr>
<td>web1</td>
<td>1.2.3.5</td>
<td>80</td>
<td>443</td>
</tr>
</tbody>
</table>

Assigning an IP address to the default WebSEAL server instance

In one scenario, the administrator must manually assign an IP address to the default WebSEAL server instance before using amwebcfg to assign an IP address to a new WebSEAL server instance. The scenario occurs when the following conditions are true:

- When the first (default) WebSEAL instance was configured, the administrator chose not to use a logical network interface.
- When configuring a new WebSEAL instance, the administrator wants to use a logical network interface.
- When configuring a new WebSEAL instance, the administrator wants to use the same port for HTTP, or the same port for HTTPS, with each logical network interface.

Note that this scenario occurs because when the first (default) WebSEAL instance is configured not to use a logical network interface, WebSEAL is configured to listen on all IP addresses on the specified ports (HTTP, HTTPS). Thus, in order to add WebSEAL server instances that listen on the same ports (for example, 80 for HTTP and 443 for HTTPS), the first (default) WebSEAL instance must be reconfigured to receive an unique IP address.

The reconfiguration is simple. The administrator must edit the WebSEAL configuration file for the default instance, and specify an IP address for the default instance. The WebSEAL configuration file for the default instance is webseald-default.conf.

For example, using the default server instance shown in the table above, the following entry must be added to the configuration file:

```
[server]
network-interface = 1.2.3.4
```

The WebSEAL server must then be stopped and restarted.

Note that the change to the configuration file is needed only once. It is not needed when each additional server instance is configured.

- SSL communication with LDAP server

WebSEAL communicates with the LDAP server during authentication procedures. Use of SSL during communication with the LDAP server is optional. However, use of SSL is highly recommended for security reasons in all production deployments. Disabling of SSL usage can be considered for temporary testing or prototyping environments.
Note: This step is specific to use of an LDAP user registry. This step is not required when using other registry types.

If you want to use secure SSL communication between a WebSEAL instance and the LDAP registry server, you must use the LDAP SSL key file for this purpose. This is the key file that was created and distributed during installation of the LDAP client. If the initial WebSEAL server is set up to use secure SSL communication with LDAP, multiple instances can use the same key file.

When enabling SSL communication between WebSEAL and the LDAP server, you must provide the following information:

- **SSL key file name**
  The file that contains the LDAP SSL certificate.

- **SSL key file password**
  The password necessary to access the LDAP SSL key file.

- **SSL Certificate label**
  The LDAP client certificate label. This is optional. When the client label is not specified, the default certificate contained in the keyfile is used. Specify the client label when the keyfile contains more than one certificate, and the certificate to be used is not the default certificate.

- **SSL LDAP server port number**
  The port number through which to communicate with the LDAP server. The default LDAP server port number is 636.

**Web document root directory**

The root directory of the hierarchy where the resources (protected objects) to be protected by WebSEAL will be created. The name of the directory can be any valid directory name.

The directory used by the default (first) WebSEAL instance is:

UNIX:
installation_directory/pdweb/www-default/docs
Windows:
installation_directory\pdweb\www-default\docs

Note that this directory could have been changed by the administrator during the configuration of the initial WebSEAL server instance.

When adding a new WebSEAL server instance, a new Web document root directory is usually created for the instance.

During an interactive installation, a new directory is suggested, based on the following syntax:

UNIX:
installation_directory/pdweb/www-instance_name/docs
Windows:
installation_directory\pdweb\www-instance_name\docs

The administrator can accept this name or specify an alternative.

When adding a server instance by using the `amwebcfg` command line, or by using `amwebcfg` with a response file, the Web document root directory is created as follows:
When the Web document root is not specified on the command line or in the response file, `amwebcfg` automatically creates a new directory and adds the entry to the WebSEAL instance configuration file. The document root is built according to the following syntax:

**UNIX:**
```
installation_directory/pdweb/www-instance_name/docs
```

**Windows:**
```
installation_directory\pdweb\www-instance_name\docs
```

When the Web document root is specified on the command line or in the response file, `amwebcfg` adds the entry to the WebSEAL instance configuration file.

**Note:** The directory must already exist. The `amwebcfg` utility will not create a new directory.

### Sharing one Web document root directory across multiple instances

Multiple WebSEAL server instances can use the same Web document root directory. When you want to use this scenario, the best way to configure the document root for each new server instance is as follows:

1. Allow `amwebcfg` to create a new Web document root directory.
2. When `amwebcfg` configuration completes, manually edit the WebSEAL configuration file and reassign the document root value to the preferred directory.

```
[content]
doc-root = full_path_to_directory
```

This procedure is recommended because each time a Web document root hierarchy is created, `amwebcfg` copies the contents of the html.tivoli directory hierarchy into the new Web document root. The contents of html.tivoli include an index.html file. This means that an existing index.html could get overwritten by the default (template) file from the html.tivoli directory. Manual editing of the WebSEAL configuration file avoids this problem. After editing the configuration file, you can remove the unneeded Web document root (the one created automatically by `amwebcfg`).

### Example server instance configuration values

The following table contains a set of example settings for a WebSEAL server instance. These example settings will be used in the sample configuration commands used in the remainder of the configuration sections of this chapter.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrative user ID</td>
<td>sec_master</td>
</tr>
<tr>
<td>Administrative user password</td>
<td>mypassw0rd</td>
</tr>
<tr>
<td>Host name</td>
<td>diamond.subnet2.ibm.com</td>
</tr>
<tr>
<td>Instance name</td>
<td>web1</td>
</tr>
<tr>
<td>Listening port</td>
<td>7235</td>
</tr>
<tr>
<td>Enable HTTP usage</td>
<td>yes</td>
</tr>
<tr>
<td>HTTP port</td>
<td>81</td>
</tr>
<tr>
<td>Enable HTTPS usage</td>
<td>yes</td>
</tr>
<tr>
<td>HTTPS port</td>
<td>444</td>
</tr>
<tr>
<td>Use logical network interface?</td>
<td>yes</td>
</tr>
<tr>
<td>IP address</td>
<td>1.2.3.5</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Use SSL to communicate with LDAP server?</td>
<td>yes</td>
</tr>
<tr>
<td>SSL key file</td>
<td>/tmp/client.kdb</td>
</tr>
<tr>
<td>SSL key file password</td>
<td>keyfilepassw0rd</td>
</tr>
<tr>
<td>SSL certificate label</td>
<td>(none)</td>
</tr>
<tr>
<td>SSL port</td>
<td>636</td>
</tr>
<tr>
<td>Web document root directory</td>
<td>/usr/docs</td>
</tr>
</tbody>
</table>

**Unique configuration file for each instance**

A unique WebSEAL configuration file is created for each WebSEAL server instance. The name of the configuration file includes the instance name. The format is:

```
/opt/pdweb/etc/webseal-instance_name.conf
```

The newly created instance-specific configuration file is automatically configured for SSL communication between the new WebSEAL instance and internal Tivoli Access Manager servers such as the policy server.

The new file is also automatically configured to use the server certificate of the initial WebSEAL server to authenticate to client browsers.

**Interactive configuration overview**

Interactive configuration of WebSEAL is accessed through the `pdconfig` utility. This utility provides a graphical user interface that prompts the administrator to enter the information needed to configure a WebSEAL server instance. The configuration utility provides online help messages to assist you to determine the appropriate values for each requested setting. When all configuration information has been entered, the utility completes the configuration and starts the new WebSEAL server instance.

The `pdconfig` utility can be used to configure many different Tivoli Access Manager components. The `pdconfig` menu includes an entry for WebSEAL. When the WebSEAL entry is selected, the information requested by `pdconfig` matches the information needed by `amwebcfg`. This means that the planning steps described earlier in this section are equally applicable to `pdconfig`.

On all systems, you can `pdconfig` from a shell prompt. Example:

```
# pdconfig
```

On Windows systems, you can also access the configuration utility through the Windows desktop menus:

Start -> Programs -> IBM Tivoli Access Manager -> Configuration

**Command line configuration overview**

You can use `amwebcfg` to configure a WebSEAL server instance from a command line. All necessary settings are supplied as command line options. The utility completes the configuration without further prompting of the administrator.

The `amwebcfg` syntax is:

```
amwebcfg -action config -host host_name -listening_port am_listener_port
-admin_id admin_id -admin_pwd admin_pwd -inst_name instance_name
-nw_interface_yn network_interface -ip_address ip_address
```

Chapter 2. WebSEAL server configuration 21
The above command must be entered as one continuous command line.

The options to **amwebcfg** are as shown in the following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-admin_id</td>
<td>Administrative user ID</td>
</tr>
<tr>
<td>-admin_pwd</td>
<td>Administrative user password</td>
</tr>
<tr>
<td>-host</td>
<td>Host name</td>
</tr>
<tr>
<td>-inst_name</td>
<td>Instance name</td>
</tr>
<tr>
<td>-listening_port</td>
<td>Listening port</td>
</tr>
<tr>
<td>-http_yn</td>
<td>Enable HTTP usage</td>
</tr>
<tr>
<td>-http_port</td>
<td>HTTP port</td>
</tr>
<tr>
<td>-https_yn</td>
<td>Enable HTTPS usage</td>
</tr>
<tr>
<td>-https_port</td>
<td>HTTPS port</td>
</tr>
<tr>
<td>-nw_interface_yn</td>
<td>Use logical network interface</td>
</tr>
<tr>
<td>-ip_address</td>
<td>IP address</td>
</tr>
<tr>
<td>-ssl_yn</td>
<td>Use SSL to communicate with LDAP server</td>
</tr>
<tr>
<td>-key_file</td>
<td>SSL key file</td>
</tr>
<tr>
<td>-key_file_pwd</td>
<td>SSL key file password</td>
</tr>
<tr>
<td>-cert_label</td>
<td>SSL certificate label</td>
</tr>
<tr>
<td>-ssl_port</td>
<td>SSL port</td>
</tr>
<tr>
<td>-doc_root</td>
<td>Web document root directory</td>
</tr>
</tbody>
</table>

For example, using the example settings listed in “Example server instance configuration values” on page 20, the command line would be:

```
amwebcfg -action config -inst_name default -host diamond.subnet2.ibm.com -listening_port 7234 -admin_id sec_master -admin_pwd mypass0rd -inst_name web1 -nw_interface_yn yes -ip_address 1.2.3.5 -ssl_yn yes -key_file /tmp/client.kdb -key_file_pwd mypass0rd -cert_label ibm_cert -ssl_port 636 -http_yn yes -http_port 81 -https_yn yes -https_port 444 -doc_root /usr/docs
```

The above command must be entered as one continuous command line.

The **amwebcfg** utility prompts the user to supply any values that are not supplied through command line options. The exceptions to this rule are as follows:

- SSL certificate label
  - When this value is not supplied, no value is set. This means that the default certificate is used.
- Web document root directory
  - A unique directory is created for the instance. The algorithm for creating the directory is described in “Planning a server instance configuration” on page 16.

For more information see the **amwebcfg** reference page in the *IBM Tivoli Access Manager for e-business Command Reference.*
Silent configuration overview
You can use `amwebcfg` to configure a WebSEAL server instance by reading all necessary values from a text file. The text file is called a response file. When `amwebcfg` obtains settings from the response file, it completes the configuration without further prompting of the administrator.

The response file is not supplied by default. You must use a text editor to create it and enter the necessary values. The values consist of a series of `key = value` pairs. Each parameter entry is based on an option to `amwebcfg`. Each `key = value` pair is placed on a separate line. To insert a comment line, place a hash character (`#`) at the start of the line. The format of the response file is identical to the format of the WebSEAL configuration file.

This installation option is useful when you have to create multiple WebSEAL instances. After you have created and used one response file, you can use it as a template for future response files. Copy the existing file to a new location, and edit it with values appropriate to the server instance. There are no restrictions on the location of the response file.

Example command line
`amwebcfg -rspfile /tmp/response_file_name`

The following table shows an example response file for the server instance values shown in “Example server instance configuration values” on page 20.

```
Example response file
[webseal-config]
action = config
host = diamond.subnet2.ibm.com
listener_port = 7234
admin_id = sec_master
admin_pwd = mypassword
inst_name = web1
nw_interface_yn = yes
# If nw_interface_yn = no, do not need to specify the value for ip_address
ip_address = 1.2.3.5
# If SSL is not enabled, do not need to specify the values for ssl_yn, key_file, key_file_pwd, cert_label, and ssl_port
ssl_yn = yes
key_file = /tmp/client.kdb
key_file_pwd = keyfilepassword
# /cert_label is optional.
# If you have no cert label, remove entry from response file
cert_label = ibm_cert
ssl_port = 636
http_yn = yes
http_port = 81
https_yn = yes
https_port = 444
# If the doc-root is not provided, amwebcfg creates the default one.
# The default is /install_dir/pdweb/www-instance/docs
# If you do not provide a value for doc-root, remove the entry from the
# response file or you will be prompted to enter it
doc_root = /usr/www-web1/docs
```
Server instance configuration tasks

Tasks:

- “Adding a WebSEAL server instance”
- “Removing a server instance” on page 26

Adding a WebSEAL server instance

Complete each of the following steps:

1. Plan the configuration. Complete the following worksheet. For information on determining appropriate values for each setting, see “Planning a server instance configuration” on page 16.

Table 3. Worksheet for adding a server instance

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrative user ID</td>
<td></td>
</tr>
<tr>
<td>Administrative user password</td>
<td></td>
</tr>
<tr>
<td>Host name</td>
<td></td>
</tr>
<tr>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td>Listening port</td>
<td></td>
</tr>
<tr>
<td>Enable HTTP usage</td>
<td>yes or no</td>
</tr>
<tr>
<td>HTTP port</td>
<td></td>
</tr>
<tr>
<td>Enable HTTPS usage</td>
<td>yes or no</td>
</tr>
<tr>
<td>HTTPS port</td>
<td></td>
</tr>
<tr>
<td>Use logical network interface?</td>
<td>yes or no</td>
</tr>
<tr>
<td>IP address</td>
<td></td>
</tr>
<tr>
<td>Use SSL to communicate with LDAP server?</td>
<td>yes or no</td>
</tr>
<tr>
<td>SSL certificate label</td>
<td></td>
</tr>
<tr>
<td>SSL key file</td>
<td></td>
</tr>
<tr>
<td>SSL key file password</td>
<td></td>
</tr>
<tr>
<td>SSL port</td>
<td></td>
</tr>
<tr>
<td>Web document root directory</td>
<td></td>
</tr>
</tbody>
</table>

2. Ensure that all configured WebSEAL server instances are running. This prevents any possible conflicts between servers over port usage. On UNIX, use the pdconfig utility to view server status. On Windows, use the Services Control Panel.

3. Determine if you need to manually edit the configuration file for the first (default) WebSEAL server instance.

   This step is necessary only when the following conditions are true:
   - You want to install a logical network interface
   - You want to use the same HTTP or HTTPS ports as used by the first server instance
   - The first server instance was installed without using a logical network interface

When the above conditions are true, you must manually edit the WebSEAL configuration file to assign an IP address to the first instance. Add a network-interface entry in the [server] stanza. For example:
network-interface = 1.2.3.4

For more information see “Planning a server instance configuration” on page 16.

4. Configure the server instance. Use one of the following configuration methods:
   • Interactive configuration
     a. Start the pdconfig utility from a command line.
        On Windows, you can alternatively use the Windows menu:
        Start -> Programs -> IBM Tivoli Access Manager -> Configuration
     b. Follow the screen prompts. When prompted, supply each value from the worksheet.
        When the configuration completes, the WebSEAL server instance is automatically started.
   • Command line configuration
     Start amwebcfg with the necessary command line options. Construct your command line options by entering the values from your worksheet as arguments to the appropriate option.
     The amwebcfg syntax is:
     amwebcfg -action config -host host_name -listening_port am_listener_port
              -admin_id admin_id -admin_pwd admin_pwd -inst_name instance_name
              -nw_interface_yn network_interface -ip_address ip_address
              -ssl_yn ssl_enable_y_n -key_file key_file
              -key_file_pwd key_file_pwd -cert_label cert_label
              -ssl_port ssl_port -http_yn allow_http_yn -http_port http_port
              -https_yn allow_https_yn -https_port https_port -doc_root doc_root

     The above command is entered as one continuous command line. For more information, see “Command line configuration overview” on page 21.
   • Silent configuration from a response file.
     a. Create a response file containing the values from your worksheet.
     b. Start amwebcfg:
        amwebcfg -rspfile /tmp/response_file_name

5. Verify that the new instance is running.
   For interactive installation, review the WebSEAL server entry in the pdconfig status window.
   For command line configuration and silent configuration, amwebcfg sends a status message indicating that the configuration succeeded. When this message displays, the WebSEAL server is running.
Removing a server instance
To remove the configuration for a WebSEAL server instance, complete the following steps:

1. Assemble the following information:
   - Instance name
   - Administrator ID
   - Administrator ID password

2. Remove the configuration using one of the following methods:
   - Interactive configuration
     a. Start the configuration utility:
        Start the `pdconfig` utility from a command line.
        On Windows, you can alternatively use the Windows menu:
        Start -> Programs -> IBM Tivoli Access Manager -> Configuration
     b. Select the server instance, and choose to unconfigure.
     c. When prompted, supply administrator ID and password.
        The removal completes without further user input.
     d. Use the pdconfig status window to verify that the WebSEAL server instance is no longer configured.
   - Command line configuration
     a. Start the configuration utility, supplying the required command line options. The syntax is:
        ```
        amwebcfg -action unconfig -inst_name instance_name -admin_id admin_id -admin_pwd admin_pwd
        ```
        The configuration utility displays a status message when the removal completes.
   - Silent configuration using a response file
     Complete the following steps:
     a. Create a response file containing the following information:
        - Action to take (unconfiguration)
        - Instance name
        - Administrator ID
        - Administrator ID password
     For example, when removing a WebSEAL server instance called `web1`, the entries would be:

     ```
     [webseal-config]
     action = unconfig
     inst_name = web1
     admin_id = sec_master
     admin_pwd = mypassw0rd
     ```

     Substitute your administrator password for the `admin_pwd` parameter in the example above.

     Save the file. For example:
     ```
     /tmp/response_web1
     ```
b. Start the **amwebcfg** utility. The syntax is:

```
amwebcfg -rspfile /path_to_response_file/response_file_name
```

Using the example above, the command line is:

```
amwebcfg -rspfile /tmp/response_web1
```

The configuration utility displays a status message when the removal completes.
Communication protocol configuration

The following sections describe general information about the WebSEAL server:

- "Configuring WebSEAL for HTTP requests" on page 28
- "Configuring WebSEAL for HTTPS requests" on page 28
- "Restricting connections from specific SSL versions" on page 30
- "Timeout parameters for HTTP/HTTPS communication" on page 30
- "Additional WebSEAL server timeout parameters" on page 30

Configuring WebSEAL for HTTP requests

WebSEAL typically handles many HTTP requests from unauthenticated users. For example, it is common to allow anonymous users read-only access to selected documents on the public section of your Web site.

Parameters for handling HTTP requests over TCP are located in the [server] stanza of the WebSEAL configuration file.

**Enabling/disabling HTTP access**

Enable or disable HTTP access during WebSEAL configuration:

```
http = {yes|no}
```

IBM HTTP Server, WebSphere Application Server (which installs IBM HTTP Server), and WebSEAL all use port 80 as the default port. If you install WebSEAL on the same system as IBM HTTP Server, ensure that you change the default port to one of these servers. Edit the `httpd.conf` configuration file or the WebSEAL configuration file.

**Setting the HTTP access port value**

The default port for HTTP access is 80:

```
http-port = 80
```

To change to port 8080, for example, set:

```
http-port = 8080
```

Configuring WebSEAL for HTTPS requests

Parameters for handling HTTP requests over SSL (HTTPS) are located in the [server] stanza of the WebSEAL configuration file.

**SSL connections using the WebSEAL test certificate (this belongs in an SSL discussion)**

Client-side certificate authentication must take place over a Secure Socket Layer (SSL) connection. The SSL connection is established prior to the certificate authentication process. The SSL connection can be established when a client attempts to access a resource over HTTPS. When the resource does not require authenticated access, the client negotiates an SSL session with the WebSEAL server. The SSL session is established when the client and server (WebSEAL) examine each other's certificate and accept the validity of the signing authority.

In order to enable the establishment of SSL sessions on a new WebSEAL server, WebSEAL contains a self-signed test server certificate. WebSEAL can present the self-signed certificate to the client. If the client accepts the certificate, the SSL session is established.
This test certificate is not suitable for permanent use by the WebSEAL server. Although this test certificate allows WebSEAL to respond to an SSL-enabled browser request, it cannot be verified by the browser. This is because the browser does not contain an appropriate root CA certificate — as is the case for when the browser receives any self-signed certificate for which a root CA certificate does not exist. Because the private key for this default certificate is contained in every WebSEAL distribution, this certificate offers no true secure communication.

To ensure secure communication over SSL, WebSEAL administrators must obtain a unique site server certificate from a trusted Certificate Authority (CA). You can use the GSKit iKeyman utility to generate a certificate request that is sent to the CA. You can also use iKeyman to install and label the new site certificate.

Use the `webseal-cert-keyfile-label` parameter in the `[ssl]` stanza of the WebSEAL configuration file to designate the certificate as the active WebSEAL server-side certificate (this setting overrides any certificate designated as “default” in the keyfile database).

If you require different certificates for other scenarios (such as for mutually authenticated junctions), you can use the iKeyman utility to create, install, and label these additional certificates. See “Configuring WebSEAL key database parameters” on page 229.

It is also important to ensure that validation of certificates includes checking of Certificate Revocation Lists (CRLs). Configure WebSEAL to access the appropriate LDAP server as an LDAP user with sufficient permission to access the appropriate CRLs. Supply values for the following configuration file entries:

```
[ssl]
crl-ldap-server
crl-ldap-server-port
crl-ldap-user
crl-ldap-user-password
```

WebSEAL can be configured to cache CRLs. To configure the cache, supply values for the following configuration file entries:

```
[ssl]
gsk-crl-cache-size
gsk-crl-cache-entry-lifetime
```

Instructions for setting values that affect CRL access and handling, including valid ranges for cache settings, are specified in the “Secure Socket Layer” on page 406 section. See also “Configuring the CRL cache” on page 231.

**Enabling/disabling HTTPS access**
Enable or disable HTTPS access during WebSEAL configuration:

```
https = {yes|no}
```

**Setting the HTTPS access port value**
The default port for HTTPS access is 443:

```
https-port = 443
```

To change to port 4343, for example, set:

```
https-port = 4343
```
Restricting connections from specific SSL versions

You can independently enable and disable connectivity for SSL (Secure Sockets Layer) version 2, SSL version 3, and TLS (Transport Layer Security) version 1. The parameters that control connections for specific SSL and TLS versions are located in the [ssl] stanza of the WebSEAL configuration file. By default, all SSL and TLS versions are enabled.

```
[ssl]
  disable-ssl-v2 = {yes|no}
  disable-ssl-v3 = {yes|no}
  disable-tls-v1 = {yes|no}
```

Timeout parameters for HTTP/HTTPS communication

WebSEAL uses the IBM Global Security Kit (GSKit) implementation of SSL. When WebSEAL receives a request from an HTTPS client, GSKit SSL establishes the initial handshake and maintains session state.

WebSEAL supports the following timeout parameters for HTTP and HTTPS communication. These parameters are located in the [server] stanza of the WebSEAL configuration file.

- **client-connect-timeout**
  After the initial handshake has occurred, this parameter dictates how long WebSEAL holds the connection open for the initial HTTP or HTTPS request. The default is 120 seconds.

```
[server]
  client-connect-timeout = 120
```

- **persistent-con-timeout**
  After the first HTTP request and server response, this parameter controls maximum number of seconds WebSEAL holds an HTTP persistent connection open before it is shutdown. The default value is 5 seconds.

```
[server]
  persistent-con-timeout = 5
```

Figure 4. Timeout parameters for HTTP and HTTPS communication

Additional WebSEAL server timeout parameters

The following additional timeout parameters are set in the WebSEAL configuration file:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[junction] http-timeout</td>
<td>The timeout value for sending to and reading from a back-end server over a TCP junction.</td>
<td>120</td>
</tr>
<tr>
<td>[junction] https-timeout</td>
<td>The timeout value for sending to and reading from a back-end server over an SSL junction.</td>
<td>120</td>
</tr>
<tr>
<td>[cgi] cgi-timeout</td>
<td>The timeout value for sending to and reading from a local CGI process. Supported on UNIX systems only.</td>
<td>120</td>
</tr>
<tr>
<td>[junction] ping-time</td>
<td>WebSEAL performs a periodic background ping of each junctioned server to determine whether it is running. WebSEAL will not try more often than once every 300 seconds (or whatever value is set).</td>
<td>300</td>
</tr>
</tbody>
</table>
Cryptographic hardware for encryption and key storage

WebSEAL, using GSKit for SSL communication and key management, provides interface support for cryptographic hardware. Cryptographic hardware can provide one or both of the following features:

- Accelerated and secure SSL encryption and decryption tasks for performance improvements during multiple online transactions
- Accelerated and secure digital certificate key storage and management for highly secure architecture during online transactions

WebSEAL and GSKit support the following interfaces to this cryptographic hardware:

- BHAPI (RSA Security, Inc.’s API supporting its BSAFE product)
- PKCS#11 (Public Key Cryptographic Standard)

Some cryptographic hardware supports both interfaces; some cryptographic hardware supports only one of the interfaces.

In general, WebSEAL (and GSKit) uses the BHAPI interface to support encryption and decryption. WebSEAL (and GSKit) uses PKCS#11 to support both encryption/decryption and key storage.

WebSEAL supports several hardware devices for selected platforms. Consult the IBM Tivoli Access Manager for e-business Release Notes to obtain the latest information on platform support for these hardware cards.

Cryptographic accelerator cards

- nCipher nForce 300
- Rainbow CryptoSwift eCommerce Accelerator
- IBM 4960

Key Storage

- nCipher nForce 300
- IBM 4758
- Eracom Orange

The following matrix illustrates the relationship between functionality and interface support for each of these cards:

<table>
<thead>
<tr>
<th></th>
<th>BHAPI</th>
<th>PKCS#11</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Acceleration</td>
<td>• Rainbow CryptoSwift</td>
<td>• IBM 4960</td>
</tr>
<tr>
<td></td>
<td>• nCipher nForce 300</td>
<td>• nCipher nForce 300</td>
</tr>
<tr>
<td>Key Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• IBM 4758</td>
<td>• nCipher nForce 300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Eracom Orange</td>
</tr>
</tbody>
</table>

The Rainbow CryptoSwift and nCipher nForce 300 (using BHAPI) are used for public key operations (RSA key decryption). Keys are not stored on the accelerator device, but are stored in the pdsrv.kdb file. Accelerator devices are used to speed up the public key cryptographic functions of SSL. Hardware acceleration frees up the server processor, increases server throughput, and shortens wait time. The
Rainbow CryptoSwift and nCipher nForce accelerators incorporate faster performance by providing more concurrent secure transactions.

With the PKCS#11 interface, RSA keys are stored on a cryptocard to ensure authentication. The IBM 4758 and Eracom Orange perform only as key storage devices. The nCipher nForce device can either perform just acceleration or it can perform both acceleration and key storage with PKCS#11 support. The IBM 4758, Eracom Orange, and nCipher nForce devices (with PKCS#11 support) ensure that keys are completely inaccessible to the outside world. Keys are never revealed in an unencrypted form because they are stored on the hardware, providing enhanced key protection and authentication.

Hardware cryptographic acceleration and key storage apply to the following WebSEAL connections:
- Browser to WebSEAL
- WebSEAL to back-end junctioned server

**Conditions and prerequisites**

**IBM 4758–023**

On Windows 2000, the IBM 4758–023 cryptographic card has an access limitation of 32 worker threads. Therefore, WebSEAL must be configured to use no more than 32 worker threads. The default setting for worker threads at WebSEAL installation time is 50 worker threads. Refer to [“Managing worker thread allocation” on page 41](#) for information about worker thread configuration.

Additional monitoring threads are used for any junctions configured with the –K option (WebSEAL authenticates with a client-side certificate) where the key (certificate) is stored on the IBM 4758 hardware. In this situation, further reduce the number of worker threads by one for each SSL –K junction using keys stored on the IBM 4758 card.

**Configuring Cipher engine and FIPS mode processing**

You can use the WebSEAL configuration file to specify the Cipher engine used by GSKit.

```
[ssl]
base-crypto-library = Default
```

Valid values for this entry are:
- Default
  This value tells GSKit to select the optimal cryptographic base to use. For WebSEAL Version 5.1, this is ICC
- ICC
- RSA

Specify RSA when using a BHAPI (Bsafe Hardware API) CryptoCard such as Rainbow CryptoSwift. The BHAPI interface is not supported by ICC. When the WebSEAL deployment environment includes previous version (before Version 5.1) WebSEAL servers, consider using this setting because RSA was used by previous versions of GSKit for cryptographic operations.

**Note:** PKCS#11 is available in all modes, except when FIPS is enabled.
You can specify whether to enable FIPS mode processing. FIPS mode processing is disabled by default. To enable it, set the following entry:

```ini
[ssl]
fips-mode-processing = yes
```

Set this to yes when using ICC and wanting to use the FIPS 140-1 approved protocols and ciphers.

### Configuring WebSEAL for cryptographic hardware over BHAPI

1. Install the device driver for the specific cryptographic hardware you are using.
2. GSKit (and therefore WebSEAL) detects the hardware and automatically uses it.

If required, you can configure WebSEAL to disable the automatic use of the hardware for SSL acceleration over BHAPI. The `disable-ncipher-bsafe` and `disable-rainbow-bsafe` parameters are available in the `[ssl]` stanza of the WebSEAL configuration file. By default, both parameters are set to "no" (that is, WebSEAL automatically uses the hardware for SSL acceleration over BHAPI). For example:

```ini
[ssl]
disable-ncipher-bsafe = no
disable-rainbow-bsafe = no
```

### Configuring WebSEAL for cryptographic hardware over PKCS#11

#### Install the cryptographic card and device driver
Follow the instructions provided by the specific vendor to install the cryptographic card and its device driver (with PKCS#11) for the specific cryptographic hardware you are using. This procedure involves shutting down and restarting the computer machine.

#### Create a token device label and password to store WebSEAL keys
In the context of cryptographic hardware and the associated device drivers, a *token* is a logical device that acts as a "container" for storing key, data, and certificate objects. Key objects can include public keys and private keys. When you configure a cryptographic card to perform key storage (using the PKCS#11 interface), you must define one or more tokens (or "containers") that store keys for different situations.

When you configure a cryptographic card to perform key storage tasks for WebSEAL (GSKit), you must specify a token label (and password) that represents the token device that stores the WebSEAL public/private key pair. WebSEAL sends the public key in the server-side certificate that it uses to authenticate itself to any client.

Use the instructions provided with the installed cryptographic hardware to create a label for the token device that stores the WebSEAL key.

For example:

```bash
token = websealtoken
password = secret
```

#### Configure iKeyman to use the PKCS#11 module (shared library)
The GSKit iKeyman utility needs to be configured for the PKCS#11 device module (shared library) of the installed cryptographic hardware device. This module
allows iKeyman to understand the hardware device’s WebSEAL token label, the password (or PIN) for the token, and the key label of any WebSEAL key stored on the device.

Configure the GSKit iKeyman utility (gsk7ikm) to use the PKCS#11 module (shared library) for the installed cryptographic hardware device:

Locate the file ikmuser.sample in the following directory:

**Solaris and HPUX:** /opt/ibm/gsk7/classes

**Linux:** /usr/local/ibm/gsk7/classes

**AIX:** /usr/opt/ibm/gskta/classes

**Windows:** C:\Program Files\ibm\gsk7\classes

Copy and rename this file to ikmuser.properties.

Edit this new file by adding the following appropriate line that specifies the location of the module (shared library) for the installed cryptographic hardware:

**UNIX:**

ncipher nForce:
DEFAULT_CRYPTOGRAPHIC_MODULE=/opt/nfast/toolkits/pkcs11/libcknfast.so

IBM 4758-023 and IBM 4960:
DEFAULT_CRYPTOGRAPHIC_MODULE=/usr/lib/pkcs11/PKCS11_API.so

Eracom Orange:
DEFAULT_CRYPTOGRAPHIC_MODULE=/opt/Eracom/lib/libcryptoki.so

**Windows:**

ncipher nForce:
DEFAULT_CRYPTOGRAPHIC_MODULE=C:\nfast\toolkits\pkcs11\cknfast.dll

IBM 4758-023:
DEFAULT_CRYPTOGRAPHIC_MODULE=C:\Program Files\ibm\PKCS11\bin\nt\cryptoki.dll

Eracom Orange
DEFAULT_CRYPTOGRAPHIC_MODULE=
C:\Program Files\Eracom\ProtectToolKit C Runtime\cryptoki.dll

When the appropriate shared library is configured, the GSKit iKeyman utility includes a new menu option: "Cryptographic Token". Now you can use iKeyman to create, store, and manipulate keys for WebSEAL on the cryptographic hardware.

**Open the WebSEAL token device using iKeyman**

1. Start the iKeyman utility (gsk7ikm).
2. Select Key Database File, then Open.
   A separate Open dialog box appears.
3. In the Open dialog window, select Cryptographic Tokens from the Key database type pull-down menu.

4. If you have the cryptographic token specified in the ikmuser.properties file, you will see both the path and the library in the dialog box. If you do not see them, you can use the Browse... menu option. Click OK when this is done.

5. Additionally, if you want to open an existing secondary key database (for key data not stored on the cryptographic hardware—such as CA root certificates), check “Open Existing Key Database”.

6. Browse for and select the default WebSEAL key database:
   UNIX:
   /opt/pdweb/www/certs/pdsrv.kdb
   Windows:
   \C:\Program Files\Tivoli\pdweb\www\certs\pdsrv.kdb

7. Click OK.

   The Token Password dialogue box appears.

8. Enter the default password “pdsrv”. Click OK.

9. The main iKeyman window returns.

**Request and store the WebSEAL server certificate**


2. Follow instructions in the IBM Global Security Kit Secure Sockets Layer and iKeyman User’s Guide to receive the WebSEAL certificate from the CA and store it in a key database. When performing this procedure, select the token device representing the cryptographic hardware as the storage location for the certificate.

3. When it is stored on the token device, the key (certificate) appears (for example) as:
   websealtoken:webseal

   The WebSEAL key is stored on the cryptographic hardware and assigned to the token device labeled “websealtoken”.

**Configure WebSEAL and GSKit to use the PKCS#11 shared library**

Configure WebSEAL to use the PKCS#11 module (shared library).

Edit the WebSEAL configuration file and add the appropriate line identifying the location of the shared library under the [ssl] stanza:

**UNIX**

nCipher nForce:

[ssl]
pkcs11-driver-path = /opt/nfast/toolkits/pkcs11/libcknfast.so

IBM 4758-023 and IBM 4960:

[ssl]
pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so

Eracom Orange
Windows

nCipher nForce:

[ssl]
pkcs11-driver-path = C:\nfast\toolkits\pkcs11\cknfast.dll

IBM 4758-023:

[ssl]
pkcs11-driver-path = C:\Program Files\ibm\PKCS11\bin\nt\cryptoki.dll

Eracom Orange

[ssl]
pkcs11-driver-path = C:\Program Files\Eracom\ProtectedToolKit C Runtime\cryptoki.dll

In addition, specify the names of the token label and password under the same [ssl] stanza:

For this example:

[ssl]
pkcs11-token-label = websealtoken
pkcs11-token-pwd = secret

Modify the WebSEAL server certificate label

Configure WebSEAL to use this new hardware-based key rather than the default key in its communications with browser clients. Modify the webseal-cert-keyfile-label parameter in the [ssl] stanza of the WebSEAL configuration file to designate the new key label.

[ssl]
webseal-cert-keyfile-label = <token-name>:<key-label>

For this example:

[ssl]
webseal-cert-keyfile-label = websealtoken:webseal

Disable acceleration mode for nCipher nForce 300

If you want to use the nCipher nForce 300 device only for key storage, and not SSL acceleration, you can configure WebSEAL to disable the automatic use of this device for BHAPI acceleration. The disable-ncipher-bsafe parameter is available in the [ssl] stanza of the WebSEAL configuration file.

To disable automatic SSL acceleration over the BHAPI interface, set this parameter to "yes". For example:

[ssl]
disable-ncipher-bsafe = yes

By default, this parameter is set to "no" (that is, WebSEAL automatically uses the hardware for SSL acceleration over the BHAPI interface).

Restart WebSEAL

You must restart WebSEAL for all cryptographic hardware configuration to take effect.

You can verify that WebSEAL is using the cryptographic hardware by examining entries contained in the msg_webseald.log file.
Quality of protection levels

You can control the default level of encryption required for access to WebSEAL over SSL (HTTPS) by configuring the quality of protection (QOP). Default quality of protection management is controlled using parameters in the "SSL QUALITY OF PROTECTION MANAGEMENT" section of the WebSEAL configuration file:

- Enable and disable QOP management with the ssl-qop-mgmt parameter.
- Specify allowed encryption levels in the [ssl-qop-mgmt-default] stanza.

1. Enable quality of protection management:

   [ssl-qop]
   ssl-qop-mgmt = yes

2. Specify the default encryption level for HTTPS access. The syntax is:

   default = {ALL|NONE|cipher_level}

Supported values for cipher_level are:

- NONE, ALL, NULL, DES-56, FIPS-DES-56, DES-168, FIPS-DES-168,
  RC2-40, RC2-128, RC4-40, RC4-56, RC4-128, AES-128, AES-256

NONE disable

For example:

   [ssl-qop-mgmt-default]
   default = ALL

Note that you can also specify a selected group of ciphers:

   [ssl-qop-mgmt-default]
   default = RC4-128
   default = RC2-128
   default = DES-168

Notes:
- NONE means that no SSL connection allowed.
- NULL means that unencrypted SSL connection allowed.
- ALL means that all types of SSL connections allowed.
- There can be multiple cipher/MAC made available to the connection for a given qop cipher selection. These will still have the same encryption bit strength, just different MAC methods (SHA1 or MD5)
- RC2-128 is only available with SSLv2. If it is the only cipher selection, WebSEAL will disable SSLv3 and TLSv1 for the affected connection.
- NULL, FIPS-DES-56, FIPS-DES-168, RC4-56, AES-128, and AES-256 are only available with SSLv3 and TLSv1. If they are the only ciphers available to a given connection, SSLv2 will be disabled for the affected connection.
- AES Support is determined automatically by GSKit based on the base-crypto-library setting. AES-128 and AES-256 are only available if AES Support is enabled by GSKit, else they will be ignored.
- FIPS-DES-56 and FIPS-DES-168 are only available when fips-mode-processing is enabled (set to yes). Otherwise they are ignored.

Tivoli Access Manager uses GSKit 7. The Cipher specifications supported by GSKIT7 when used in SSLv2/TLS in internet security are:

- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA

These TLS cipher specifications are also used with SSLV3.

**QOP for individual hosts and networks**

The `ssl-qop-mgmt = yes` parameter also enables any settings that appear in the `ssl-qop-mgmt-hosts` and `ssl-qop-mgmt-networks` stanzas. These stanzas allow quality of protection management by specific host/network/netmask IP address.

The `ssl-qop-mgmt-default` stanza lists the ciphers used for all IP addresses not matched in the `ssl-qop-mgmt-hosts` and `ssl-qop-mgmt-networks` stanzas.

Example configuration syntax for hosts:
```
[ssl-qop-mgmt-hosts]
xxx.xxx.xxx.xxx = ALL
yyy.yyy.yyy.yyy = RC2-128
```

Example configuration syntax for network/netmask:
```
[ssl-qop-mgmt-networks]
xxx.xxx.xxx.xxx/255.255.255.0 = RC4-128
yyy.yyy.yyy.yyy/255.255.0.0 = DES-56
```

The `ssl-qop-mgmt-hosts` and `ssl-qop-mgmt-networks` stanzas are provided for backward compatibility only. It is recommended that you not use them for Tivoli Access Manager configuration.

Note that entry for an IP address specified under `ssl-qop-mgmt-hosts` takes priority over an entry for the same address in `ssl-qop-mgmt-networks`. Likewise, an entry in `ssl-qop-mgmt-networks` takes priority over an entry for the same address in `ssl-qop-mgmt-default`. If you must use `ssl-qop-mgmt-hosts` or `ssl-qop-mgmt-networks` for backwards compatibility, review the IP address settings under all stanzas to ensure that a specific IP address is not listed under more than one stanza. If an IP address is listed under more than one stanza, ensure that the order of evaluation yields the desired configuration.
Configuring authorization database updates and polling

The Tivoli Access Manager policy server (pdmgrd) manages the master authorization policy database and maintains location information about other Tivoli Access Manager servers in the secure domain. A Tivoli Access Manager administrator can make security policy changes to the secure domain at any time. The policy server makes the necessary adjustments to the master authorization database whenever security policy changes are implemented.

When the policy server makes a change to the master authorization database, it can send out notification of this change to all replica databases in the secure domain that support individual policy enforcers (such as WebSEAL). The policy enforcers must then request an actual database update from the master authorization database.

WebSEAL, as a resource manager and policy enforcer, has three options to obtain information about authorization database changes:

- Listen for update notifications from the policy server (configurable and enabled by default).
- Check (poll) the master authorization database at regular intervals (configurable and disabled by default).
- Enable both listening and polling.

The [aznapi-configuration] stanza of the WebSEAL configuration file contains parameters for configuring update notification listening and database polling.

The path to WebSEAL’s local replica authorization policy database is defined by the db-file parameter:

```
[aznapi-configuration]
db-file = /var/pdweb/db/webseald.db
```

Configuring update notification listening

The listen-flags parameter, found in the [aznapi-configuration] stanza, enables and disables update notification listening by WebSEAL. By default, listening is disabled. To disable listening, enter "enable".

```
[aznapi-configuration]
listen-flags = enable
```

The ssl-listening-port parameter, found in the [ssl] stanza, configures the SSL port for the listener:

```
[ssl]
ssl-listening-port = 7234
```

Configuring authorization database polling

You can configure WebSEAL to regularly poll the master authorization database for update information. The cache-refresh-interval parameter can be set to "default", "disable", or a specific time interval in seconds. The "default" setting is equal to 600 seconds. By default, polling is disabled.

```
[aznapi-configuration]
.cache-refresh-interval = disable
```
Managing worker thread allocation

- “Configuring WebSEAL worker threads” on page 41
- “Allocating worker threads for junctions (junction fairness)” on page 42
- “Configuration on AIX”

Configuring WebSEAL worker threads

The number of configured worker threads specifies the number of concurrent incoming requests that can be serviced by a server. Other connections that arrive when all worker threads are busy will be buffered until a worker thread is available.

You can set the number of threads available to service incoming connections to WebSEAL. Configuring the number of worker threads should be done carefully due to possible performance impacts.

This configuration parameter does not impose an upper boundary on the number of simultaneous connections. This parameter simply specifies the number of threads made available to service a potentially unlimited work queue.

Choosing the optimal number of worker threads depends on understanding the quantity and type of traffic on your network. In all cases, you must only enter a value that is less than the worker threads limit imposed by the operating system.

By increasing the number of threads, you are, in general, decreasing the average time it takes to finish the requests. However, increasing the number of threads impacts other factors that could have an adverse effect on server performance.

WebSEAL maintains a single, generic worker list and worker threads pool for handling requests from clients using TCP or SSL. This enhanced mechanism enables WebSEAL to consume fewer system resources while handling significantly greater load.

You can configure the worker thread pool size by setting the worker-threads parameter in the [server] stanza portion of the WebSEAL configuration file.

```
[server]
worker-threads = 50
```

**Note:** The value of this parameter must remain within the worker threads limits set by the operating system.

Configuration on AIX

On AIX systems only, when the WebSEAL worker threads limit in the [server] stanza of the WebSEAL configuration file is increased beyond the default of 50 to a value greater than 800 on an AIX system, WebSEAL can fail and produce a core dump.

Workaround: Increase the AIX process size limits to unlimited as follows:

1. Locate the limits file in the /etc/security directory.
2. Search for the word default and find parameters for cpu, rss, and data. Default settings are as follows:

   ```
data = 262144
rss = 65536
```
3. Change the settings for the data and rss parameters to unlimited using the –1 value. The –1 setting specifies unlimited or is bounded by the capabilities of the operating system and the CPU.

\[
data = -1 \\
rss = -1
\]

4. Reboot the system for these changes to take effect.

**Allocating worker threads for junctions (junction fairness)**

You can configure the allocation of WebSEAL worker threads used to process requests across multiple junctions on a global or per-junction basis. The configuration mechanism maintains a “fair” distribution of worker threads across all junctions and prevents depletion of the worker thread pool by any one junction.

**Background**

WebSEAL draws from its pool of worker threads to process multiple requests. The number of worker threads available to WebSEAL is specified by the worker-threads parameter in the WebSEAL configuration file.

You can adjust the worker-threads value to best serve your particular WebSEAL implementation. When no worker threads are available to handle incoming requests, users experience a WebSEAL server that is not responding.

Worker threads are used to handle incoming requests to applications residing on multiple junctioned back-end servers. However, the worker thread pool can be quickly drained if a particular back-end application is unusually slow when responding to and processing a high volume of requests. A depletion of the worker thread pool by this one application renders WebSEAL incapable of responding to requests for services on the remaining junctioned application servers.

You can configure global or per-junction limits on the number of worker threads used to service applications on multiple junctions. These limits allow “fairness” to prevail for all junctions and prevents any one application from claiming more than its share of worker threads.

**Note:** For information on worker thread resource usage limits, and for instructions on detecting worker thread starvation, see the *IBM Tivoli Access Manager for e-business Performance Tuning Guide*

**Global allocation of worker threads for junctions**

Two parameters located in the [junction] stanza of the WebSEAL configuration file control the global allocation of worker threads across all junctions for a particular WebSEAL server. The values used for these parameters are expressed as percentages within the range of 0 to 100.

- **worker-thread-soft-limit**
  This parameter is set to send a warning before the “hard” limit is reached. When the **worker-thread-soft-limit** is exceeded, warning messages are sent (every 30 seconds) to the WebSEAL error log file.

  For example, when **worker-threads** = 50, a setting of 60 (%) causes warning messages to be issued when the junction consumes more than 30 worker threads. All requests above 30 worker threads are still processed, until the “hard” limit is reached.

  The default value is 90 percent.

- **worker-thread-hard-limit**
This parameter determines the cut-off point for servicing requests across a junction. When the worker-thread-hard-limit is exceeded, error messages are sent (every 30 seconds) to the WebSEAL error log file. In addition, the user is sent a 503 “Service Unavailable” message.

For example, when worker-threads = 50, a setting of 80 (%) causes error messages to be issued when the junction tries to consume more than 40 worker threads. All requests representing greater than 40 worker threads on the junction are returned with a 503 “Service Unavailable” message.

The default value of 100 (%) indicates there is no limit.

These global settings apply equally to all configured junctions. When configuring these two parameters, it is logical to set the ”soft” limit to a lower value than the ”hard” limit.

**Per-junction allocation of worker threads for junctions**

Alternatively, you can limit worker thread consumption on a per-junction basis. The following options to the `pdadmin server task create` command allow you to specify hard and soft worker thread limits on a specific junction:

- `–l percent_value`
  
  This option sets a value (percent) on the junction that defines the soft limit for consumption of worker threads. As in the global soft limit setting, this option causes warning messages to be issued when the junction consumes more worker threads than allowed by the setting.

- `–L percent_value`
  
  This option sets a value (percent) on the junction that defines the hard limit for consumption of worker threads. As in the global hard limit setting, this option causes warning messages to be issued when the junction tries to consume more worker threads than allowed by the setting. In addition, the user is sent a 503 “Service Unavailable” message.

For example:

```bash
pdadmin> server task webseald-<server-name> create -t tcp -h <host-name> \ 
-1 60 -L 80 <jct-point>
```

Per-junction settings always override the global settings in the WebSEAL configuration file. Ensure that the settings on a specific junction do not adversely affect the policy established by the global settings.

**Troubleshooting notes**

- You can use the `pdadmin server task show` command to view the number of active worker threads on a specific junction:

```bash
pdadmin> server task webseald-<server-name> show /<jct-point>
```

This information might be useful when you want to determine the location of a junction that is absorbing more than its share of worker thread resources.

- If you specify a soft limit value that is greater than the hard limit value on a specific junction, the junction will not be created.

- You must specify both soft and hard limit values (both –l and –L options) on a specific junction.
Multi-locale support with UTF-8

This section consists of the following topics:

- “Multi-locale support concepts”
- “Configuring multi-locale support” on page 48

Multi-locale support concepts

This section contains the following topics:

- “WebSEAL data handling using UTF-8”
- “UTF-8 dependency on user registry configuration” on page 45
- “UTF-8 data conversion issues” on page 45
- “UTF-8 environment variables for CGI programs” on page 46
- “UTF-8 impact on authentication” on page 46
- “URLs must use only one encoding type” on page 47
- “UTF-8 support during WebSEAL upgrade” on page 47

WebSEAL data handling using UTF-8

WebSEAL Version 5.1 support multiple locales. This support enables WebSEAL to process data from multiple languages at once. WebSEAL implements multi-locale support by internally maintaining and handling all data using UCS Transformation Format 8 byte (UTF-8) encoding. UTF-8 is a multi-byte code page with variable width.

Prior versions of WebSEAL provided limited multi-locale support. While WebSEAL could support a variety of languages as requested by a browser (client), WebSEAL could support only one language at a time. In Version 5.1, WebSEAL adopts UTF-8 as the default code page for all internal data handling.

WebSEAL administrators can configure how WebSEAL handles data input and output. An example of data input is characters sent to WebSEAL by a browser, such as user logins and forms data. An example of data output is logging information written out to the filesystem by the Tivoli Access Manager event logging manager.

The change within WebSEAL to use UTF-8 should be transparent to administrators whose systems do not need to provide multi-locale (and multiple language) supports.

WebSEAL handles data internally in UTF-8 regardless of the locale in which the WebSEAL process is running. When locale-specific data is needed as input or output, the locale in which the WebSEAL process is running becomes important. Note that most operating systems do not use UTF-8 by default. Administrators expecting locale-specific behavior need to know which locale is being used, and need to set WebSEAL UTF-8 configuration options to match the required behavior.

The system locale consists of two parts: the language and the local code page. Local code pages can be by UTF-8 or not UTF-8. Historically, most operating systems use a local code page that is not UTF-8. For example, a common local code page used to represent the 8-bit ASCII character set for United States English is en_US.ISO88591, which uses the ISO-8859-1 character set.

Administrators running systems that need to process client requests and forms data in the local code page will want to modify the default settings for URL
support (utf8-url-support-enabled) and forms support (utf8-forms-support-enabled). The default WebSEAL setting is to consume data in UTF-8 format only.

For example, the need to change default settings applies to administrators running systems that need to process clients requests and forms data use non-UTF-8 local code pages, such as those for:

- A single-byte Latin character set, such as Spanish, French, or German
- A multi-byte character set, such as Japanese or Chinese

If you are running systems that need to provide true multi-locale support to handle users and data in multiple languages, you should review your local code page setting and consider converting to a UTF-8 code page. Also review the default WebSEAL multi-locale UTF-8 settings. You might want to customize configuration settings to best fit your deployment.

**UTF-8 dependency on user registry configuration**

For optimal multi-locale support, all users should be stored in one common user registry, regardless of which language they prefer. Most user registries support UTF-8 by default. Some LDAP user registries, and their supporting databases, can optionally be configured to not support UTF-8. Ensure that the LDAP user registry and database used with Tivoli Access Manager uses UTF-8.

IBM Directory Services is by default configured to use UTF-8.

**UTF-8 data conversion issues**

WebSEAL can be deployed into environments where the local code page uses UTF-8. Similarly, WebSEAL can also be deployed into environments where the local code page does not use UTF-8. The use of Version 5.1 WebSEAL with operating system environments that use non-UTF-8 local code pages requires WebSEAL to convert data upon data input and output. When WebSEAL reads data in, it must convert the data from non-UTF-8 to UTF-8. When WebSEAL writes data out, it must convert the data from UTF-8 to non-UTF-8.

If conversion to a local code page is required, no data loss will occur when running in a UTF-8 locale.

The conversion from a UTF-8 locale to a non-UTF-8 locale (local code page) can, in some situations, result in data loss.

Conversion of data from UTF-8 to a non-UTF-8 locale can result in data loss. For example, if WebSEAL is running in an en_US.ISO8859 environment, and a Japanese user name must be converted to the local code page, the result is a string of question marks ("????"). This occurs because there is no way to represent Japanese characters in ISO-8859-1. For this reason, it is recommended that WebSEAL is run using UTF-8.

There is a risk of data loss when executing administrative commands (pdadmin) from a non-UTF-8 environment. Prior to Version 5.1, WebSEAL always ran in the same locale as the pdadmin utility. With multi-locale support, WebSEAL can now run in a different locale. WebSEAL must return messages to the administrator in the administrator’s chosen language. To do so, WebSEAL obtains messages from the appropriate language pack, as determined by the locale presented by pdadmin. All messages are transmitted in UTF-8, but pdadmin converts those messages to local code page prior to displaying them. When the local code page is non-UTF-8, data loss is potentially possible. When pdadmin is run in a UTF-8 environment, there will be no data loss.
WebSEAL generates logging and auditing data using UTF-8. To prevent possible data loss, it is recommended that UTF-8 be used to write the data to the appropriate logging and auditing files. When the local code page is non-UTF-8, data must be converted to non-UTF-8 before it can be written. In this case, the possibility of data loss exists.

All log audit files generated by WebSEAL are in the language specified by the locale in which the server runs. The code page used to write the messages is configurable in the WebSEAL routing file. For example, on UNIX systems, the file is `/opt/pdweb/etc/routing`.

**UTF-8 environment variables for CGI programs**

CGI scripts use environment variables are used to communicate with WebSEAL, and the environment variables must be in the local code page. Legacy CGI scripts expect raw (binary) local code page strings.

To enable CGI scripts to understand environment variable values that can consist of UTF-8 data, WebSEAL provides additional environment variables. These variables have the same names as current CGI variables, but with the characters "_UTF8" appended to the end. The values of these variables are URI (Uniform Resource Indicator) encoded UTF-8 strings. URI encoding is used to prevent data loss on platforms which expect local code page environment variables in spawned processes.

The variables are:
- `REMOTE_USER_UTF8`
- `IV_USER_UTF8`
- `HTTP_IV_USER_UTF8`
- `IV_GROUPS_UTF8`
- `HTTP_IV_GROUPS_UTF8`

New CGI programs should use these variables because their values contain UTF-8 data. WebSEAL stores the data for these variables internally in UTF-8 format. The data must be converted to local code page in order for CGI programs to use it. When the old CGI variables (for example, `REMOTE_USER`) are used, and the local code page is not UTF-8 encoded, the conversion of the UTF-8 data to the local code page can, in some cases, result in data corruption.

**UTF-8 impact on authentication**

The use of UTF-8 for internal data handling has the following impacts on WebSEAL’s processing of authentication requests:

- UTF-8 logins over basic authentication not supported
  
  Use of UTF-8 with basic authentication (BA) login is not supported. UTF-8 logins with BA cannot be supported because browsers transmit data in inconsistent ways. In prior releases, WebSEAL did not support multi-byte BA logins because of browser inconsistency. This is unchanged for Version 5.1.

  WebSEAL consumes BA login strings with the expectation that they are in local code page. WebSEAL supports 7-bit ASCII and single-byte Latin code pages. Thus, for example, a server that wants to allow French users to use BA logins must run in a Latin locale. WebSEAL consumes the BA login string and converts it to UTF-8 internally. However, if the French user has a UTF-8 code page, a BA login is not available, because the login string will be multi-byte.

- Forms login
In previous versions of WebSEAL, forms login data was always consumed by WebSEAL with the auto functionality. This meant that WebSEAL examined the login data to see if it was in UTF-8 format. If the data was not in UTF-8 format, the data was processed as local code page. In Version 5.1, this setting is configurable, as described in “UTF-8 support for forms” on page 49.

- Cross-domain single sign-on, e-community single sign-on, and failover authentication
  Each of these authentication methods employs encoded tokens. The encoding of these tokens must be configured to either use UTF-8 encoding or non-UTF-8 encoding. For more information, see “Configuring multi-locale support” on page 48.

- Conversion shared library
  To maintain backwards compatibility with customized authentication libraries, such as CDASs, that expect data in non-UTF-8 format, WebSEAL provides a conversion shared library for use during authentication. This library automatically converts data between non-UTF-8 and UTF-8 format. Use of this library requires a specific configuration step. For more information, see the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

**URLs must use only one encoding type**
WebSEAL requires that any URL presented for processing contain only a single character encoding type such as UTF-8 or ShiftJIS. When URLs contain multiple character encoding types, WebSEAL cannot guarantee the accuracy of the data in the request because the decoded value of the UTF-8 characters might not match the decoded value of the same characters in the local code page. This possible inaccuracy in the data could cause WebSEAL to mistakenly grant unauthorized users access to protected objects.

When WebSEAL encounters a URL with multiple character encoding types, the URL is returned as a Bad Request.

**UTF-8 support during WebSEAL upgrade**
Upgrading of WebSEAL from a prior version (prior to Version 5.1) results in the following configuration:
- The value of the existing configuration option `utf8-url-support-enabled` is preserved.
- The new configuration option `utf8-form-support-enabled` is set to auto. This setting preserves the behavior of existing WebSEAL servers.
  ```
  [server]
  utf8-form-support-enabled = auto
  ```
- All existing WebSEAL junctions are migrated with the following option:
  ```
  -e lcp_bin
  ```

Use of this value allows existing environments and applications to operate without change.

Note that this is not the default WebSEAL Version 5.1 configuration value.
Configuring multi-locale support

Configuration topics for multi-locale support:

- "UTF-8 support for uniform resource locators"
- "UTF-8 support for forms" on page 49
- "UTF-8 support in query strings" on page 50
- "UTF-8 encoding of tokens for cross domain single sign-on" on page 50
- "UTF-8 encoding of tokens for e-community single sign-on" on page 51
- "UTF-8 encoding of cookies for failover authentication" on page 51
- "UTF-8 encoding in junction requests" on page 51

UTF-8 support for uniform resource locators

Browsers are limited to a defined character set that can legally be used within a Uniform Resource Locator (URL). This range is defined to be the printable characters in the ASCII character set (between hex code 0x20 and 0x7e). For non-English languages, and other purposes, characters outside the printable ASCII character set are often required in URLs. These characters can be encoded using printable characters for transmission and interpretation.

There are a number of different encoding methods for transmitting characters outside the printable ASCII range. WebSEAL, acting as a Web proxy, must be able to handle all these cases. The UTF-8 locale support addresses this need.

The manner in which WebSEAL processes URLs from browsers can be specified in the WebSEAL configuration file:

```
[server]
utf8-url-support-enabled = {yes|no|auto}
```

The three possible values are as follows:

- **yes**
  
  In this mode, WebSEAL only recognizes URI encoded UTF-8 data in URL strings and they are used without modification. These UTF-8 characters are then validated and taken into account when determining access rights to the URL. WebSEAL supports both raw UTF-8 and URI encoded UTF-8 strings in URLs. In this mode, other encoding techniques are not accepted.
  
  This is the default value, and is recommended.
  
  Servers that run in an 7-bit ASCII English locale should use this value.

- **no**

  In this mode, WebSEAL does not recognize UTF-8 format data in URL strings. Used for local code page only. If the string can be validated it is converted to UTF-8 for internal use.

  Servers that do not need to process multi-byte input, and are running in a single-byte Latin locale, such as French, German, or Spanish, should use this setting.

  Use this setting when supporting existing applications and Web servers do not function correctly with WebSEAL if UTF-8 support is enabled. These applications might use DBCS (such as Shift-JIS) or other encoding mechanisms in the URL.

  **Note:** When setting this value to no, ensure that all junctioned servers do NOT accept UTF-8 format URLs. It is important from a security perspective, that WebSEAL interprets URLs in the same manner as the junctioned servers.

- **auto**
WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. WebSEAL correctly processes any correctly constructed UTF-8 encoding. If the encoding does not appear to be UTF-8, then the coding is processed as DBCS or Unicode.

If a URL has Unicode in the format "%uHHHH", WebSEAL converts it to UTF-8. The rest of the decoding proceeds as if the configuration settings was yes. If the double-byte-encoding option in the [server] stanza is set to yes, WebSEAL converts %HH%HH to UTF-8.

Servers running in a single-byte Latin locale that need to process multi-byte strings should use the auto setting.

Servers running in a multi-byte locale but just need to support one language (for example, Japanese) can use the auto setting.

A recommended deployment strategy is as follows:

1. Unless required for content purposes, immediately check and set the default-webseal ACL on existing production deployments to NOT allow unauthenticated "r" access. This limits security exposure to users who do have a valid account within the Tivoli Access Manager domain.

2. Ensure that the utf8-url-support-enabled parameter is set to the default value of yes.

3. Test your applications. If they function correctly, use this setting.

4. If any applications fail with "Bad Request" errors, retry the application with the utf8-url-support-enabled parameter set to no. If this works, you may deploy with this setting. Ensure, however, that no junctioned Web server is configured to accept UTF-8 encoded URLs.

5. If the application continues to have problems, try setting utf8-url-support-enabled to auto.

**UTF-8 support for forms**

The manner in which WebSEAL processes data from forms (for example, a login form) can be specified in the WebSEAL configuration file:

```
[server]
utf8-form-support-enabled = {yes|no|auto}
```

The forms providing data to the server are forms that are part of WebSEAL, such as login forms. These forms all declare the character set to be UTF-8. Thus the default value is yes. If an administrator edits these forms and changes the character set to a non-UTF-8 setting, such as a local code page, this configuration setting should be changed. If some forms use UTF-8 and some use a local code page, use the auto value. If all forms are modified to use a non-UTF-8 setting, use the no value.

The three possible values are as follows:

- **yes**

  WebSEAL only recognizes UTF-8 encoding in forms and the data is used without modification. These UTF-8 characters are then validated and taken into account when processing the data. Other encoding techniques are not accepted. When double-byte-encoding is set to yes, Unicode of the form %HH%HH is supported. When a double-byte Unicode character is detected, the entire string must be double-byte encoded.

  This is the default value, and is recommended.

- **no**
WebSEAL does not recognize UTF-8 encoding in forms. Used for local code page only. If the form data can be validated it is converted to UTF-8 for internal use.

- auto
  WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. WebSEAL correctly processes any correctly constructed UTF-8 input. If the encoding does not appear to be UTF-8, then the coding is processed as non-UTF-8.

**UTF-8 support in query strings**
The manner in which WebSEAL processes data from query strings can be specified in the WebSEAL configuration file:

```
[server]
utf8-qstring-support-enabled = {yes|no|enabled}
```

The default setting is no. Thus, WebSEAL default behavior is to assume all query strings are local code page.

The three possible values are as follows:

- yes
  WebSEAL only recognizes UTF-8 encoding in query strings and the data is used without modification. These UTF-8 characters are then validated and taken into account when processing the data. Other encoding techniques are not accepted. Use this setting when your WebSEAL server must process query strings that use UTF-8.

  Servers that operate in an single-byte Latin locale, such as French, German, or Spanish, and process queries from an application that uses UTF-8, should use this setting. Servers that operate in a multi-byte locale and process only UTF-8 query strings can use this setting.

- no
  WebSEAL does not recognize UTF-8 encoding in query string. Used for local code page only. If the form data can be validated it is converted to UTF-8 for internal use.

  This is the default value, and is recommended.

  Servers that operate in a 7-bit ASCII English locale can use this setting.

- auto
  WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding (DBCS and Unicode). WebSEAL correctly processes any correctly constructed UTF-8 encoding. If the encoding does not appear to be UTF-8, then the coding is processed as DBCS or Unicode.

  Servers that operate in a multi-byte locale and process a mixture of UTF-8 and non-UTF-8 query strings can use this setting.

  Servers that operate in an single-byte Latin locale, such as French, German, or Spanish, and process a mixture of UTF-8 and non-UTF-8 query strings can use this setting.

**UTF-8 encoding of tokens for cross domain single sign-on**
The use of UTF-8 encoding for strings within tokens used for cross domain single sign-on is specified in the WebSEAL configuration file.

```
[cdsso]
use-utf8 = {true|false}
```

The default value is true.
When use-utf8 is set to false, strings are encoded using the local code page. Use this value when implementing cross domain single sign-on with older (pre-Version 5.1) WebSEAL servers. WebSEAL servers from versions prior to 5.1 do not use UTF-8 encoding for tokens. When deploying an environment that includes these older servers, configure the Version 5.1 WebSEAL server to not use UTF-8 encoding. This setting is necessary for backwards compatibility.

Note: Note that when this value is set to false, data loss can occur during conversion from UTF-8 to a non-UTF-8 local code page.

**UTF-8 encoding of tokens for e-community single sign-on**
The use of UTF-8 encoding for strings within tokens used for e-community single sign-on is specified in the WebSEAL configuration file.

```
[e-community-sso]
use-utf8 = {yes|no}
```

The default value is yes.

When use-utf8 is set to no, strings are encoded using the local code page. Use this value when implementing cross domain single sign-on with older (pre-Version 5.1) WebSEAL servers. WebSEAL servers from versions prior to 5.1 do not use UTF-8 encoding for tokens. When deploying an environment that includes these older servers, configure the Version 5.1 WebSEAL server to not use UTF-8 encoding. This setting is necessary for backwards compatibility.

Note: Note that when this value is set to no, data loss can occur during conversion from UTF-8 to a non-UTF-8 local code page.

**UTF-8 encoding of cookies for failover authentication**
The use of UTF-8 encoding for strings within failover authentication cookies is specified in the WebSEAL configuration file.

```
[failover]
use-utf8 = {yes|no}
```

The default value is yes.

When use-utf8 is set to no, failover authentication cookies are encoded using the local code page. Use this value when implementing failover authentication with older (pre-Version 5.1) WebSEAL servers. WebSEAL servers from versions prior to 5.1 do not use UTF-8 encoding for failover authentication cookies. When deploying an environment that includes these older servers, configure the Version 5.1 WebSEAL server to not use UTF-8 encoding. This setting is necessary for backwards compatibility.

Note: Note that when this value is set to no, data loss can occur during conversion from UTF-8 to a non-UTF-8 local code page.

**UTF-8 encoding in junction requests**
WebSEAL inserts information into HTTP headers for requests to the backend server. This information can include extended attributes or user data. In WebSEAL versions prior to 5.1, the headers were added to the request using raw local code page. In Version 5.1, the header data is transmitted in a configurable format.

By default, WebSEAL now adds information to HTTP headers using a UTF-8 code page. This prevents any potential data loss that could occur when converting to a non-UTF-8 code page. Also by default, this data is sent URI encoded. For
backwards compatibility, the format of the header data can be configured to raw local code page. In addition, two other formats are supported: Raw UTF-8 and URI encoded local code page.

The -e option for creating junctions specifies the encoding of user name, groups, and other extended attributes which are sent within the HTTP header to the backend server. The encode option can take one of the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| utf8_uri   | URI encoded UTF-8 data  
All white space and non-ascii bytes are encoded %XY, where X and Y are hex values (0–F). |
| utf8_bin   | Un-encoded UTF-8 data.  
This setting allows data to be transmitted without data loss, and the customer does not need to URI-decode the data.  
This setting should be used with caution, because it is not part of the HTTP specification |
| lcp_uri    | URI encoded local code page data  
Any UTF-8 characters which can not be converted to local code page will be converted to question marks (?). Use this option with caution and only in environments where local code page produces the desired strings. |
| lcp_bin    | Un-encoded local code page data.  
This mode was used by versions of WebSEAL prior to Version 5.1. Use of this mode enables migration from previous versions, and is used in upgrade environments.  
Note that data loss can potentially occur with this mode. Use with caution. |
Multi-locale messages

Standard WebSEAL server responses to client browsers, such as error messages, custom HTML login and logout pages, and serviceability messages, can be delivered in the client’s preferred language.

WebSEAL supports multi-locale capabilities by using the values contained in the `accept-language` HTTP header to determine the correct language for server-generated messages and HTML pages. Translated information is available for the following server resources:

- HTTP error messages
  `install_path/www/lib/errors/locale_directory`
- Custom account management pages
  `install_path/www/lib/html/locale_directory`
- Serviceability messages

Browsers adopt a standard set of language values. Basic language values are represented by two characters, indicating the language. Location-specific values are expressed in a two part format, indicating the language and the country where this version of the language is used. Examples include:

- `es` (Spanish)
- `de` (German)
- `en` (English)
- `it` (Italian)
- `en-US` (English/United States)
- `en-GR` (English/United Kingdom)
- `es-ES` (Spanish/Spain)
- `es-MX` (Spanish/Mexico)
- `pt-BR` (Portuguese/Brazil)

The `accept-language` header can include more than one language. Each additional language is separated by a comma. For example:

```
accept-language: es-mx,es,en
```

The order in which the values appear in the header determine the hierarchy of importance. WebSEAL checks the first listed value for an existing installed language pack. If no language pack for this language is installed, WebSEAL checks the next language in the list for its associated language pack.

*Note:* The `accept-language` header can use a "q=x.x" parameter to express a preference level for a language. This parameter is not recognized by WebSEAL. The listed order of languages in the header determines the order of priority for WebSEAL.

Several language packs for WebSEAL server messages are available for installation. Each language pack installation creates a locale-specific sub-directory within the path for each message storage location. For example, a Spanish language pack creates the following sub-directory for the server error message storage location:

```
install_path/www/lib/errors/es
```

The following table lists the languages supported by WebSEAL, with the associated sub-directory name:
### Table: Language System Directory

<table>
<thead>
<tr>
<th>Language</th>
<th>System Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>English (default)</td>
<td>C</td>
</tr>
<tr>
<td>Czech</td>
<td>cs</td>
</tr>
<tr>
<td>German</td>
<td>de</td>
</tr>
<tr>
<td>Spanish</td>
<td>es</td>
</tr>
<tr>
<td>French</td>
<td>fr</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu</td>
</tr>
<tr>
<td>Italian</td>
<td>it</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja</td>
</tr>
<tr>
<td>Korean</td>
<td>ko</td>
</tr>
<tr>
<td>Polish</td>
<td>pl</td>
</tr>
<tr>
<td>Portuguese, Brazil</td>
<td>pt_BR</td>
</tr>
<tr>
<td>Russian</td>
<td>ru</td>
</tr>
<tr>
<td>Chinese, China</td>
<td>zh_CN</td>
</tr>
<tr>
<td>Chinese, Taiwan</td>
<td>zh_TW</td>
</tr>
</tbody>
</table>

The following example process flow illustrates how WebSEAL evaluates the `accept-language` header:

1. The `accept-language` header contains `pt-br` as the first value in the list.
2. The `pt-br` language is converted to `pt_BR`, representing the WebSEAL language sub-directory for this language.
3. If this sub-directory does not exist for the required message (for example, no language pack is installed for this language), WebSEAL checks for a `pt` directory.
4. If no `pt` directory exists, WebSEAL attempts to find message sub-directories for the next language listed in the header.
5. If there are no installed language packs for all messages listed in the header, WebSEAL defaults to the language environment which WebSEAL is running in, as determined by the LC_ALL or LANG environment variables set on the operating system.

**Conditions affecting multi-locale support on WebSEAL:**

- Multi-locale support is enabled at all times on the WebSEAL server.
- Installation of specific language packs determines what languages are supported.
- WebSEAL always returns the UTF-8 character set to the user, regardless of what the `accept-charset` HTTP header value requests.
- If WebSEAL accesses a locale directory for a translated message, and the directory is empty (for example, the contents were removed by the administrator), a server error page is returned.
Handling invalid character encoding in URL query strings

Problem:

It is possible for the query string of a URL to contain character encoding that is unacceptable to WebSEAL, and therefore rejected by WebSEAL. For example, a query string containing binary encoding is rejected by WebSEAL.

The problem of invalid character encoding is caused by the specific requirements of the back-end server application. In a typical scenario, the client makes a request to this back-end application. The request includes a query string, required by the back-end application, but containing character encoding unknown to WebSEAL. WebSEAL returns a "Bad Request" (400) error. The error log contains a message such as: "Illegal character in URL."

Solution:

The solution to the problem of invalid character encoding is to configure WebSEAL to completely ignore the query string. The query string component of the URL can then be passed unchanged to the back-end application.

However, if WebSEAL is configured to ignore the query string component, then dynamic URL mapping for authorization checking must be disabled. If this compromise is acceptable for your WebSEAL environment, perform the following two changes to the WebSEAL configuration file:

1. Disable the dynurl-map parameter with a comment character:
   
   ```
   [server]
   #dynurl-map = bin/dynurl.conf
   ```

2. Manually add the decode-query parameter and set it equal to "no":
   
   ```
   [server]
   decode-query = no
   ```

   The default behavior when decode-query is not defined is:

   ```
   decode-query = yes
   ```

   If you disable dynurl-map and fail to add decode-query, the WebSEAL process does not start.
Preventing vulnerability caused by cross-site scripting

Cross-site scripting refers to a technique used to cause Web server vulnerability by embedding malicious code into the URLs of Web requests. WebSEAL provides certain built-in protection for this type of vulnerability and allows you to further refine the protection by configuring URL string filtering.

Note: The term "cross-site scripting", although accepted by the industry, does not entirely describe the range of issues involving embedded malicious code.

Background

Cross-site scripting is a specific type of Web Server vulnerability created when a client URL request includes embedded malicious scripting. For example (Javascript):

```
<script>malicious_code</script>
```

Other scripting tags that could be used to create vulnerability include <OBJECT>, <APPLET>, and <EMBED>. When a user clicks on a link containing the malicious code (or enters such a URL directly), the script is executed when the HTML is read by the user’s browser.

For example, an attack can occur when a user clicks on a link that contains the following URL:

```
https://<webseal-host>/<script>malicious_code</script>
```

In this example, the object is not found and WebSEAL responds by returning a 404 "Page Not Found" HTML error page. This error page happens to include the URL containing the malicious Javascript. The browser interprets the URL and executes the script.

Refer to the following CERT advisory for complete information about the mechanics of cross-site scripting and general preventative measures:

```
http://www.cert.org/advisories/CA-2000-02.html
```

Configuring URL string filtering

The problem of cross site scripting—and embedded malicious code in general—is handled in two ways.

WebSEAL encodes angle brackets (< >) in re-directed URLs. The encoding can help prevent normal interpretation of scripts by the browser.

You can define string patterns to be illegal by adding entries to the [illegal-url-substrings] stanza in the WebSEAL configuration file. For example:

```
[illegal-url-substrings]
substring = <script
substring = <applet
substring = <embed
```

If WebSEAL detects any configured string fragment in the requested URL, the URL is deemed illegal and not accepted. WebSEAL returns a 400 "Bad Request" error page.

This flexible mechanism allows you to handle future attack schemes by adding additional substring values.
WebSEAL, by default, filters strings containing `<script`. You do not need to manually add the `[illegal-url-substrings]` stanza to filter this particular string. However, when you require additional filtering, you must create the stanza and list all substrings individually, as in the example above.

You can completely disable the URL string filtering feature (including the default behavior) by placing an empty `[illegal-url-substrings]` stanza into the WebSEAL configuration file.

Functional notes:

- Substring entries in the configuration file must be ASCII. Multi-byte characters in URLs are fine, but the configuration file entries must be ASCII.
- Substrings are located using a case insensitive search
- Substring filtering accommodates multi-byte characters
- The mechanism protects junctioned servers
Replicated front-end WebSEAL servers

Note: The following information replaces the former `pdadmin server modify baseurl` command, used in previous versions of Tivoli Access Manager.

In a heavy load environment, it is advantageous to replicate front-end WebSEAL servers to provide better load-balancing and fail-over capability. When you replicate front-end WebSEAL servers, each server must contain an exact copy of the Web space, the junction database, and the dynurl database.

This version of Tivoli Access Manager supports a manual configuration procedure to replicate front-end WebSEAL servers. The `pdadmin` command is no longer used for this task.

In the following example, "WS1" is the host name of the primary WebSEAL server. "WS2" is the host name for the replica WebSEAL server.

1. Install and configure WebSEAL on both WS1 and WS2 servers.
2. Using the `pdadmin` command, create a new object to be the root of the authorization space for both WebSEAL servers. For example:
   ```
padmin> object create /WebSEAL/newroot
   ```
4. On WS1, change the value of the `server-name` parameter in the WebSEAL configuration file from "WS1" to "newroot":
   ```
   [server]
   server-name = newroot
   ```
5. Restart WebSEAL on WS1.
6. Repeat Steps 3-5 for WS2.

The WS1 and WS2 servers now use the object `/WebSEAL/newroot` as the base for authorization evaluations. Either the WS1 or the WS2 server can respond to `object list` and `object show` commands for objects residing below `/WebSEAL/newroot`.

Use the following procedure when unconfiguring either WS1 or WS2:

1. Stop the WebSEAL server.
2. Change the `server-name` parameter back to its original value. For example, for WS1:
   ```
   [server]
   server-name = WS1
   ```
3. Proceed with normal unconfiguration procedures.

Conditions:

- Unified object space management: Although a single object hierarchy is visible to the administrator, all replicated WebSEAL servers are affected by administration commands applied to that object hierarchy and all servers are able to respond to these commands.
- Unified authorization evaluation: Both WS1 and WS2 use `/WebSEAL/newroot` as the base for authorization evaluations.
- Unified configuration: For front-end WebSEAL replication to function correctly, the Web space, junction database, and dynurl database configuration must be identical on each server.
Platform for Privacy Preferences (P3P)

This section contains the following topics:

- “Compact policy overview”
- “Compact policy declaration” on page 60
- “Junction header preservation” on page 60
- “Default compact policy in the P3P header” on page 61
- “P3P header configuration” on page 62
- “Specifying a custom P3P compact policy” on page 67
- “Troubleshooting” on page 68

Compact policy overview

WebSEAL supports the Platform for Privacy Preferences (P3P) 1.0 specification. P3P is a standard for the declaration of privacy policies in a machine-readable format. The standard allows user agents to make decisions on the part of the user regarding whether to access certain URIs or accept certain cookies based on the policy presented by the Web site. In the absence of a policy, the decision can be made based on a set of assumptions about the site’s policy.

Commercial browsers support P3P, particularly as part of the decision process for accepting or rejecting cookies. Microsoft Internet Explorer 6 has P3P-based cookie filtering enabled by default. Browsers based on Mozilla provide optional P3P cookie filtering. WebSEAL provides P3P support to ensure that these browsers accept WebSEAL session cookies.

The P3P specification describes a compact policy and a full policy. A compact policy is a subset of a full policy. WebSEAL provides a default compact policy and also provides configuration settings to enable customization of the compact policy. WebSEAL does not provide a full policy. Full policies are specific to the vendor, application, or security environment into which WebSEAL is deployed. Implementation of a full policy is the responsibility of the vendor (service provider). WebSEAL includes a configuration setting that can be used to point clients to the location of a full policy.

The P3P specification states that an HTTP header can only have a single P3P header (additional P3P headers are ignored). However, an HTTP response can have multiple cookies. Thus, the compact policy specified in the HTTP header applies to all cookies in the response. Since there can be only a single policy, the policy must represent the most strict of the actual policies for the cookies. For WebSEAL, this means, for example, that if session cookies are accepted in a response but failover cookies are not, the worst case P3P policy should be returned for all cookies. The worst case is defined to be the minimum set of conditions that would cause the browser to reject the cookie.

WebSEAL returns three types of cookies to the user agent (browser):

- Session cookie
- Failover cookie
- e-community cookie

There is no need to configure policy for the e-community cookie. The cookie contents are limited to specifying the location of the Web server to which the user authenticated. This cookie contains no information that identifies the user.
The session cookie links to session data, and the failover cookie contains enough session information to enable reconstruction of the session. The session cookie is intended only for the origin server, is not retained past the end of the session, and assists in the process of session maintenance. The failover cookie is intended for the failover (replicated) server, is not retained past the end of the session, and also assists in the process of session maintenance. Thus, session and failover cookies have the same P3P policy. This means that the combined worst case policy for the cookies is the session cookie policy.

**Compact policy declaration**

The WebSEAL configuration file provides a set of configuration options that match the compact policy XML syntax as specified in the World Wide Web Consortium Platform for Privacy Preferences specification. The complete specification can be accessed at the following URL:

http://www.w3.org/TR/P3P/

WebSEAL provides configuration file entries that map to the following XML elements in the compact policy:

- **access**
  Indicates whether the site provides access to various kinds of information.
- **categories**
  The type of information stored in the cookie
- **disputes**
  Specifies whether the full P3P policy contains some information regarding disputes over the information contained within the cookie.
- **non-identifiable**
  This element signifies that either no data is collected (including Web logs), or that the organization collecting the data will make the data anonymous.
- **purpose**
  Purposes for data processing relevant to the Web.
- **recipients**
  The legal entity, or domain, beyond the service provider and its agents where data may be distributed.
- **remedies**
  Remedies in case a policy breach occurs.
- **retention**
  The type of retention policy in effect.
- **p3p-element**
  This entry can be used to specify any elements to add to the P3P header in addition to the compact policy. This can be used to supply a reference to a full XML policy.

The values for **purpose** (except current) and **recipients** (except ours) have an additional option describing how the cookie data can be used. This defines whether the user is given a choice to opt-in or opt-out.

**Junction header preservation**

WebSEAL enables you to specify whether P3P headers from junctioned applications are preserved or replaced. Note that this is not part of the P3P compact policy, but is a WebSEAL function.
The configuration file entry is:

```
[p3p-header]
preserve-p3p-policy = {yes|no}
```

The default behavior is no. This means that P3P headers from junctioned servers are replaced.

WebSEAL replaces backend P3P policy headers by default to ensure that WebSEAL cookies are not excluded due to a more strict policy set by the backend server.

When using the default setting, you might find that cookies that the backend server must set are not allowed due to the WebSEAL compact policy. In this case, you should choose one of the following options:

- Set `preserve-p3p-policy = yes` to force WebSEAL to preserve the compact policy set by the backend server.
- Modify the WebSEAL compact policy header to make the policy more permissive, so that backend cookies are allowed.

When WebSEAL processes responses from backend servers, WebSEAL’s actions can include the addition of a cookie to the response. This occurs when the WebSEAL junction has been created to generate junction cookies. These cookies are used to map URLs across junctions, to ensure connectivity between the browser and the backend server. This means that when the administrator chooses to preserve the compact policy set by the backend server (`preserve-p3p-policy = yes`), the administrator must ensure that the compact policy is permissive enough to accept the addition of the WebSEAL junction cookie. When the compact policy forbids the addition of the junction cookie, the URL requests from the browser will not successfully resolve to the URLs on the backend server.

**Default compact policy in the P3P header**

WebSEAL adds a P3P header to every response in which cookies are set. The header contains a P3P Compact Policy. The policy is a sequence of terms that describe the policy regarding information contained within the cookies in the response.

The following WebSEAL configuration file entries represent the default P3P compact policy:

```
[p3p-header]
access = none
purpose = current
purpose = other-purpose:opt-in
recipients = ours
retention = no-retention
categories = uniquoid
```

The default configuration file entries result in a P3P header with the following contents:

```
P3P: CP="NON CUR OTPi OUR NOR UNI"
```

The following table explains the values in the default policy header:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON</td>
<td>User has no access to information either in the cookie or linked to by the cookie.</td>
</tr>
</tbody>
</table>
Table 4. P3P default header values (continued)

<table>
<thead>
<tr>
<th>CUR</th>
<th>Cookie helps provide the current service. The current service is the access to the protected Web site.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTPi</td>
<td>Cookie provides another service, to which the user has opted-in.</td>
</tr>
<tr>
<td>OUR</td>
<td>The Web site itself is the only recipient of the cookie and the information linked to by the cookie</td>
</tr>
<tr>
<td>NOR</td>
<td>Neither the cookie data nor the data to which it links is retained after the user logs out or after the user session expires.</td>
</tr>
<tr>
<td>UNI</td>
<td>The cookie uses a unique identifier that represents the user, by using the session ID and the user name.</td>
</tr>
</tbody>
</table>

P3P header configuration

User that deploy WebSEAL servers as part of the security solution for their Web servers must specify the P3P compact policy for their site. This step requires determining policy for each of the privacy settings defined by the P3P specification. WebSEAL provides a default policy that is accepted by the default settings for the Microsoft Internet Explorer 6 browser. Web administrators should modify the default policy as needed to match the site policies for handling of user data in cookies. Web administrators should test use of their policies with IE 6 to ensure that the WebSEAL cookies continue to be accepted by IE 6 browsers.

Web administrators should consult the P3P specification when defining their site policy.

Multiple values are allowed for each configuration entry, with the exception of the entries that require a value of yes or no. When a particular configuration entry is not declared, no indicators are added to the compact policy for that entry.

To configure the P3P compact policy for use with WebSEAL, complete the following steps:

1. Open the WebSEAL configuration file for editing. Go to the [server] stanza.
2. Decide if P3P headers from junctioned servers will be replaced or preserved. Set the following value:
   ```
   preserve-p3p-policy = {yes|no}
   ```
   The default value is no. Set this to yes if you want to preserve P3P headers. For more information, see “Junction header preservation” on page 60.
3. Go to the [p3p-header] stanza. Specify the access that the user will have to the information in the cookie. Set the value for the following entry:
   ```
   access = {none|all|nonident|contact-and-other|ident-contact|other-ident}
   ```
   The default setting is:
   ```
   access = none
   ```

Table 5. Supported values for the access entry

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No access to identified data is given.</td>
</tr>
<tr>
<td>all</td>
<td>Access is given to all identified data</td>
</tr>
<tr>
<td>nonident</td>
<td>Web site does not collect identified data.</td>
</tr>
</tbody>
</table>
Table 5. Supported values for the access entry (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contact-and-other</td>
<td>Access is given to identified online and physical contact information as well as to certain other identified data.</td>
</tr>
<tr>
<td>ident-contact</td>
<td>Access is given to identified online and physical contact information. For example, users can access things such as a postal address.</td>
</tr>
<tr>
<td>other-ident</td>
<td>Access is given to certain other identified data. For example, users can access things such as their online account charges.</td>
</tr>
</tbody>
</table>

4. Specify the type of information stored in the cookies or linked to by the cookies. Set the value for the following entry:

categories = {physical online|uniqueid|purchase|financial|computer|navigation|interactive|demographic|content|state|political|health|preference|location|government|other-category}

The default setting is:

categories = uniqueid

Table 6. Supported values for the categories entry

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical</td>
<td>Information that allows an individual to be contacted or located in the physical world. For example, telephone number or address.</td>
</tr>
<tr>
<td>online</td>
<td>Information that allows an individual to be contacted or located on the Internet.</td>
</tr>
<tr>
<td>uniqueid</td>
<td>Non-financial identifiers, excluding government-issued identifiers, issued for purposes of consistently identifying or recognizing the individual.</td>
</tr>
<tr>
<td>purchase</td>
<td>Information actively generated by the purchase of a product or service, including information about the method of payment.</td>
</tr>
<tr>
<td>financial</td>
<td>Information about an individual’s finances including account status and activity information such as account balance, payment or overdraft history, and information about an individual’s purchase or use of financial instruments including credit or debit card information.</td>
</tr>
<tr>
<td>computer</td>
<td>Information about the computer system that the individual is using to access the network. For example, IP number, domain name, browser type or operating system.</td>
</tr>
<tr>
<td>navigation</td>
<td>Data passively generated by browsing the Web site. For example, which pages are visited, and how long users stay on each page.</td>
</tr>
<tr>
<td>interactive</td>
<td>Data actively generated from or reflecting explicit interactions with a service provider through its site. For example, queries to a search engine, or logs of account activity.</td>
</tr>
<tr>
<td>demographic</td>
<td>Data about an individual’s characteristics. For example, gender, age, and income.</td>
</tr>
<tr>
<td>content</td>
<td>The words and expressions contained in the body of a communication. For example, the text of email, bulletin board postings, or chat room communications.</td>
</tr>
<tr>
<td>state</td>
<td>Mechanisms for maintaining a stateful session with a user or automatically recognizing users who have visited a particular site or accessed particular content previously. For example, HTTP cookies.</td>
</tr>
<tr>
<td>political</td>
<td>Membership in or affiliation with groups such as religious organizations, trade unions, professional associations and political parties.</td>
</tr>
</tbody>
</table>
Table 6. Supported values for the categories entry (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>health</td>
<td>Information about an individual’s physical or mental health, sexual orientation, use or inquiry into health care services or products, and purchase of health care services or products.</td>
</tr>
<tr>
<td>preference</td>
<td>Data about an individual’s likes and dislikes. For example, favorite color or musical tastes.</td>
</tr>
<tr>
<td>location</td>
<td>Information that can be used to identify an individual’s current physical location and track them as their location changes. For example, Global Positioning System position data.</td>
</tr>
<tr>
<td>government</td>
<td>Identifiers issued by a government for purposes of consistently identifying the individual.</td>
</tr>
<tr>
<td>other-category</td>
<td>Other types of data not captured by the above definitions.</td>
</tr>
</tbody>
</table>

5. Specify whether the full P3P policy contains some information regarding disputes over the information contained within the cookie. Set the value for the following entry:
   disputes = {yes|no}

   The dispute entry is not specified in the WebSEAL configuration file. The P3P specification states that when the dispute entry is not specified, the default value is automatically assigned no.

Table 7. Supported values for the dispute entry

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>The full P3P policy contains information regarding disputes over the information contained within the cookie.</td>
</tr>
<tr>
<td>no</td>
<td>The full P3P policy does not contain information regarding disputes over the information contained within the cookie.</td>
</tr>
</tbody>
</table>

6. Specify the types of remedies in case a policy breach occurs. Set the value for the following entry:
   remedies = {correct|money|law}

   The default setting is:
   remedies = correct

Table 8. Supported values for the remedies entry

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>Errors or wrongful actions arising in connection with the privacy policy will be remedied by the service.</td>
</tr>
<tr>
<td>money</td>
<td>If the service provider violates its privacy policy it will pay the individual an amount specified in the human readable privacy policy or the amount of damages.</td>
</tr>
<tr>
<td>law</td>
<td>Remedies for breaches of the policy statement will be determined based on the law referenced in the human readable description.</td>
</tr>
</tbody>
</table>

7. Specify either that no data is collected (including Web logs), or that the organization collecting the data will make anonymous any information that identifies the user. Set the value for the following entry:
   non-identifiable = {yes|no}

   The non-identifiable entry is not specified in the WebSEAL configuration file. The P3P specification states that when the non-identifiable entry is not
specified, the default value is automatically assigned no.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Data that is collected identifies the user.</td>
</tr>
<tr>
<td>no</td>
<td>No data is collected (including Web logs), or the information collected does not identify the user.</td>
</tr>
</tbody>
</table>

Table 9. Supported values for the non-identifiable entry

8. Specify the purpose of the information in the cookie. Set the value for the following entry:

```
purpose = {current|admin|develop|tailoring|pseudo-analysis|pseudo-decision|
            individual-analysis|individual-decision|contact|historical|
            telemarketing|other-purpose} [:[opt-in|opt-out|always]]
```

The default setting is:

```
purpose = current
```

Table 10. Supported values for the purpose entry

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>Information can be used by the service provider to complete the activity for which it was provided</td>
</tr>
<tr>
<td>admin</td>
<td>Information can be used for the technical support of the Web site and its computer system.</td>
</tr>
<tr>
<td>develop</td>
<td>Information can be used to enhance, evaluate, or otherwise review the site, service, product, or market.</td>
</tr>
<tr>
<td>tailoring</td>
<td>Information can be used to tailor or modify content or design of the site where the information is used only for a single visit to the site.</td>
</tr>
<tr>
<td>pseudo-analysis</td>
<td>Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals for purpose of research, analysis and reporting.</td>
</tr>
<tr>
<td>pseudo-decision</td>
<td>Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals to make a decision that directly affects that individual.</td>
</tr>
<tr>
<td>individual-analysis</td>
<td>Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data for the purpose of research, analysis and reporting.</td>
</tr>
<tr>
<td>individual-decision</td>
<td>Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data to make a decision that directly affects that individual.</td>
</tr>
<tr>
<td>contact</td>
<td>Information can be used to contact the individual, through a communications channel other than voice telephone, for the promotion of a product or service.</td>
</tr>
<tr>
<td>historical</td>
<td>Information can be archived or stored for the purpose of preserving social history as governed by an existing law or policy.</td>
</tr>
<tr>
<td>telemarketing</td>
<td>Information can be used to contact the individual via a voice telephone call for promotion of a product or service.</td>
</tr>
<tr>
<td>other-purpose</td>
<td>Information may be used in other ways not captured by the above definitions.</td>
</tr>
</tbody>
</table>
For each value specified for purpose, excepting the value current, you can optionally specify the opt-in policy. The syntax consists of a colon (:) immediately following the purpose value, followed by one of the supported values for the opt-in policy. For example:

```
purpose = telemarketing:opt-in
```

The following table lists the supported values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt-in</td>
<td>Data may be used for this purpose only when the user affirmatively requests this use.</td>
</tr>
<tr>
<td>opt-out</td>
<td>Data may be used for this purpose unless the user requests that it not be used in this way.</td>
</tr>
<tr>
<td>always</td>
<td>Users cannot opt-in or opt-out of this use of their data. This is the default value. When the opt-in policy is not specified, the always policy applies.</td>
</tr>
</tbody>
</table>

9. Specify the recipients of the information in the cookie. Set the value for the following entry:

```
recipient = {ours|delivery|same|unrelated|public|other-recipient}
[:[opt-in|opt-out|always]]
```

The default setting is:

```
recipient = ours
```

The following table lists the supported values for the recipient entry:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours</td>
<td>Ourselves and/or entities acting as our agents, or entities for whom we are acting as an agent. An agent is a third party that processes data only on behalf of the service provider.</td>
</tr>
<tr>
<td>delivery</td>
<td>Legal entities performing delivery services that may use data for purposes other than completion of the stated purpose.</td>
</tr>
<tr>
<td>same</td>
<td>Legal entities following our practices. These are legal entities who use the data on their own behalf under equitable practices.</td>
</tr>
<tr>
<td>unrelated</td>
<td>Unrelated third parties. These are legal entities whose data usage practices are not known by the original service provider.</td>
</tr>
<tr>
<td>public</td>
<td>Public forums. These are public forums such as bulletin boards, public directories, or commercial CD-ROM directories.</td>
</tr>
<tr>
<td>other-recipient</td>
<td>Legal entities following different practices. These are legal entities that are constrained by and accountable to the original service provider, but may use the data in a way not specified in the service provider’s practices.</td>
</tr>
</tbody>
</table>

For each value specified for recipient, excepting ours, you can optionally specify the opt-in policy. The syntax consists of a colon (:) immediately following the recipient, followed by one of the supported values for the opt-in policy. For example:

```
recipient = delivery:opt-in
```
The following table lists the supported values:

*Table 13. Opt-in policy values*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt-in</td>
<td>Data may be used for this purpose only when the user affirmatively requests this use.</td>
</tr>
<tr>
<td>opt-out</td>
<td>Data may be used for this purpose unless the user requests that it not be used in this way.</td>
</tr>
<tr>
<td>always</td>
<td>Users cannot opt-in or opt-out of this use of their data. This is the default value. When the opt-in policy is not specified, the always policy applies.</td>
</tr>
</tbody>
</table>

10. Specifies how long the information in the cookie is retained. Set the value for the following entry:

```
retention = {no-retention|stated-purpose|legal-requirement|business-practices|indefinitely}
```

The default setting is:

```
retention = no-retention
```

*Table 14. Supported values for the retention entry*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-retention</td>
<td>Information is not retained for more than a brief period of time necessary to make use of it during the course of a single online interaction.</td>
</tr>
<tr>
<td>stated-purpose</td>
<td>Information is retained to meet the stated purpose, and is to be discarded at the earliest time possible.</td>
</tr>
<tr>
<td>legal-requirement</td>
<td>Information is retained to meet a stated purpose, but the retention period is longer because of a legal requirement or liability.</td>
</tr>
<tr>
<td>business-practices</td>
<td>Information is retained under a service provider’s stated business practices.</td>
</tr>
<tr>
<td>indefinitely</td>
<td>Information is retained for an indeterminate period of time.</td>
</tr>
</tbody>
</table>

11. Optionally, specify a reference to a full XML compact policy file. Specify a value for the following entry:

```
<p3p-element = policyref=uri_to_default_location_of_full_policy>
```

This entry is present but commented out, and therefore not active, in the default WebSEAL configuration file. The default entry is the default location for the full policy on any web site.

```
# p3p-element = policyref="/w3c/p3p.xml"
```

When p3p-element is not set, browsers look by default for the full policy in /w3c/p3p.xml. Note that some browsers might not refer to p3p-element but proceed directly to /w3c/p3p.xml.

**Note:** Ensure that unauthenticated access is granted to /w3c/p3p.xml. See "Troubleshooting" on page 68.

**Specifying a custom P3P compact policy**

As an alternative to setting values for the entries in the WebSEAL configuration file, you can specify the exact contents of the P3P header. This can be useful, for
example, when your compact policy string has been generated by another utility, and you want to use that string for the P3P policy.

To specify a custom P3P compact policy, complete the following steps:

1. Remove the default values from the WebSEAL configuration file. For example, change the default WebSEAL entries to the following:
   
   ```
   [p3p-header]
   access =
   purpose =
   purpose =
   recipients =
   retention =
   categories =
   ```

2. Add your custom compact policy string to the p3p-element entry:
   
   ```
   p3p-element = CP="your_series_of_compact_policy_abbreviations"
   ```

   Any number of values can be added. The order of the values is not significant.

**Troubleshooting**

- **Problem:** Browser cannot access a the full P3P policy file.
- **Solution:** When the p3p-element is used to specify the location of a file containing the full policy, the browser attempts to access the file. The P3P specification does not require browsers to submit cookies with the request for the full policy. Internet Explorer 6 does not submit a session cookie when accessing the full policy. This means that access to the full policy must be granted to unauthenticated users. When the browser receives either a login form or a 401 error, modify the permissions on the full policy to allow access by unauthenticated users.
### Suppressing server identity

HTTP server responses normally contain the identity and version of the server:

```
content-type: text/html
date: Tue, 05 Mar 2002 02:34:18 GMT
content-length: 515
server: WebSEAL/3.9.0
last-modified: Thu, 21 Feb 2002 08:03:46 GMT
connection: close
```

For security reasons, you might want WebSEAL to suppress this information in its responses to clients. To suppress server identity in HTTP server responses, set the suppress-server-identity parameter in the [server] stanza of the WebSEAL configuration file to "yes":

```
[server]
suppress-server-identity = yes
```

The default setting is "no".

### Handling BASE HREF tags

You can configure how WebSEAL handles BASE HREF tags from filtered HTML documents. In the WebSEAL configuration file, use the preserve-base-href setting to specify the behavior:

```
[server]
preserve-base-href = no
```

When preserve-base-href is set to no, WebSEAL removes all BASE HREF tags from filtered HTML documents and prepends the base tag to filtered links. When preserve-base-href is set to yes, the BASE HREF tag is filtered.

### Enabling HTTP TRACE method

RFC 2616 for HTTP defines a TRACE method as follows: "This method is used to invoke a remote, application-layer loopback of the requested message. The recipient of the request is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request."

The TRACE method has been used by hackers to implement a security attack on Web servers. To provide optimal security, WebSEAL by default blocks the TRACE method for all requests to the WebSEAL server.

You can enable the TRACE method (disable blocking) by setting two entries in the WebSEAL configuration file.

To enable TRACE methods for local responses, set the following entry:

```
[server]
http-method-trace-enabled = yes
```

To enable TRACE methods for junctioned responses, set the following entry:

```
[server]
http-method-trace-enabled-remote = yes
```

The default WebSEAL configuration file does not set any value for these configuration file entries. The default behavior for WebSEAL, even when the configuration file entries are not specified, is to block all TRACE methods.
Chapter 3. WebSEAL server administration

This chapter contains information that describes administration tasks you can perform to manage the WebSEAL server.

Topic Index:

- “Server tasks” on page 72
- “Managing the Web space” on page 74
- “HTTP data compression” on page 80
- “HTTP error message pages” on page 84
- “Managing custom account management pages” on page 88
- “Backup and restore” on page 92
- “Problem determination tools for WebSEAL” on page 94
Server tasks

Start a WebSEAL server

UNIX:

```bash
pdweb start instance_name
```

**Note:** The command `pdweb` is a symbolic link with `pdweb_start`. You can use `pdweb_start` for all of the `pdweb` commands described in this chapter. This symbolic link to `pdweb_start` exists for backwards compatibility.

Examples:

- Start the initial WebSEAL server and all configured server instances:
  ```bash
  # /usr/bin/pdweb start
  ```

- Start a specific server instance only:
  ```bash
  # /usr/bin/pdweb start webseal3
  ```

Windows

Use the Services Control Panel:

Start > Settings > Control Panel > Administrative Tools > Services

Alternatively, open a command window and enter:

```bash
net start instance_name
```

To start the default WebSEAL server:

```bash
net start default
```

You must issue a separate `net start` command for each WebSEAL instance to be started.

Stop a WebSEAL server

UNIX:

```bash
pdweb stop instance_name
```

Examples:

- Stop the initial WebSEAL server and all configured server instances:
  ```bash
  # /usr/bin/pdweb stop
  ```

- Stop a specific server instance only:
  ```bash
  # /usr/bin/pdweb stop webseal3
  ```

Windows

Use the Services Control Panel:

Start > Settings > Control Panel > Administrative Tools > Services

Alternatively, open a command window and enter:

```bash
net stop instance_name
```

To stop the default WebSEAL server:

```bash
net stop default
```
You must issue a separate `net stop` command for each WebSEAL instance to be started.

**Restart a WebSEAL server**

This command stops and restarts the specified WebSEAL server

**UNIX and Linux**

```
pdweb restart instance_name
```

For example, to restart the initial WebSEAL server and all configured server instances:

```
#/usr/bin/pdweb restart
```

**Display WebSEAL server status**

**UNIX:**

```
pdweb status
```

Show status of all configured servers:

```
#/usr/bin/pdweb status
```

<table>
<thead>
<tr>
<th>Access Manager Servers</th>
<th>Enabled</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-webseald</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>webseald-webseal2</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>webseald-webseal3</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Windows:**

Identify the WebSEAL server process in the Services Control Panel and use the appropriate control buttons.
Managing the Web space

The following sections describe tasks required to manage the Web space:

- “Root directory of the Web document tree” on page 74
- “Configuring directory indexing” on page 75
- “Windows: File naming for CGI programs” on page 76
- “Configuring Web document caching” on page 77
- “Specifying document MIME types for URL filtering” on page 79

WebSEAL represented in the protected object space

The server-name parameter in the WebSEAL configuration file specifies the point in the Tivoli Access Manager protected object space that represents this WebSEAL server instance.

For a single WebSEAL server installation, this value is automatically set using the host name of the machine where this WebSEAL server is being installed.

For example, if the machine (host) name is sales1, the parameter value is set as:

```
[server]
server-name = sales1
```

The representation of this WebSEAL server instance in the Tivoli Access Manager protected object space would appear as:
```
/WebSEAL/sales1
```

Root directory of the Web document tree

The Web document tree location is the absolute path to the root of the document tree for document resources made available by WebSEAL. This path name is represented by the doc-root parameter in the [content] stanza of the WebSEAL configuration file. The default location is initially established during the installation of WebSEAL:

**UNIX:**

```
[content]
doc-root = /opt/pdweb/www-instance_name/docs
```

**Windows:**

```
[content]
doc-root = C:\Program Files\Tivoli\PDWeb\www-instance_name\docs
```

This value is used only once—the first time WebSEAL is started after installation. The value is then stored in the junction database. Future modification of this value has no impact.

**How to change the document root after installation:**

After installation, you must use the pdadmin utility to change the document root directory location value. The following example (machine, or host, name is websealA) illustrates this procedure:

1. Login to pdadmin:
2. Use the `server task list` command to display all current junction points:

   pdadmin> server task webseald-websealA list

3. Use the `server task show` command to display details of the junction:

   pdadmin> server task webseald-websealA show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /opt/pdweb/www/docs

4. Create a new local junction to replace the current junction point (the `-f` option is required to force a new junction that overwrites an existing junction):

   pdadmin> server task webseald-websealA create -t local -f -d /tmp/docs /
   Created junction at /

5. List the new junction point:

   pdadmin> server task webseald-websealA list /

6. Display the details of this junction:

   pdadmin> server task webseald-websealA show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /tmp/docs

### Configuring directory indexing

You can specify the name of the default file returned by WebSEAL when the URL expression of a request ends with a directory name. If this default file exists, WebSEAL returns the file to the client. If the file does not exist, WebSEAL dynamically generates a directory index and returns the list to the client.

The parameter for configuring the directory index file is located in the `[content]` stanza of the WebSEAL configuration file.

The default value for the index file is:

```
[content]
directory-index = index.html
```

You can change this filename if your site uses a different convention. For example:

```
[content]
directory-index = homepage.html
```

WebSEAL dynamically generates a directory index if the directory in the request does not contain the index file defined by the `directory-index` parameter. The generated index contains a list of the directory contents, with links to each entry in the directory. The index is generated only if the client requesting access to the directory has the "list" (l) permission on the ACL for that directory.
You can configure the specific graphical icons used by WebSEAL for each file type listed in a generated index. The [content-index-icons] stanza of the WebSEAL configuration file contains a list of the document MIME types and the associated .gif files that are displayed:

[content-index-icons]
image/* = /icons/image2.gif
video/* = /icons/movie.gif
audio/* = /icons/sound2.gif
text/html = /icons/generic.gif
text/* = /icons/text.gif
application/x-tar = /icons/tar.gif
application/* = /icons/binary.gif

You can configure this list to specify other icons for each MIME type. Icons can also be located remotely. For example:

application/* = http://www.acme.com/icons/binary.gif

You can also configure these additional icon values:

- Icon used to represent sub-directories:
  
  [icons]
  diricon = /icons/folder2.gif

- Icon used to represent the parent directory:
  
  [icons]
  backicon = /icons/back.gif

- Icon used to represent unknown file types:
  
  [icons]
  unknownicon = /icons/unknown.gif

**Note:** The supplied icons are in GIF format, but this format is not required.

**Windows: File naming for CGI programs**

Parameters contained in the [cgi-types] stanza of the WebSEAL configuration file allow you to specify the Windows file extension types that are recognized and executed as CGI programs.

The UNIX operating system has no file name extension requirements. File extension types must be defined, however, for Windows operating systems. The [cgi-types] stanza lists all valid extension types and maps each extension (when necessary) to an appropriate CGI program.

[cgi-types]
<extension> = <cgi-program:>

By default only those files with extensions matching those listed in the stanza will be executed as CGI programs. If a CGI program has an extension that is not contained in this list, the program will not be executed.

Files with the .exe extensions are run as programs by Windows default and require no mapping.

**Note:** Anytime you want to install a .exe file on Windows for download, however, you must rename the extension or install the file as part of an archive (such as .zip).

You must supply the appropriate interpreter programs for extensions that represent interpreted script files. Examples of these types of extensions include: shell scripts (.sh and .ksh), Perl scripts (.pl), and Tcl scripts (.tcl) files.
The following example illustrates a typical [cgi-types] stanza configuration:

[cgi-types]
bat = cmd
    cmd = cmd
    pl = perl
    sh = sh
tcl = tclsh76

Executable UNIX files on the WebSEAL server host system
WebSEAL supports the creation of local junctions. These junctions exist on the same host system as the WebSEAL server.

When accessing a local junction on a UNIX system, WebSEAL interprets files as CGI scripts if the files are given the UNIX execute permission. For example, if an HTML page located on the local WebSEAL junction is given execute permission, WebSEAL interprets the file as an executable, and will not display it.

To ensure that local files are displayed correctly, remove execute permission from all non-CGI files that are accessed through the local junction.

Configuring Web document caching
Clients can often experience extended network access time and file downloading time due to poor Web document retrieval performance. Poor performance can occur because the WebSEAL server is waiting for documents retrieved from junctioned back-end servers or even slow local storage.

Web document caching gives you the flexibility of serving documents locally from WebSEAL rather than from a back-end server across a junction. The Web document caching feature allows you to store commonly accessed Web document types in the WebSEAL server’s memory. Clients can experience much faster response to follow-up requests for documents that have been cached in the WebSEAL server.

Cached documents can include static text documents and graphic images. Dynamically generated documents, such as database query results, cannot be cached.

Caching is performed on the basis of MIME type. When you configure WebSEAL for Web document caching, you identify the following three parameters:
- Document MIME type
- Type of storage medium
- Size of storage medium

You define Web document caching in the [content-cache] stanza of the WebSEAL configuration file. The following syntax applies:

<mime-type> = <cache-type>:<cache-size>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mime-type</td>
<td>Represents any valid MIME type conveyed in an HTTP &quot;Content-Type:&quot; response header. This value may contain a asterisk (*). A value of <em>/</em> represents a default object cache that will hold any object that does not correspond to an explicitly configured cache. Note that the asterisk here is a wildcard only for a MIME-type directory, and its contents. This is not a wildcard for regular expression matching.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache-type</td>
<td>Specifies the type of storage medium to use for the cache. This release of Tivoli Access Manager supports only &quot;memory&quot; caches.</td>
</tr>
<tr>
<td>cache-size</td>
<td>Specifies the maximum size (in kilobytes) to which the given cache can grow before objects are removed according to a &quot;Least Recently Used&quot; algorithm.</td>
</tr>
</tbody>
</table>

#### Example:

- `text/html = memory:2000`
- `image/* = memory:5000`
- `/*/* = memory:1000`

### Conditions affecting Web document caching

The Web document caching mechanism observes the following conditions:

- Caching occurs only when a cache is defined in the WebSEAL configuration file.
- By default, no caches are defined at installation.
- If you do not specify a default cache, documents that do not match any explicit cache are not cached.
- Authorization is still performed on all requests for cached information.
- The caching mechanism does not cache responses to requests containing query strings.
- The caching mechanism does not cache responses to requests over junctions configured with the `–c` and `–C` options.

### Flushing all caches

You can use the `pdadmin` utility to flush all configured caches. The utility does not allow you to flush individual caches.

You must login to the secure domain as the Tivoli Access Manager administrator `sec_master` before you can use `pdadmin`.

To flush all Web document caches, enter the following command:

**UNIX:**

```bash
# pdadmin server task webseald-<machine-name> cache flush all
```

**Windows:**

```cmd
MSDOS> pdadmin server task webseald-<machine-name> cache flush all
```

### Controlling caching for specific documents

You can control caching for specific documents by attaching a special Protected Object Policy (POP) to those objects. This POP must contain an extended attribute called `document-cache-control`.

The `document-cache-control` extended attribute recognizes the following two values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-cache</td>
<td>The <code>no-cache</code> value instructs WebSEAL not to cache this document. Remember that all children of the object with the POP also inherit the POP conditions.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>public</td>
<td>The <em>public</em> value allows WebSEAL to cache the document by ignoring the fact that the junction was created with a <code>-c</code> or <code>-C</code> option. In addition, this value also allows caching of this document when the request is sent with an authorization header (such as Basic Authentication). This condition also includes a request where WebSEAL inserts BA information on behalf of the client (such as with GSO or <code>-b supply</code> junctions). Normally, proxy servers do not cache the response documents to requests that include authorization headers.</td>
</tr>
</tbody>
</table>

Use the *pdadmin pop create*, *pdadmin pop modify*, and *pdadmin pop attach* commands. The following example illustrates creating a POP called "doc-cache" with the *document-cache-control* extended attribute and attaching it to an object (budget.html):

```bash
pdadmin> pop create doc-cache
pdadmin> pop modify doc-cache set attribute document-cache-control no-cache
pdadmin> pop attach /WebSEAL/hostA/junction/budget.html doc-cache
```

The `budget.html` document is never cached by WebSEAL. Each request for this document must be made directly to the back-end server where it is located.

Details about the *pdadmin* command line utility can be found in the *IBM Tivoli Access Manager Base Administrator’s Guide*.

**Specifying document MIME types for URL filtering**

To ensure proper performance of links across WebSEAL junctions, WebSEAL can apply specific URL filtering rules to document responses from back-end junctioned servers. You must first specify the document MIME types that WebSEAL can recognize.

The `type` parameter in the *[filter-content-types]* stanza of the WebSEAL configuration file specifies a MIME type value. WebSEAL is configured by default to recognize documents of two MIME types:

```ini
[filter-content-types]
type = text/html
        type = text/vnd.wap.wml
```

WebSEAL can apply the following URL filtering functionality to all configured document types:

- **URL scheme filtering**
  WebSEAL filters only URLs using schemes defined in the *[filter-schemes]* stanza of the WebSEAL configuration file.
- **URL attribute filtering**
- **Script filtering for absolute URLs**
  See “Modifying absolute URLs with script filtering” on page 290.
HTTP data compression

WebSEAL servers can be configured to compress data that is transferred over HTTP between the WebSEAL server and the client. WebSEAL uses the gzip compression algorithm described in RFC 1952. Gzip is supported by all major browsers.

HTTP compression in WebSEAL can be configured based on MIME-type, browser type, and protected object policies (POPs).

See the following sections:
- “Compression based on MIME-type”
- “Compression based on user agent type” on page 81
- “Compression policy in POPs” on page 82
- “Data compression limitation” on page 82
- “Data compression configuration” on page 82

Compression based on MIME-type

You can create an entry in the WebSEAL configuration file for each MIME-type or group of MIME-types for which data compression is needed. The syntax is:

```
[compress-mime-types]
mime_type = minimum_doc_size[:compression_level]
```

The `mime_type` can specify one particular MIME type or can use wildcard characters to specify a class of MIME types. Each mime-type declaration is a separate entry in the `[compress-mime-types]` stanza. The wildcard character (*) is limited to entries of one collection of MIME types. For example, `text/*`. Any MIME-type not listed in the stanza is not compressed. Order is important. The first entry that matches a returned document is used for that document.

The `minimum_doc_size` value specifies policy regarding what size of documents will be compressed. This value is an integer. Valid values are shown in the following list:
- -1
  When the minimum size is -1, documents of the specified MIME-type are never compressed
- 0
  When the minimum size is 0, documents of the specified MIME-type are always compressed
- Integer greater than zero
  When the minimum size is greater than zero, documents of the specified MIME-type are compressed when the number of bytes in the response to WebSEAL exceeds this integer value

Any negative number other than -1 generates an error message.

When WebSEAL receives a request from a browser, the server examines the content-length field in the HTTP header to determine the size of the incoming data. However, not all HTTP responses contain the content-length field. When the content-length field is not found, WebSEAL compresses the document unless the applicable MIME-type has been configured to never be compressed (`minimum_doc_size` of -1).
The *compression_level* is an optional setting that specifies a data compression level. Valid values are integers between 1 and 9. The larger the integer, the greater the amount of compression that takes place. Note that the greater the amount of compression, the greater the load placed on the CPU. The value of increased compression must be weighed against any performance impacts. When the *compression_level* is not specified, a default level of 1 is used.

The following example compresses all documents of size greater than 1000 bytes:
```
[compress-mime-type]
*/* = 1000
```

The following set of entries disables compression for all images, disables compression for CSS files, enables compression at level 5 for all PDF documents, enables compression for HTML documents of size greater than 2000 bytes, and enables compression for all other text documents, regardless of size:
```
[compress-mime-type]
image/* = -1
text/css = -1
application/pdf = 0:5
text/html = 2000
text/* = 0
```

**Compression based on user agent type**

WebSEAL returns compressed data to user agents that request compressed data. WebSEAL does not return compressed data to user agents that do not request it. However, some user agents request compressed data but do not know how to handle the data properly. WebSEAL administrators can use the WebSEAL configuration file to explicitly enable or disable compression for various browsers.

The configuration file entry syntax is:
```
[compress-user-agents]
user_agent_pattern = {yes|no}
```

The *user_agent_pattern* consists of wildcard patterns that match characters found in the user-agent header sent to WebSEAL. The value *yes* means to compress data that is returned to the browser. The value *no* means to return the data uncompressed.

When the user-agent header does not match any of the stanza entries in the WebSEAL configuration file, WebSEAL honors the accept-encoding header sent by the browser.

For example, the following entry enables compression for Internet Explorer 6, but disables compression for all other browsers:
```
[compress-user-agents]
*MSIE 6.0" = yes
* = no
```

The following example disables compression for Netscape 4 but allows other browsers to receive compressed data. Note that the entry *MSIE* is necessary because the user-agent header sent by Internet Explorer begins with Mozilla:
```
[compress-user-agents]
*MSIE* = yes
Mozilla/4.* = no
```
Compression policy in POPs

You can specify a compression policy of do not compress in a protected object policy (POP). You can attach the POP to an object, and WebSEAL disables compression of that object. To specify this policy, add the following attribute to the POP:

document-compression = no

A POP without this attribute set, or with this attribute set to any value other than no, allows documents to be compressed.

For example, to disable compression for the junction /appOne:

```bash
pdadmin> pop create appOnePop
pdadmin> pop modify appOnePop set attribute document-compression no
pdadmin> pop attach /WebSEAL/host/appOne appOnePop
```

To allow compression for a subdirectory beneath /appOne with an overriding POP, attach a different POP that does not have the document-compression attribute. For example:

```bash
pdadmin> pop create dataPop
pdadmin> pop attach /WebSEAL/host/appOne/data dataPop
```

This method of applying compression policy can be used with URLs. For example, to disable compression based on wild card patterns applied to URLs, you can use dynurl. To disable compression for all requests to a junction that have a particular argument in the query string, you can create a dynurl.conf file with entries like the following:

```bash
/disableCompression /appOne/*?want-response=text/xml
```

You can then attach a pop to /WebSEAL/host/disableCompression with the document-compression attribute set to no.

Data compression limitation

- The transfer of compressed data between WebSEAL and backend servers is not supported.
  
  WebSEAL performs filtering on URLs for various purposes. WebSEAL does not perform filtering on compressed data.

Data compression configuration

To specify data compression policy for communication between WebSEAL and client browsers, complete the following steps:

1. In the WebSEAL configuration file, specify each MIME-type for which a data compression policy applies. Assign a value that enforces the policy.

   ```bash
   [compress-mime-types]
   mime_type = minimum_doc_size
   ```

   Note that the default setting leaves all data uncompressed:

   ```bash
   [compress-mime-types]
   */* = -1
   ```

   For more information on creating entries in this stanza, see "Compression based on MIME-type" on page 80.

2. In the WebSEAL configuration file, specify each type of user agent (browser) for which a data compression policy applies. Enable data compression by assigning the value yes. Disable data compression by assigning the value no.
[compress-user-agent]
user_agent = {yes|no}

No entries are set by default. When no entry matches the user-agent’s accept-encoding header, the value in the accept-encoding header is honored.

For more information on creating entries in this stanza, see "Compression based on user agent type" on page 81.

3. Optionally, specify compression policies in POPs, and apply the POPs to the appropriate objects in the protected object space.
For more information, see "Compression policy in POPs" on page 82.
HTTP error message pages

When WebSEAL is unable to process an access request from a client browser, WebSEAL returns an HTML error message page to the client. The error message page explains why the access request failed.

WebSEAL provides a number of default message pages. You can modify the contents of these pages. You can also create your own error message pages, based on error codes returned by WebSEAL.

This section contains the following topics:

- "Default error message pages"
- "Modifying existing HTTP error pages" on page 86
- "Creating new HTTP error pages" on page 86
- "Enabling the time of day error page" on page 86
- "Specifying error page location" on page 87
- "Backwards compatibility" on page 87

Default error message pages

The error message pages are installed when the WebSEAL server instance is configured. Each error message page is a separate HTML file. The names of the files are the hexadecimal values of the returned error codes. Do not modify these file names.

The location of the error files is configurable. See “Specifying error page location” on page 87. The names of the error message files are listed in the following table.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Title</th>
<th>Description</th>
<th>HTTP Error Code</th>
</tr>
</thead>
</table>
| 132120c8.html  | Authentication Failed                | Credentials could not be retrieved for the client certificate used. Possible reasons include:  
|                |                                      | * the user supplied an incorrect certificate  
|                |                                      | * the certificate has been revoked  
<p>|                |                                      | * the user's credentials are missing from the authentication database       |                 |
| 38ad52fa.html  | Non-Empty Directory                  | The requested operation requires the removal of a non-empty directory. This is an illegal operation. |                 |
| 38cf013d.html  | Request Caching Failed               | The request-max-cache or request-body-max-read values have been exceeded.   |                 |
| 38cf0259.html  | Could Not Sign User On               | The resource requested requires the WebSEAL server to sign user on to another Web server. However, a problem occurred while WebSEAL was attempting to retrieve the information. |                 |
| 38cf025a.html  | User Has No Single Sign-on Information | WebSEAL could not locate the GSO user for the requested resource.           |                 |
| 38cf025b.html  | No Single Sign-on Target for User     | WebSEAL could not locate the GSO target for the requested resource.         |                 |
| 38cf025c.html  | Multiple Sign-on Targets for User     | Multiple GSO targets are defined for the requested resource. This is a mis-configuration. |                 |</p>
<table>
<thead>
<tr>
<th>Filename</th>
<th>Title</th>
<th>Description</th>
<th>HTTP Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>38cf025d.html</td>
<td>Login Required</td>
<td>The resource requested is protected by a junctioned back-end Web server, requiring WebSEAL to sign user on to that Web server. In order to do this, user must first log into WebSEAL.</td>
<td></td>
</tr>
<tr>
<td>38cf025e.html</td>
<td>Could Not Sign User On</td>
<td>The resource requested requires WebSEAL to sign user on to another Web server. However, the sign-on information for the user account is incorrect.</td>
<td></td>
</tr>
<tr>
<td>38cf025f.html</td>
<td>Unexpected Authentication Challenge</td>
<td>WebSEAL received an unexpected authentication challenge from a junctioned back-end Web server.</td>
<td></td>
</tr>
<tr>
<td>38cf0421.html</td>
<td>Moved Temporarily</td>
<td>The requested resource has been temporarily moved. This usually occurs if there has been a mishandled redirect.</td>
<td>302</td>
</tr>
<tr>
<td>38cf0424.html</td>
<td>Bad Request</td>
<td>WebSEAL received an invalid HTTP request.</td>
<td>400</td>
</tr>
<tr>
<td>38cf0425.html</td>
<td>Login Required</td>
<td>The resource you have requested is secured by WebSEAL, and in order to access it, you must first login.</td>
<td>403</td>
</tr>
<tr>
<td>38cf0427.html</td>
<td>Forbidden</td>
<td>User does not have permissions to access requested resource.</td>
<td>403</td>
</tr>
<tr>
<td>38cf0428.html</td>
<td>Not Found</td>
<td>Requested resource cannot be located.</td>
<td>404</td>
</tr>
<tr>
<td>38cf0432.html</td>
<td>Service Unavailable</td>
<td>A service required by WebSEAL to complete request is currently not available.</td>
<td>503</td>
</tr>
<tr>
<td>38cf0434.html</td>
<td>Privacy required</td>
<td>Quality of protection privacy level required.</td>
<td></td>
</tr>
<tr>
<td>38cf0437.html</td>
<td>Server Suspended</td>
<td>The WebSEAL server has been temporarily suspended by the System Administrator. No requests will be handled until the server is returned to service by the Administrator.</td>
<td></td>
</tr>
<tr>
<td>38cf0439.html</td>
<td>Session Information Lost</td>
<td>The browser/server interaction was a stateful session with a junctioned back-end server that is no longer responding. WebSEAL requires a service located on this server to complete your request.</td>
<td></td>
</tr>
<tr>
<td>38cf0442.html</td>
<td>Service Unavailable</td>
<td>The service required by WebSEAL is located on a junctioned back-end server where SSL mutual authentication has failed.</td>
<td></td>
</tr>
<tr>
<td>38cf07aa.html</td>
<td>CGI Program Failed</td>
<td>A CGI program failed to execute properly.</td>
<td></td>
</tr>
<tr>
<td>38cf08cc.html</td>
<td>Access Denied</td>
<td>The resource you have requested is protected by a policy restricting access to specific time periods. The current time is outside of those permitted time periods.</td>
<td></td>
</tr>
<tr>
<td>38cf04d7.html</td>
<td>Third-party server not responding</td>
<td>The requested resource is located on a third-party server. WebSEAL has tried to contact that server, but it is not responding.</td>
<td></td>
</tr>
<tr>
<td>38cf04c6.html</td>
<td>Third-party server not responding</td>
<td>The requested resource is located on a third-party server. WebSEAL has tried to contact that server, but it is not responding.</td>
<td></td>
</tr>
<tr>
<td>default.html</td>
<td>Server Error</td>
<td>WebSEAL could not complete your request due to an unexpected error.</td>
<td>500</td>
</tr>
<tr>
<td>deletesuccess.html</td>
<td>Success</td>
<td>Client-initiated DELETE request completed successfully.</td>
<td>200</td>
</tr>
<tr>
<td>Filename</td>
<td>Title</td>
<td>Description</td>
<td>HTTP Error Code</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>putsuccess.html</td>
<td>Success</td>
<td>Client-initiated PUT operation completed successfully.</td>
<td>200</td>
</tr>
<tr>
<td>relocated.html</td>
<td>Temporarily Moved</td>
<td>The requested resource has temporarily moved.</td>
<td>302</td>
</tr>
<tr>
<td>websealerror.html</td>
<td>400 WebSEAL Server Error</td>
<td>WebSEAL server internal error.</td>
<td>400</td>
</tr>
</tbody>
</table>

**Modifying existing HTTP error pages**

You can customize the error message pages to reflect the current deployment. Observe the following notes:

- Do not modify the name of the file. The hexadecimal number is used by WebSEAL to display the proper error file.
- Use an HTML or text editor to modify page contents. Ensure that you use valid HTML tagging.
- WebSEAL supplies a set of macros that you can use to capture dynamic information. See “Macro support for account management pages” on page 89.

**Creating new HTTP error pages**

You can create new error message pages for hexadecimal errors returned by WebSEAL. The hexadecimal errors returned by WebSEAL are documented in the *IBM Tivoli Access Manager Error Message Reference*.

For example, when WebSEAL encounters an invalid HTTP header, it returns the following error:

wand_s_jct_invalid_http_header 0x38cf04d5

To create a new error message page for this error, complete the following steps:

1. Create a new HTML file. To name the file, delete the 0x (hex) prefix characters from the error number and supply the .html suffix. For example, 0x38cf04d5 becomes:

38cf04d5.html

   Optionally, you can use one of the existing HTTP error files as a template. Copy it and rename it.

2. Consult the *IBM Tivoli Access Manager Error Message Reference* for information on the exact error encountered. Use this information to edit the body of the HTML page.

3. Optionally, you can use the macros described in “Modifying existing HTTP error pages.”

4. Save the new file in the same directory as the rest of the HTTP error messages.

**Enabling the time of day error page**

Activation of one error message is controlled by a setting in the WebSEAL configuration file. To enable WebSEAL to display 38cf08cc.html, you must set the following entry:

```
[acnt-mgt]
client-notify-tod = yes
```
When client-notify-tod = yes, WebSEAL sends the client an error message stating that the authorization failure was due to a failed time-of-day # POP access check.

This entry is set to no by default.

**Specifying error page location**

To specify the directory location for the error pages, set the error-dir entry in the [content] stanza:

```
[content]
error-dir = lib/errors
```

The value lib/errors is the default directory. You can modify this value.

This location is relative to the directory specified by the server-root entry in the [server] stanza.

In addition, the directory specific to your locale is automatically appended to the end of the directory hierarchy.

For example, given a system with the instance name webseal1, and the following configuration settings:

- server-root = /opt/pdweb/www-webseal1
- error-dir = lib/errors
- English locale directory of C

the location of the error pages would be:

```
/opt/pdweb/www-webseal1/lib/errors/C
```

**Backwards compatibility**

- WebSEAL Version 5.1 introduced the following new error pages:
  - 38cf04d7.html
  - 38cf04c6.html

These messages provide information indicating that the encountered failure originated with a backend server, not with WebSEAL. In past releases, WebSEAL returned the default error page only. If you want to retain the previous behavior, remove the new HTTP error message pages from the error message page directory.

- WebSEAL Version 5.1 introduced a new HTTP error message for use when access is denied because a protected object policy (POP) time of day policy was not satisfied. Use of this error message page is controlled by a WebSEAL configuration file setting. To disable use of this page, change the following configuration setting:

  ```
  [acnt-mgmt]
  client-notify-tod = no
  ```

  **Note:** A 403 is always logged, regardless of the value assigned to client-notify-tod.
Managing custom account management pages

Tivoli Access Manager includes sample account management HTML pages that can be customized to contain site-specific messages or perform site-specific actions. Most pages are appropriate for Forms, token, and BA authentication over HTTP or HTTPS.

The file locations for these pages are defined by the `mgt-pages-root` parameter in the `[acnt-mgt]` stanza of the WebSEAL configuration file.

```
mgt-pages-root = lib/html/<lang-dir>
```

The actual directory used is based on localization. The default United States English directory is:

```
lib/html/EN
```

The Japanese locale files are in:

```
lib/html/JP
```

See “Multi-locale messages” on page 53 for further information on multi-locale support.

Custom page parameters and values

The following special HTML page parameters and values are located in the `[acnt-mgt]` stanza of the WebSEAL configuration file. Some pages are used only by the Forms login method of providing identity information.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>login.html</td>
<td>Forms login</td>
</tr>
<tr>
<td>login-success</td>
<td>login_success.html</td>
<td>Forms login</td>
</tr>
<tr>
<td>logout</td>
<td>logout.html</td>
<td>Forms login</td>
</tr>
<tr>
<td>account-locked</td>
<td>acct_locked.html</td>
<td>Any method</td>
</tr>
<tr>
<td>passwd-expired</td>
<td>passwd_exp.html</td>
<td>Any method</td>
</tr>
<tr>
<td>passwd-change</td>
<td>passwd.html</td>
<td>Any method</td>
</tr>
<tr>
<td>passwd-change-success</td>
<td>passwd_rep.html</td>
<td>Any method</td>
</tr>
<tr>
<td>passwd-change-failure</td>
<td>passwd.html</td>
<td>Any method</td>
</tr>
<tr>
<td>help</td>
<td>help.html</td>
<td>Any method</td>
</tr>
<tr>
<td>token-login</td>
<td>tokenlogin.html</td>
<td>Token login</td>
</tr>
<tr>
<td>next-token</td>
<td>nexttoken.html</td>
<td>Token login</td>
</tr>
<tr>
<td>certificate-login</td>
<td>certlogin.html</td>
<td>Certificate login</td>
</tr>
<tr>
<td>cert-stepup-http</td>
<td>certstepuphttp.html</td>
<td>Certificate login</td>
</tr>
<tr>
<td>stepup-login</td>
<td>stepuplogin.html</td>
<td>Step-up authentication</td>
</tr>
<tr>
<td>switch-user</td>
<td>switchuser.html</td>
<td>Any method</td>
</tr>
<tr>
<td>cert-failure</td>
<td>certfailure.html</td>
<td>Certificate login</td>
</tr>
</tbody>
</table>
Custom HTML page descriptions

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acct_locked.html</td>
<td>Page displayed if user authentication failed due to a locked account.</td>
</tr>
<tr>
<td>certfailure.html</td>
<td>Page displayed if client fails to authenticate with a certificate when accept-client-certs = required.</td>
</tr>
<tr>
<td>certlogin.html</td>
<td>Certificate login form used when accept-client-certs = prompt_as_needed.</td>
</tr>
<tr>
<td>certstepup.html</td>
<td>Error message indicating that step-up to certificates was attempted over HTTP. The use of HTTPS is required.</td>
</tr>
<tr>
<td>help.html</td>
<td>Page containing links to valid administration pages.</td>
</tr>
<tr>
<td>login.html</td>
<td>Standard request form for username and password</td>
</tr>
<tr>
<td>login_success.html</td>
<td>Page displayed after successful login.</td>
</tr>
<tr>
<td>logout.html</td>
<td>Page displayed after successful logout.</td>
</tr>
<tr>
<td>nexttoken.html</td>
<td>Next token form.</td>
</tr>
<tr>
<td>passwd_exp.html</td>
<td>Page displayed if user authentication failed due to an expired password.</td>
</tr>
<tr>
<td>passwd.html</td>
<td>Change password form. Also displayed if password change request failed.</td>
</tr>
<tr>
<td>passwd_rep.html</td>
<td>Page displayed if password change request was successful.</td>
</tr>
<tr>
<td>stepuplogin.html</td>
<td>Step-up authentication login form.</td>
</tr>
<tr>
<td>switchuser.html</td>
<td>Switch user management form.</td>
</tr>
<tr>
<td>tokenlogin.html</td>
<td>Token login form.</td>
</tr>
</tbody>
</table>

Macro support for account management pages

The following macros are available for use in customizing the HTML pages listed in the previous section and the HTML error message pages listed in “Default error message pages” on page 84. These macro strings can be placed in the template files. Macros dynamically substitute appropriate information that is available. Macros are populated in each default page only when the value is relevant to that page.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%AUTHNLEVEL%</td>
<td>Authentication level. Used in authentication strength policy (stepup).</td>
</tr>
<tr>
<td>%BACK_NAME%</td>
<td>The value &quot;BACK&quot; if a referer header is present in the request, or &quot;HOME&quot; if none.</td>
</tr>
<tr>
<td>%BACK_URL%</td>
<td>The value of the referer header from the request, or &quot;/&quot; if none.</td>
</tr>
<tr>
<td>%BASICAUTHN%</td>
<td>Substitution variable used to alter login form. When authentication method valid, WebSEAL removes the variable. When authentication method invalid, WebSEAL comments out the macro. Used in tokenlogin.html, certlogin.html, and stepuplogin.html.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>%CERTAUTHN%</td>
<td>Substitution variable used to alter login form. When authentication method valid, WebSEAL removes the variable. When authentication method invalid, WebSEAL comments out the macro. Used in tokenlogin.html, certlogin.html, and stepuplogin.html.</td>
</tr>
<tr>
<td>%ERROR%</td>
<td>The hard-coded error message returned from Tivoli Access Manager. Same as %ERROR_TEXT%. Both exist for backwards compatibility.</td>
</tr>
<tr>
<td>%ERROR_CODE%</td>
<td>The numeric value of the error code.</td>
</tr>
<tr>
<td>%ERROR_TEXT%</td>
<td>The text associated with an error code in the message catalog. Same as %ERROR%. Both exist for backwards compatibility.</td>
</tr>
<tr>
<td>%FAILREASON%</td>
<td>Error message.</td>
</tr>
<tr>
<td>%HOSTNAME%</td>
<td>Fully qualified host name.</td>
</tr>
<tr>
<td>%HTTP_BASE%</td>
<td>Base HTTP URL of the server &quot;<a href="http://host:tcpport/">http://host:tcpport/</a>&quot;.</td>
</tr>
<tr>
<td>%HTTPS_BASE%</td>
<td>Base HTTPS URL of the server, &quot;<a href="https://host:sslport/">https://host:sslport/</a>&quot;.</td>
</tr>
<tr>
<td>%LOCATION%</td>
<td>Contains the URL to which the client is being redirected. Sent only in redirects.</td>
</tr>
<tr>
<td>%METHOD%</td>
<td>The HTTP method requested by the client.</td>
</tr>
<tr>
<td>%PROTOCOL%</td>
<td>The client connection protocol used. Can be http or https.</td>
</tr>
<tr>
<td>%REFERER%</td>
<td>The value of the referer header from the request, or &quot;Unknown&quot;, if none.</td>
</tr>
<tr>
<td>%REFERER_ENCODED%</td>
<td>A URI encoded version of the referer header and macro.</td>
</tr>
<tr>
<td>%STEPUP%</td>
<td>A message specifying the step-up level required. Only sent when returning a step-up login form</td>
</tr>
<tr>
<td>%TOKENAUTHN%</td>
<td>Substitution variable used to alter login form. When authentication method valid, WebSEAL removes the variable. When authentication method invalid, WebSEAL comments out the macro. Used in tokenlogin.html, certlogin.html, and stepuplogin.html.</td>
</tr>
<tr>
<td>%URL%</td>
<td>The URL requested by the client.</td>
</tr>
<tr>
<td>%URL_ENCODED%</td>
<td>A URI encoded version of the URI and macro</td>
</tr>
<tr>
<td>%USERNAME%</td>
<td>The name of the logged in user.</td>
</tr>
</tbody>
</table>

WebSEAL provides a configuration file that specifies how the data string specified by macro strings in inserted into WebSEAL HTML pages. The default setting specifies that data is inserted in UTF-8 format.

```
[content]
utf8-template-macros-enabled = yes
```

WebSEAL HTML pages use a UTF8 charset by default. If you modify charset to specify the local code page, set this entry to no.

Note that this setting affects files located in the directories specified by the error-dir and mgt-pages-root (in the [acnt-mgt] stanza) configuration file entries.
Modifying pages from prior versions of Tivoli Access Manager

Some modifications are required to use customized HTML pages (containing login forms) created in prior versions of Tivoli Access Manager. The following HTML pages are affected:
- login.html
- tokenlogin.html
- nexttoken.html
- stepuplogin.html

The updated versions of these pages POST to different form URLs and make use of a hidden field to pass the form type to WebSEAL.

If pages (forms) created in prior versions of Tivoli Access Manager are used with this current version of Tivoli Access Manager, the following symptoms are possible:
- 404 errors received when attempting to post the forms
- Authentication not allowed

The following modifications must be made to older pages (forms):
1. Every form must post to the following URL:
   /pkmslogin.form
2. The standard login forms must include the following hidden field:
   `<INPUT TYPE="HIDDEN" NAME="login-form-type" VALUE="pwd"></TD>`
3. The token login forms must include the following hidden field:
   `<INPUT TYPE="HIDDEN" NAME="login-form-type" VALUE="token"></TD>`

Examine the updated forms in the current version of Tivoli Access Manager as a reference for applying any of the above modifications to older forms.
Backup and restore

Tivoli Access Manager provides the pdbackup utility to backup and restore Tivoli Access Manager data. This utility can be used to backup and restore WebSEAL data.

The pdbackup utility reads a backup list file that specifies the files and directories to back up. The files and directories to backup for WebSEAL are listed in the following files:

UNIX
/opt/pdweb/etc/amwebbackup.lst

Windows
C:\Program Files\Tivoli\PDweb\etc\amwebbackup.lst

On Windows, the file list also lists registry keys to backup.

The file list is a plain text file. You can customize the file contents to add information specific to each deployment. For example, information about cross-domain peers, e-community servers, and e-community domain keys can be added. Follow the format of the amwebbackup.lst file.

Example:
[cldsso-peers]
machine_name = key_file_location

[e-community-sso]
master-authn-server = server_name

[e-community-domain-keys]
domain_name = key_file

Backup WebSEAL data

Use pdbackup to backup WebSEAL data. Specify the backup list file that the utility must read to obtain the list of necessary WebSEAL data. The pdbackup utility creates a default archive file. On UNIX this is a .tar file. On Windows, it is a .dir file. The file is placed in a default location. The default location is:

UNIX
/var/PolicyDirector/backup

Windows
C:\Program Files\Tivoli\PolicyDirector\pdbackup

When WebSEAL is installed in a non-default location on Windows, the pdbackup directory is a sub-directory under the installation location.

You can provide optional command line arguments to specify an alternative name and alternative location for the archive file.

The following example command creates the default archive file in the default directory.

UNIX
pdbackup -a backup -l /opt/pdweb/etc/amwebbackup.lst

Windows
pdbackup -a backup -l installation_dir\etc\amwebbackup.lst
Example archive file created by this command:

UNIX:
/var/PolicyDirector/pdbackup/amwebbackuplst_15jul2003.10_41.tar

Windows
\installation_dir\pdbackup\amwebbackuplst_15jul2003.10_41.dir

For further information on **pdbackup**, including syntax, options and examples, see the **pdbackup** reference page in the *IBM Tivoli Access Manager for e-business Command Reference*.

### Restore WebSEAL data

Use **pdbackup** to restore WebSEAL data from an archive file. You must specify the archive file on the command line. On UNIX, files are restored to the root directory. On Windows, files are restored to the original directory. On UNIX only, you can use the **-path** option to specify an alternative directory for file restoration.

The following command restores data when the archive file is located in the default location:

UNIX
```
pdbackup -a restore -f pdbackup.1st_29June2002.07_24.tar
```

Windows
```
pdbackup -a restore -f base_dir\pdbackup\pdbackup.1st_29Jun2002.07_24.dir
```

You can use further command line options to specify a non-default location for the archive file.

For further information on **pdbackup**, including syntax, options and examples, see the **pdbackup** reference page in the *IBM Tivoli Access Manager for e-business Command Reference*.

### Extract archived WebSEAL data

Use the *pdbackup* utility to extract archived WebSEAL data. Use this option to access one or more archived files without performing an entire restoration. Files are extracted into a single directory. A directory hierarchy is not built during extraction.

For example, the following command extract the contents of an archive file from /var/pdbackup (UNIX) or C:\pdback (Windows) to a directory named amwebextract.

UNIX
```
pdbackup -a extract -p amwebextract
    -f /var/pdbackup/pdbackup.1st_29Jun2002.07_25.tar
```

Windows
```
pdbackup -a extract -p e:\amwebextract
    -f c:\pdback pdbackup.1st_29Jun2002.07_25.dir
```

The above commands must be entered as one continuous command line.

For further information on **pdbackup**, including syntax, options and examples, see the **pdbackup** reference page in the *IBM Tivoli Access Manager for e-business Command Reference*. 

---

Chapter 3. WebSEAL server administration  93
Problem determination tools for WebSEAL

The following problem determination tools for WebSEAL are discussed in the *IBM Tivoli Access Manager Problem Determination Guide*.

- statistics utility
- `pdadmin trace` command
Chapter 4. Serviceability and logging

This chapter describes WebSEAL serviceability and logging features. WebSEAL uses routing files to record serviceability messages. WebSEAL uses Tivoli Access Manager event logging to capture and log authentication and HTTP events. WebSEAL also supports legacy Tivoli Access Manager auditing functions.

This chapter contains the following sections:

- “Logging WebSEAL serviceability messages” on page 96
- “Event capturing and logging” on page 98
- “Legacy auditing” on page 105
**Logging WebSEAL serviceability messages**

Tivoli Access Manager WebSEAL serviceability messages are controlled by the Tivoli Access Manager WebSEAL routing file. A routing file template is distributed with WebSEAL:

**UNIX:**

```
/opt/pdwebrte/etc/routing.template
```

**Windows:**

```
C:\Program Files\Tivoli\PDWeb\etc\routing.template
```

During installation, the routing template file is customized for the current system, and copied to a file named `routing` in the same directory.

The routing file is an ASCII file that contains additional documentation in the form of comment lines. Entries in this configuration file determine the types of serviceability messages that are logged. Enable any entry by removing the comment character (#). The routing file includes the following default entries:

**UNIX:**

```
FATAL:STDERR:--
ERROR:STDERR:--
WARNING:STDERR:--
#NOTICE:FILE.10.100:/opt/pdweb/log/msg__notice.log
#NOTICE_VERBOSE:FILE.10.100:/opt/pdweb/log/msg__notice.log
```

**Windows:**

```
FATAL:STDERR:--
ERROR:STDERR:--
WARNING:STDERR:--
#NOTICE:FILE.10.100:%PDWEBDIR%\log\msg__notice.log
#NOTICE_VERBOSE:FILE.10.100:%PDWEBDIR%\log\msg__notice.log
```

**Note:** On a Windows system, the special environment variable PDWEBDIR is set at run time to the WebSEAL installation directory.

By default, when WebSEAL runs in the foreground, all messages are sent to the screen (STDERR).

By default, when WebSEAL runs in the background, messages are redirected from STDERR and sent to the WebSEAL server log file as defined in the `[logging]` stanza of the WebSEAL configuration file:

- Server process: `webseald`
- Configuration file location: `webseald-instance_name.conf`
- Log file location:
  - **UNIX:**
    
    ```
    [logging]
    server-log=/var/pdweb/log/msg_webseald.log
    ```
  - **Windows:**
    
    ```
    [logging]
    server-log= C:\Program Files\Tivoli\PDWeb\log\msg_webseald.log
    ```

To enable `msg_verbose.log`, uncomment the NOTICE_VERBOSE line.
The FILE syntax for the NOTICE message controls log roll over and file recycling:

```
FILE.max_files.max_records
```

The `max_files` value specifies the number of files that are used.

The `max_records` value specifies the maximum number of entries per file.

In the default example above, FILE.10.100 means there are 10 files created, each with a maximum of 100 entries. The files are named:

notice.log.1
notice.log.2
.
.
notice.log.10

The messages “wrap around” to the first file after the last file has reached its limit or when the server is stopped and restarted. When a log file is reused, the existing records are written over (erased).

In some cases, when files are created in the background by WebSEAL, the files can be owned by root. To ensure that WebSEAL can write messages to these log files, you can optionally specify file permissions, owner and group. In the following example, the optional arguments that accomplish this are 

```
".666:ivmgr:ivmgr": 
```

```
```

**Note:** For more information on routing file configuration options, review the routing file template that is distributed with WebSEAL. For more information on Tivoli Access Manager support for serviceability messages, see the IBM Tivoli Access Manager for e-business Problem Determination Guide.

### Serviceability messages in UTF-8 format

WebSEAL produces serviceability messages using UTF-8 encoding. When the operating system uses a non-UTF-8 local code page, WebSEAL automatically converts serviceability messages to a data format to match the non-UTF-8 local code page. In some cases, the conversion from UTF-8 to non-UTF-8 encoding can result in data loss. For this reason, it is recommended that the WebSEAL process run in a locale environment that uses UTF-8 encoded code pages.

It is possible to obtain UTF-8 logs when running in a non-UTF-8 locale. Use the UTF8FILE type in the routing file. For example,

```
ERROR:UTF8FILE:/var/pdweb/log/msg_error.log
```

For more information, see the routing file template that is distributed with WebSEAL:

```
```

Note that when data loss occurs, the routing file entry contains a series of question marks (????) at the location where data conversion was problematic.

For more information, see “Multi-locale support with UTF-8” on page 44.
Event capturing and logging

You can capture events for logging and auditing by using the Tivoli Access Manager event logging facility. Event logging provides a structured hierarchy for gathering messages for logging and auditing purposes. The event logging feature also supports the use of alternate destinations for logging output, such as consoles (stdout), pipes, and remote servers.

The event logging facility has many different configuration options. This chapter describes how to configure event logging to capture common events generated by WebSEAL. Before using this chapter, it is recommended that you read the overview of event logging in the IBM Tivoli Access Manager Base Administration Guide. This overview provides many details on configuration options, including output destinations, that might apply to your deployment.

**Note:** WebSEAL also supports the Tivoli Access Manager legacy auditing model. Use of this model is recommended for backwards compatibility only. For more information, see “Legacy auditing” on page 105.

Topic index:
- “Event logging configuration tasks”
- “Example configuration” on page 99
- “Configuring HTTP logging using event logging” on page 100
- “Authentication event log output” on page 102
- “Audit data in UTF-8 format” on page 104

Event logging configuration tasks

The following configuration tasks are required for each WebSEAL audit trail file:

1. Enable auditing
   Enable the creation of audit records.

2. Specify type of audit event.
   Supported types of audit events for WebSEAL:
   - Authorization
   - Credential acquisition authentication
   - HTTP requests

3. Specify audit file location
   Location on the filesystem for the audit records.

4. Specify audit file size
   Maximum size of an audit trail file. When the maximum size is reached, the file is backed up and a new file is started.

5. Specify file flush interval
   Frequency with which the server flushes audit trail buffers to the file.

The following configuration tasks are optional for each WebSEAL audit trail file:

- Specify maximum event queue
  Maximum number of events to queue in memory.
- Specify event queue high water mark
  Number of events in queue which trigger a flush from memory to file.
- Specify maximum buffer size
Maximum size, in bytes, of event buffer in memory to be built from individual events.

- Specify file mode
  Binary or text. Text mode is available for Windows platform only.

The optional configuration tasks are supported by Tivoli Access Manager event logging but not by legacy auditing. For more information on these tasks, see IBM Tivoli Access Manager Base Administration Guide.

Example configuration

The syntax for using logcfg to configure an audit trail file is:

```
logcfg category:file path=file_pathname, flush_interval=seconds, \
rollover_size=number, log_id=logid, queue_size=number, hi_water=number, 
buffer_size=number, mode={text|binary}
```

Supported values for category are:

<table>
<thead>
<tr>
<th>Audit event type</th>
<th>Category setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credentials acquisition authentication</td>
<td>audit.authn</td>
</tr>
<tr>
<td>Authorization</td>
<td>audit.azn</td>
</tr>
<tr>
<td>HTTP logging information</td>
<td>http</td>
</tr>
<tr>
<td>HTTP request information in common log format</td>
<td>http.clf</td>
</tr>
<tr>
<td>HTTP Referer header information</td>
<td>http.ref</td>
</tr>
<tr>
<td>HTTP User Agent header information</td>
<td>http.agent</td>
</tr>
<tr>
<td>The NCSA combined format captures HTTP request information (with time stamp) and appends the quoted referer and agent strings to the standard common log format.</td>
<td>http.cof</td>
</tr>
</tbody>
</table>

For example, the following logcfg entry creates a WebSEAL audit trail file that collects authentication events:

```
logcfg audit.authn:file path=/var/pdweb/log/audit.log,flush_interval=20, \
rollover_size=2000000
```

**Note:** The above example is entered as one continuous line in the WebSEAL configuration file.

In this example, auditing is enabled when Tivoli Access Manager reads the logcfg entry at startup time. The required configuration settings are provided by the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit.authn:file path</td>
<td>Type of audit event</td>
</tr>
<tr>
<td>path=/var/pdweb/log/audit.log</td>
<td>Audit file location</td>
</tr>
<tr>
<td>rollover_size=2000000</td>
<td>Audit file size</td>
</tr>
<tr>
<td>flush_interval=20</td>
<td>File flush interval</td>
</tr>
</tbody>
</table>
In this example, none of the optional configuration settings are specified. When the optional settings are not specified, Tivoli Access Manager uses default values for each setting:

Table 17. Default values for optional configuration settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Optional setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_size=number</td>
<td>Maximum event queue. The default value is 0. This value means that no maximum size is imposed.</td>
</tr>
<tr>
<td>hi_water=number</td>
<td>Event queue high water mark. The default value is two-thirds of the queue_size. When queue_size is 0, the default is 100.</td>
</tr>
<tr>
<td>buffer_size=number</td>
<td>Maximum buffer size. Default size is 0, which disables buffering.</td>
</tr>
<tr>
<td>mode={text</td>
<td>binary}</td>
</tr>
</tbody>
</table>

The logcfg entries are placed in the WebSEAL configuration file, under the [aznapi-configuration] stanza. Create a separate logcfg entry for each type of audit event.

See also:
- The auditing section of the WebSEAL configuration file reference: “Auditing” on page 474.
- The logging and auditing chapter of the IBM Tivoli Access Manager Base Administration Guide.

Configuring HTTP logging using event logging

HTTP logging using event logging can be configured in the [aznapi-configuration] stanza of the WebSEAL configuration file. Use the logcfg event logging parameter to define one or more log agents (loggers) that gather a specific category of log information from the event pool and direct this information to a destination:

```
[aznapi-configuration]
logcfg = category:{stdout|stderr|file|pipe|remote} [[param=value]] [,param=value]]
```

The values for category that are appropriate for HTTP logging include:
- http
  All HTTP logging information
- http.clf
  HTTP request information in common log format
- http.ref
  HTTP Referer header information
- http.agent
  HTTP User_Agent header information
- http.cof
  The NCSA combined format captures HTTP request information (with time stamp) and appends the quoted referer and agent strings to the standard common log format.

Compatibility with legacy auditing settings
The following log agent configurations are enabled when the legacy WebSEAL HTTP logging parameters are enabled (see "Legacy auditing" on page 105). Note that the log agent configurations accept the values of the requests-file, referers-file, agents-file, flush-time, and max-size parameters from the WebSEAL configuration file [logging] stanza:

**request.log** (common log format):
```
logcfg = http.clf:file path=<requests-file>,flush=<flush-time>, \ 
rollover=<max-size>,log=clf,buffer_size=8192,queue_size=48
```

**referer.log**:
```
logcfg = http.ref:file path=<referers-file>,flush=<flush-time>, \ 
rollover=<max-size>,log=ref,buffer_size=8192,queue_size=48
```

**agent.log** (common log format):
```
logcfg = http.agent:file path=<agents-file>,flush=<flush-time>, \ 
rollover=<max-size>,log=agent,buffer_size=8192,queue_size=48
```

Because legacy HTTP logging is configured in a different stanza ([logging]) than event logging configuration ([aznapi-configuration]), it is possible to have two duplicate entries for each event appear in a log file when both logging mechanisms are enabled.

For more information on setting logcfg, refer to the IBM Tivoli Access Manager Base Administration Guide.

**HTTP event log output**

The "http" audit tag configuration captures the same information as the standard HTTP log files (request.log, referrer.log, and agent.log).

The following is a sample HTTP audit record from a "change password" event:
```
<date>2003-10-29-10:30:35.003-08:00</date>
<outcome status="953091558">1</outcome>
<originator blade="webseald"><component rev="1.1">http</component>
<action>2</action>
<location>meow</location>
</originator>
<accessor name="">
<principal auth="IV_LDAP_V3.0" domain="Default">testuser</principal>
</accessor>
<target resource="5"><object><![CDATA[/pkmspasswd.form]]]>;;;</object></target>
<data>
<![CDATA[POST /pkmspasswd.form HTTP/1.1]]]>;;;
512
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461; .NET CLR 1.0.3705)
<![CDATA[https://meow.cruz.ibm.com/pkmspasswd]]]>;;;
</data>
</event>
```

```
<event rev="1.2">
<date>2003-10-29-10:30:40.008-08:00</date>
<outcome status="953091102">1</outcome>
<originator blade="webseald"><component rev="1.1">http</component>
<action>2</action>
<location>meow</location>
</originator>
<target resource="5"><object></object></target>
<data>
</data>
</event>
```
Authentication event log output

Authentication of a principal is performed during credential acquisition. Audit records can be captured by Tivoli Access Manager to record the success or failure of such authentication attempts.

The following is a sample authentication event logged from WebSEAL for an unauthenticated user.

```
<event rev="1.2">
<date>2001-11-14-23:04:26.630+00:00I-----</date>
<outcome status="0">0</outcome>
<originator blade="webseald"><component rev="1.2">authn</component>
<action>0</action>
<location>phaedrus</location>
</originator>
<accessor name="">
<principal auth="invalid"></principal>
</accessor>
<target resource="7"><object></object></target>
<data>
</data>
</event>
```

The following is a sample authentication event logged from WebSEAL for an authenticated user.

```
<event rev="1.2">
<date>2001-11-14-15:56:06.551+00:00I-----</date>
<outcome status="0">0</outcome>
<originator blade="webseald"><component rev="1.2">authn</component>
<action>0</action>
<location>phaedrus</location>
</originator>
<accessor name="">
<principal auth="IV_LDAP_V3.0" domain="Default">testuser2</principal>
</accessor>
<target resource="7"><object></object></target>
<data>
</data>
</event>
```

The following is a sample authentication failure due to a bad password. Note that the outcome value is 1.

```
<event rev="1.2">
<date>2003-10-21-17:23:29.250-07:00I-----</date>
<outcome status="320938184">1</outcome>
<originator blade="webseald"><component rev="1.2">authn</component>
<action>0</action>
<location>meow</location>
</originator>
<accessor name="">
<principal auth="password" domain="Default">testuser</principal>
</accessor>
<target resource="7"><object></object></target>
<data>
Password Failure: testuser</data>
</event>
```

The following is a sample authentication failure (due to expired password) event logged from WebSEAL. Note that the outcome is 0. The status value 320938188 indicates that the failure was due to an expired password. This status value is documented in the IBM Tivoli Access Manager Error Message Reference.
The following is a sample authentication failure event due to too many invalid login attempts (three strikes policy) logged from WebSEAL. Note that the outcome is 1.

The following is a sample successful change password event logged from WebSEAL.

An outcome status of "0" indicates a successful change password. Any other outcome indicates a failure status code.

The following table lists the authentication error codes and their <data> element structures that are returned when an authentication attempt fails:

Table 18. Authentication errors

<table>
<thead>
<tr>
<th>Error type</th>
<th>Error code (in hex)</th>
<th>Error code (in decimal)</th>
<th>XML generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password failure</td>
<td>132120c8</td>
<td>320938184</td>
<td>&lt;data&gt;Password failure: user &lt;/data&gt;</td>
</tr>
</tbody>
</table>
To determine the reason for an audited event such as an account lock-out (due, for example, to a three strikes policy), obtain the error code as shown in the table above. The error code is contained in the audit output in the `<outcome status>` tag:

```
<outcome status>13212132</outcome>
```

Invoke the `padmin errtext` command for the error code to receive a reason for the outcome. For example:

```
> padmin errtext 13212132
This account has been temporarily locked out due to too many failed login attempts
```

Error codes for Tivoli Access Manager are described in the *IBM Tivoli Access Manager Error Message Reference*.

### Audit data in UTF-8 format

WebSEAL produces audit data using UTF-8 encoding. When the operating system uses a non-UTF-8 local code page, WebSEAL automatically converts audit data to a data format to match the non-UTF-8 local code page. In some cases, the conversion from UTF-8 to non-UTF-8 encoding can result in data loss. For this reason, it is recommended that the WebSEAL process run in a locale environment that uses UTF-8 encoded code pages.

It is possible to obtain UTF-8 logs when running in a non-UTF-8 locale. Use the `UTF8FILE` type in the routing file.

Note that when data loss occurs, the audit file entry contains a series of question marks (????) at the location where data conversion was problematic.

For more information, see “Multi-locale support with UTF-8” on page 44.
Legacy auditing

Legacy auditing is configured by supplying a value for each the following keys:
[aznapi-configuration]
logaudit
auditlog
auditcfg
logsize
logflush

Use of this method is comparable to the event logging method, when directing output to a file. Note, however, that the event logging method provides additional control over buffer size and event queues. Also, legacy auditing does not support output to consoles, pipes, or remote servers.

Legacy auditing for authentication

To use legacy auditing to accomplish the configuration tasks for the authentication auditing example above, the comparable configuration file entries would be:
[aznapi-configuration]
logaudit = yes
auditcfg = authn
auditlog = /var/pdweb/log/audit.log
logsize = 2000000
logflush = 20

Legacy auditing does not support the optional configuration settings.

For more information on legacy auditing configuration settings, see IBM Tivoli Access Manager Base Administration Guide.

Legacy auditing for HTTP

WebSEAL maintains three conventional HTTP log files that record activity rather than messages:
- request.log
- agent.log
- referer.log

By default, these log files are located under the following directory:

UNIX:
/var/pdweb/www/log/

Windows:
C:\Program Files\Tivoli\PDWeb\www\log\n
Parameters for configuring standard HTTP logging are located in the [logging] stanza of the WebSEAL configuration file.

The following table illustrates the relationship between the HTTP log files and the configuration file parameters:

<table>
<thead>
<tr>
<th>Log Files</th>
<th>Location Parameter</th>
<th>Enable/Disable Parameter (= yes or no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>request.log</td>
<td>requests-file</td>
<td>requests</td>
</tr>
<tr>
<td>Log Files</td>
<td>Location Parameter</td>
<td>Enable/Disable Parameter ( = yes or no)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>referer.log</td>
<td>referers-file</td>
<td>referers</td>
</tr>
<tr>
<td>agent.log</td>
<td>agents-file</td>
<td>agents</td>
</tr>
</tbody>
</table>

For example, the entry for the default location of the request.log file appears as follows:

**UNIX:**

requests-file = /var/pdweb/www/log/request.log

**Windows:**

requests-file = \Program Files\Tivoli\PDWeb\www\log\request.log

**Enabling and disabling HTTP logging**

By default, all HTTP logging is enabled:

```plaintext
[logging]
requests = yes
referer = yes
agents = yes
```

Each log can be enabled or disabled independently from the others. If any parameter is set to "no", logging is disabled for that file.

**Specifying the timestamp type**

You can choose to have the timestamps in each log recorded in Greenwich Mean Time (GMT) instead of the local time zone. By default, the local time zone is used:

```plaintext
[logging]
gmt-time = no
```

To use GMT timestamps, set:

```plaintext
gmt-time = yes
```

**Specifying log file rollover thresholds**

The `max-size` parameter specifies the maximum size to which each of the HTTP log files may grow and has the following default value (in bytes):

```plaintext
[logging]
max-size = 2000000
```

When a log file reaches the specified value — known as its rollover threshold — the existing file is backed up to a file of the same name with an appended current date and timestamp. A new log file is then started.

The various possible `max-size` values are interpreted as follows:

- If the `max-size` value is less than zero (< 0), then a new log file is created with each invocation of the logging process and every 24 hours from that instance.
- If the `max-size` value is equal to zero (= 0), then no rollovers are performed and the log file grows indefinitely. If a log file already exists, new data is appended to it.
- If the `max-size` value is greater than zero (> 0), then a rollover is performed when a log file reaches the configured threshold value. If a log file already exists at startup, new data is appended to it.
Specifying the frequency for flushing log file buffers

Log files are written to buffered data streams. If you are monitoring the log files in real time, you might want to alter the frequency with which the server forces a flush of the log file buffers.

By default, log files are flushed every 20 seconds:

```
[logging]
flush-time = 20
```

If you specify a negative value, a flush will be forced after every record is written.

HTTP common log format (for request.log)

Every response (success or failure) sent back by the Tivoli Access Manager server is recorded with a one-line entry in the request.log file using following HTTP Common Log Format:

```
host - authuser [date] request status bytes
```

where:

- **host** Specifies the IP address of the requesting machine.
- **authuser** This field contains the identity information of the user. The value "unauth" is used for an unauthenticated user.
- **date** Specifies the date and time of the request.
- **request** Specifies the first line of the request as it came from the client.
- **status** Specifies the HTTP status code sent back to the requesting machine.
- **bytes** Specifies the number of bytes sent back to the requesting machine. This value—either the unfiltered content size or a zero size—is configured with the `log-filtered-pages` parameter.

Displaying the request.log file

The request.log records standard logging of HTTP requests, such as information on URLs that have been requested and information on the client (for example, IP address) that made the request.

The following example shows a sample version of a request.log file:

```
130.15.1.90 - Unauth [09/Oct/2003: 10:12:06 -0700] "GET / HTTP/1.1" 200 1979
130.15.1.90 - Unauth [09/Oct/2003: 11:07:18 -0700] "GET / HTTP/1.1" 200 1052
```

Displaying the agent.log file

The agent.log file records the contents of the User_Agent: header in the HTTP request. This log reveals information about the client browser, such as architecture or version number, for each request.

The following example shows a sample version of a agent.log file:

```
Mozilla/4.01 [en] (WinNT; U)
Mozilla/4.01 [en] (WinNT; U)
Mozilla/4.01 [en] (WinNT; U)
Mozilla/4.01 [en] (WinNT; U)
```
Displaying referer.log

The referer.log records the Referer: header of the HTTP request. For each request, the log records the document that contained the link to the requested document.

The log uses the following format:
referer -> object

This information is useful for tracking external links to documents in your Web space. The log reveals that the source indicated by referer contains a link to a page object. This log allows you to track stale links and to find out who is creating links to your documents.

The following example shows a sample version of a referer.log file:

http://manuel/maybam/index.html -> /pics/tivoli_logo.gif
http://manuel/maybam/pddl/index.html -> /pics/tivoli_logo.gif
http://manuel/maybam/ -> /pddl/index.html
http://manuel/maybam/ -> /pddl/index.html
http://manuel/maybam/pddl/index.html -> /pics/tivoli_logo.gif
http://manuel/maybam/ -> /pddl/index.html

Log data in UTF-8 format

WebSEAL produces log data using UTF-8 encoding. When the operating system uses a non-UTF-8 local code page, WebSEAL automatically converts log data to a data format to match the non-UTF-8 local code page. In some cases, the conversion from UTF-8 to non-UTF-8 encoding can result in data loss. For this reason, it is recommended that the WebSEAL process run in a locale environment that uses UTF-8 encoded code pages.

For more information, see "Multi-locale support with UTF-8" on page 44.
Chapter 5. WebSEAL security policy

This chapter contains information that describes how you can configure and customize WebSEAL security policy.

Topic Index:
- “WebSEAL-specific ACL policies” on page 110
- “Configuring three strikes login policy” on page 112
- “Configuring password strength policy” on page 114
- “Authentication strength policy” on page 117
- “Quality of protection POP policy” on page 126
- “Handling unauthenticated users (HTTP / HTTPS)” on page 126
WebSEAL-specific ACL policies

The following security considerations apply for the /WebSEAL container in the protected object space:

- The WebSEAL object begins the chain of ACL inheritance for the WebSEAL region of the object space
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire Web space
- The traverse permission is required for access to any object below this point

Refer to the IBM Tivoli Access Manager Base Administrator’s Guide for complete information about Tivoli Access Manager ACL policies.

/WebSEAL/host-instance_name

This sub-directory entry represents the beginning of the Web space for a particular WebSEAL server instance. The following security considerations apply for this object:

- The traverse permission is required for access to any object below this point
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire object space on this machine

/WebSEAL/host-instance_name/file

This sub-directory entry represents the resource object checked for HTTP access. The permissions checked depend on the operation being requested.

WebSEAL ACL permissions

The following table describes the ACL permissions applicable for the WebSEAL region of the object space:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>read</td>
</tr>
<tr>
<td>x</td>
<td>execute</td>
</tr>
<tr>
<td>d</td>
<td>delete</td>
</tr>
<tr>
<td>m</td>
<td>modify</td>
</tr>
</tbody>
</table>
| l         | list        | Required by policy server to generate an automated directory listing of the Web space.  
This permission also governs whether a client can see the directory contents listing when the default "index.html" page is not present. |
| g         | delegation  | Assigns trust to a WebSEAL server to act on behalf of a client and pass requests to a junctioned WebSEAL server. |

Default /WebSEAL ACL policy

Core entries for the WebSEAL ACL, default-webseal, include:

- Group iv-admin: Tcmdbsvarxl
- Group webseal-servers: Tgmdbsrxl
- User sec_master: Tcmdbsvarxl
- Any-other: Trx
- Unauthenticated: T
At installation, this default ACL is attached to the /WebSEAL container object in the object space.

The group, webseal-servers, contains an entry for each WebSEAL server in the secure domain. The default permissions allow the servers to respond to browser requests.

The traverse permission allows expansion of the Web space as represented in the Web Portal Manager. The list permission allows the Web Portal Manager to display the contents of the Web space.

Valid characters for ACL names

The following characters are valid for creating ACL names:

- A-Z
- a-z
- underscore (_)
- hyphen (-)
- backslash (\)
- Any character from a double-byte character set

For detailed information about creating ACL names, refer to the IBM Tivoli Access Manager Base Administration Guide.
Configuring three strikes login policy

The three strikes login policy, available for LDAP-based Tivoli Access Manager installations, enables you to specify a maximum number of failed login attempts (n) and a penalty lockout time (x), such that after “n” failed login attempts a user is locked out for “x” seconds (or the account is disabled).

The three strikes login policy is used to prevent computer password attacks. The policy creates a condition where a user must wait a period of time before making more login attempts that fail. For example, a policy could dictate 3 failed attempts followed by a 180 second penalty. This type of login policy can prevent random computer-generated login attempts occurring many times a second.

The three strikes login policy requires the joint contribution of two pdadmin policy command settings:

- Maximum number of failed login attempts
  policy set max-login-failures
- Penalty for exceeding failed login attempt setting
  policy set disable-time-interval
  The penalty setting can include an account lockout time interval or a complete disabling of the account.

If a login policy is set (as an example) for three failed attempts followed by specific lockout time penalty, a fourth attempt (correct or incorrect) will result in an error page that states the account is temporarily unavailable because of password policy.

The time interval is specified in seconds—the minimum recommended time interval is 60 seconds.

If the disable-time-interval policy is set to "disable", the user is locked out of the account and the LDAP account valid attribute for this user is set to "no". An administrator re-enables the account through the Web Portal Manager.

Note: Setting the disable-time-interval to "disable" results in additional administration overhead, because the account must be manually re-enabled by the administrator. Once the account is re-enabled, the updated account valid information might not be immediately available. This situation can occur when using WebSEAL with an LDAP environment that includes replicated LDAP servers. In this case, the updated information is propagated to the LDAP replicas according to the LDAP configuration settings that specify the time interval for performing updates.

Account lock policy with load-balanced WebSEAL servers

You use the three strikes login policy to ensure that an account is locked after a specified number of login attempts. This policy performs as expected in a configuration involving one WebSEAL server. In a configuration involving multiple front-end WebSEAL servers with a load-balancing mechanism, the results of the policy are affected by the fact that each WebSEAL server maintains its own local count of failed login attempts.

For example, if the max-login-failures value is set to three (3) attempts, and the client fails the first three attempts, the account on this server is locked. However, as the client continues login attempts, the load-balancing mechanism—detecting a
failure to connect to the first server—re-directs the request to another available replicated WebSEAL server. Now the client has three more opportunities to attempt a successful login.

For "n" attempts configured on each WebSEAL server, and "m" front-end replicated WebSEAL servers, you are guaranteed an initial account lock on one server after "n" attempts. You are also guaranteed "n" x "m" total attempts to log in across all configured servers. However, after "n" attempts, it is not clear whether subsequent authentication failures are due to the lock on a particular server, or due to continuing incorrect login attempts across the remaining replicated servers.

The "n" x "m" calculation provides a fixed maximum upper limit on the total number of consecutive login attempts before a complete lockout occurs. A case can be made that this number is still probably far less than the number of attempts statistically required to "break" a password.

If your business security solution requires a three strikes login policy, understand the implications of a load-balanced/multiple front-end WebSEAL configuration on this policy.

**Syntax for three strikes login commands**
The following `pdadmin` commands are used to set three strikes login policy.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy set max-login-failures [number] [-user &lt;username&gt;]</td>
<td>Manages the policy controlling the maximum number of failed login attempts allowed before a penalty is imposed. This command depends on a penalty set in the <code>policy set disable-time-interval</code> command. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the LDAP registry. The <code>unset</code> parameter eliminates settings for the specified user. The global policy will continue to be enforced. The default setting is 10 attempts.</td>
</tr>
<tr>
<td>policy set disable-time-interval [number] [-user &lt;username&gt;]</td>
<td>Manages the penalty policy controlling the time period an account should be disabled if the maximum number of failed login attempts is reached. As the administrator, you can apply this penalty policy to a specific user or apply the policy globally to all users listed in the LDAP registry. The <code>unset</code> parameter eliminates settings for the specified user. The global policy will continue to be enforced. The default setting is 180 seconds.</td>
</tr>
</tbody>
</table>

Chapter 5. WebSEAL security policy   113
Configuring password strength policy

Password strength policy refers to the stipulations placed on the construction of a password by password policy rules. Tivoli Access Manager provides two means of controlling password strength policy:

- Five pdadmin password policy commands
- You can write a customized authentication module to enforce your password policy

Refer to the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

Password strength policy set by the pdadmin utility

The five password strength attributes implemented through the pdadmin utility include:

- Minimum password length
- Minimum alphabetic characters
- Minimum non-alphabetic characters
- Maximum repeated characters
- Spaces allowed

These policies are enforced when you create a user with pdadmin or the Web Portal Manager, and when a password is changed with pdadmin, the Web Portal Manager, or the pkmspasswd utility.

Syntax for password strength policy commands

The following pdadmin commands, used to set password strength policy, are appropriate for use only with an LDAP registry. The unset option disables this policy attribute—that is, the policy is not enforced.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy set min-password-length (&lt;number&gt;) (\text{set}) [(-\text{user }&lt;username&gt;)]</td>
<td>Manages the policy controlling the minimum length of a password.</td>
</tr>
<tr>
<td></td>
<td>As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.</td>
</tr>
<tr>
<td></td>
<td>The default setting is 8.</td>
</tr>
<tr>
<td>policy set min-password-alphas (&lt;number&gt;) (\text{sets}) [(-\text{user }&lt;username&gt;)]</td>
<td>Manages the policy controlling the minimum number of alphabetic characters allowed in a password.</td>
</tr>
<tr>
<td></td>
<td>As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.</td>
</tr>
<tr>
<td></td>
<td>The default setting is 4.</td>
</tr>
<tr>
<td>policy set min-password-non-alphas (&lt;number&gt;) (\text{sets}) [(-\text{user }&lt;username&gt;)]</td>
<td></td>
</tr>
<tr>
<td>policy get min-password-length (-\text{user }&lt;username&gt;)</td>
<td></td>
</tr>
<tr>
<td>policy get min-password-alphas (-\text{user }&lt;username&gt;)</td>
<td></td>
</tr>
<tr>
<td>policy get min-password-non-alphas (-\text{user }&lt;username&gt;)</td>
<td></td>
</tr>
</tbody>
</table>
Command | Description
--- | ---
Manages the policy controlling the minimum number of non-alphabetic (numeric) characters allowed in a password.
As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.
The default setting is 1.

policy set max-password-repeated-chars \(<\text{number}> | \text{unset} \) [-user <username>]
policy get max-password-repeated-chars [-user <username>]

Manages the policy controlling the maximum number of repeated characters allowed in a password.
As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.
The default setting is 2.

policy set password-spaces \(\text{yes} | \text{no} | \text{unset} \) [-user <username>]
policy get password-spaces [-user <username>]

Manages the policy controlling whether a password can contain spaces.
As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.
The default setting is unset.

**Default policy parameter values**
The following table lists the policy parameters and the default values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>min-password-length</td>
<td>8</td>
</tr>
<tr>
<td>min-password-alphas</td>
<td>4</td>
</tr>
<tr>
<td>min-password-non-alphas</td>
<td>1</td>
</tr>
<tr>
<td>max-password-repeated-chars</td>
<td>2</td>
</tr>
<tr>
<td>password-spaces</td>
<td>not set</td>
</tr>
</tbody>
</table>

To create the password policy behavior found in earlier releases of Tivoli Access Manager, apply the **unset** option to each of the five password parameters listed above.

**Valid and invalid password examples**
The following table illustrates several password examples and the policy results based on the default values of the five **pdadmin** parameters:

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Not valid: must contain at least one non-alphabetic character.</td>
</tr>
<tr>
<td>pass</td>
<td>Not valid: must contain at least 8 characters.</td>
</tr>
<tr>
<td>passs1234</td>
<td>Not valid: contains more than two repeated characters.</td>
</tr>
<tr>
<td>12345678</td>
<td>Not valid: must contain at least four alphabetic characters.</td>
</tr>
<tr>
<td>password3</td>
<td>Valid.</td>
</tr>
</tbody>
</table>
**Specific user and global settings**

The `pdadmin policy` commands can be set for a specific user (with the `-user` option) or globally (by not using the `-user` option). Any user-specific setting overrides a global setting for the policy. You can also disable (`unset`) a policy parameter, which means the parameter contains no value. Any policy with the `unset` option is not checked or enforced.

For example:
```
pdadmin> policy set min-password-length 8
pdadmin> policy set min-password-length 4 -user matt
pdadmin> policy get min-password-length
Minimum password length: 8
pdadmin> policy get min-password-length -user matt
Minimum password length: 4
```

(User **matt** has a minimum password length policy of 4 characters; all other users have a minimum password length policy of 8.)
```
pdadmin> policy set min-password-length unset -user matt
```

(User **matt** is now governed by the global minimum password length policy of 8 characters.)
```
pdadmin> policy set min-password-length unset
```

(All users, including user **matt**, now have no minimum password length policy.)
Authentication strength policy

This section consists of the following topics:

- “Overview of authentication strength”
- “Authentication strength configuration” on page 119

Overview of authentication strength

WebSEAL supports many authentication methods. These include basic authentication, forms authentication, token authentication, certificate authentication, and others. Any client that accesses a WebSEAL server has an authentication state, such as unauthenticated or token, which indicates the method by which the client last authenticated with WebSEAL.

WebSEAL provides a feature that enables administrators to assign a ranking or level to some of the authentication methods. Administrators can define an ordered list that ranks each authentication method from lowest to highest. This hierarchal ranking can be arbitrarily tailored to each individual WebSEAL deployment.

There is no absolute ranking between the authentication methods. This means that no one authentication method is inherently better or stronger than another method. The ranking is simply a method for an administrator to define a relative level for each authentication method for use with a specific Tivoli Access Manager WebSEAL protected object namespace. The only rule governing the assignment of levels is that the unauthenticated level is always lower than all other authenticated levels.

This set of authentication levels can be used to implement an authentication strength policy. Authentication strength is sometimes called step-up authentication. Note, however, that step-up authentication is not a unique authentication method like forms authentication or certificate authentication. Instead, it is a defined process for allowing users to change their current authentication method to another authentication method.

The concept of changing the authentication method is useful as a way of providing additional protection for selected resources in the IBM Tivoli Access Manager WebSEAL protected object namespace. For example, a user can log in using certificate authentication, and then access many resources that are protected by Tivoli Access Manager security. When the user attempts to access a more sensitive resource, which has been marked to require a higher level of access, the user is prompted to log in to a different authentication level.

Note that when a user activates authentication strength by attempting to access a protected object, the user does not have to log out first. Instead, the user is presented with a login prompt, and simply logs in again to the higher level.

Users can change authentication strength multiple times per authentication session. When a user increases authentication strength, the user can jump directly to any level that is higher than the user’s current level.

The following authentication methods can be assigned an authentication level:

- unauthenticated
- password authentication

Note that this covers both basic authentication and forms authentication. Both of these methods are implemented by the same built-in shared library in WebSEAL.
• token authentication
• certificate authentication

Authentication strength is supported over both HTTP and HTTPS, with the exception of certificate authentication. Since certificates are valid only over an SSL connection, it is not possible to step up to certificates over HTTP. If an object that requires certificate authentication is requested over HTTP, an error page will be served, as specified by the certstepup entry in the [acnt-mgt] stanza of the WebSEAL configuration file.

Administrators apply the authentication levels to a protected resource by declaring and attaching a Tivoli Access Manager protected object policy (POP) to the resource object. The POP is a standard Tivoli Access Manager POP. Authentication strength policy is set and stored in a POP attribute called an IP Endpoint Authentication Method. The attribute takes an integer value that represents the authentication level. The lowest level, unauthenticated, is always 0. Each level increases the integer index up to the total number of authentication methods that have been assigned a level.

When clients first authenticate to WebSEAL, the authentication method used is stored as an extended attribute in the client’s credential. The Tivoli Access Manager authorization service compares the authentication method (level) in the credential against the authentication level for the requested resource, as specified in the POP. When the level in the POP exceeds the level in the credential, the user is prompted to increase the authentication strength level.

The IP Endpoint Authentication Method attribute can also optionally be used to restrict access to a resource, based on the network address of the client that sent the access request. The access can be restricted based on an individual network (IP) address, or a range of network addresses.

WebSEAL uses the following algorithm to process the conditions in a POP:

1. Check the IP endpoint authentication method policy on the POP.
2. Check ACL permissions.
3. Check time-of-day policy on the POP.
4. Check the audit level policy on the POP.
Authentication strength configuration

To configure authentication strength levels, complete the instructions in each of the following sections:

1. “Establish an authentication strength policy”
2. Stop the WebSEAL server.
3. “Specify authentication levels”
4. “Specify the authentication strength login form” on page 120
5. Restart the WebSEAL server.
6. “Create a protected object policy” on page 121
7. “Specify network-based access restrictions” on page 122
8. “Attach a protected object policy to a protected resource” on page 124
9. “Enforce user identity match across authentication levels” on page 125

Establish an authentication strength policy

This section consists of planning steps to be taken before specifying authentication strength settings in the WebSEAL configuration file.

Complete the following steps:

1. Compile a list of protected objects for which access will be limited only to users who have successfully authenticated through a specific authentication method. For each protected object, specify the authentication method that applies.
2. Compile a complete list of all authentication mechanisms that will be active (enabled) on the WebSEAL server system.
3. Determine a hierarchy (ranking) for the active authentication mechanisms. Order the mechanisms from weakest to strongest.
4. Determine if, during authentication strength level step-up, the user identity must be identical across the increased authentication level.
5. Determine if any protected resources require access restriction based on the network address of the requesting client.

Specify authentication levels

Complete the following steps:

1. Edit the [authentication-levels] stanza in the WebSEAL configuration file. For each authentication method to be used for authentication level step-up, add an entry to the stanza.

   The supported authentication methods are described in the following table:

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>Configuration file entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>level = unauthenticated</td>
</tr>
<tr>
<td>Basic authentication</td>
<td>level = password</td>
</tr>
<tr>
<td>Forms authentication</td>
<td>level = token</td>
</tr>
<tr>
<td>Certificate authentication</td>
<td>level = certificate</td>
</tr>
</tbody>
</table>

   The default entries are:
   
   [authentication-levels]
   level = unauthenticated
   level = password

Chapter 5. WebSEAL security policy 119
The entry `level = unauthenticated` must always be the first in the list. Additional entries can be placed in any order.

For example, to enable authentication strength levels for certificate authentication at the highest level, the completed stanza entry would be:

```
[authentication-levels]
level = unauthenticated
level = password
level = certificate
```

**Note:** There should be only one entry for each authentication mechanism.

2. Verify that each authentication method listed in `[authentication-levels]` is enabled. To determine if an authentication method is enabled, check the appropriate entries in the WebSEAL configuration file. To review the necessary entries and access the authentication configuration instructions, see the following sections:

   - “Enabling and disabling basic authentication” on page 145
   - “Enabling and disabling forms authentication” on page 147
   - “Enable token authentication” on page 163.
   - “Enable certificate authentication” on page 151

**Note:** Basic authentication is enabled by default.

**Specify the authentication strength login form**

When a client attempts to access a protected resource, and is required to reauthenticate to a higher authentication strength level, WebSEAL presents a special HTML form. The client uses the form to supply the information needed for the type of authentication required.

WebSEAL supplies a default form. Administrators can either use the default form or customize it to fit the local WebSEAL deployment.

The location of the default form is specified in the WebSEAL configuration file:

```
[acct-mgt]
stepup-login = stepuplogin.html
```

Complete the following steps:

1. Specify the name of the authentication strength login form.
   
   To use the default location for the form, verify that the WebSEAL configuration file entry for `stepup-login` contains the default value, `stepuplogin.html`.

2. Optionally, customize the contents of the authentication strength login form.
   
   This file contains macros, in the form of `%TEXT%` sequences, which are replaced with the appropriate values. This substitution occurs within WebSEAL’s template file processing functions and allows the form to be used for the supported authentication methods with correct formatting. It also allows other information, such as error message and authentication method name, to be supplied in the form for the user.
   
   For more information on using macros, see “Macro support for account management pages” on page 89.

The configuration of authentication strength levels is now complete.
Create a protected object policy

Complete the following steps:

1. Create a POP. For example, use `pdadmin` to create a new POP named test:
   
   ```
pdadmin> pop create test
   ```

2. Display the contents of the new POP:
   
   ```
pdadmin> pop show test
   ```

   The new POP contains new settings similar to the following:
   
   ```
pdadmin> pop show test
   Protected object policy: test
   Description: 
   Warning: none 
   Audit level: none 
   Quality of protection: none 
   Time of day access: any time 
   IP Endpoint Authentication Method Policy
   Any Other Network 0
   ```

3. Note the default values in the POP for the attribute IP Endpoint Authentication Method Policy.

   ```
   ... 
   IP Endpoint Authentication Method Policy 
   Any Other Network 0 
   ```

   The IP Endpoint Authentication Method Policy attribute is used to specify two different attributes:
   - Authentication strength level
     
     The default value is 0.
   - Network-based access policy
     
     The default value is Any Other Network.

4. Use `pdadmin pop modify` to modify the IP Endpoint Authentication Method Policy attribute to specify the authentication strength level that you want to apply to the resources identified in “Establish an authentication strength policy” on page 119.

   The syntax is:
   
   ```
pdadmin> pop modify pop-name set ipauth anyothernw level-index
   ```

   The value `level-index` is an integer. The default value is 0. The default value maps to the authentication strength level unauthenticated.

   Specify the index that corresponds to the necessary authentication strength level. To determine the correct `level-index`, examine the `[authentication-level]` stanza in the WebSEAL configuration file. For example:

   ```
   [authentication-levels]
   level = unauthenticated 
   level = password 
   level = certificate 
   ```

   For the above entry, the index values are described in the following table:

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>Index value</th>
</tr>
</thead>
<tbody>
<tr>
<td>unauthenticated</td>
<td>0</td>
</tr>
</tbody>
</table>

Chapter 5. WebSEAL security policy  121
Table 20. Example integer values for authentication strength levels (continued)

<table>
<thead>
<tr>
<th>password</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificate</td>
<td>2</td>
</tr>
</tbody>
</table>

For example, to add the password authentication strength level (index value 1) to the test POP, enter:

```bash
padmin> pop modify test set ipauth anyothernw 1
```

To verify the modification, display the POP:

```bash
padmin> pop show test
Protected object policy: test
Description: Test POP
Warning: no
Audit level: none
Quality of protection: none
Time of day access: sun, mon, tue, wed, thu, fri, sat:
    anytime:local
IP Endpoint Authentication Method Policy
    Any Other Network 1
```

**Note:** In the above example, the only valid index values are: 0,1,2. If any other index value is configured, WebSEAL presents an error page whenever a client requests any object with that has the POP attached.

**Specify network-based access restrictions**

Tivoli Access Manager supports an optional POP configuration setting that enables the application of authentication strength levels to client requests originating from specified network addresses. The network addresses can be defined as either a single IP address, or as a range of IP addresses.

**Note:** In most deployments, user access is not restricted based on the IP address within POPs. In most deployments, this configuration section can be skipped.

The `padmin pop modify set ipauth` command is used to specify IP addresses. Note that this is the same `padmin` command used to specify authentication levels.

The default usage of `padmin pop modify set ipauth` does not impose any network-based access restrictions. This usage consists of specifying the command line argument `anyothernw` as the value for the IP Endpoint Authentication Method Policy attribute. This setting applies to all user access, regardless of the IP address of the requestor, and requires all users to authenticate at the specified level.

The syntax is:

```bash
padmin> pop modify pop-name set ipauth anyothernw level_index
```

For example, in ["Create a protected object policy" on page 121](#) above, the following command created a POP that required all users to authenticate at authentication level 1, and did not impose any network-based access requirements:

```bash
padmin> pop modify test set ipauth anyothernw 1
```

The following network-based access restrictions can be applied:
- Require a specific authentication strength level when the IP address of the requesting client is within a defined range of IP addresses.

Syntax:

```
pdadmin> pop modify pop_name set ipauth add network netmask level_index
```

Note that the **pdadmin pop modify set ipauth add** command specifies both the network addresses and the required authentication level in the IP Endpoint Authentication Method attribute.

For example, to require users from IP address range 9.1.2.[0–255] to use authentication strength level 1:

```
pdadmin> pop modify test set ipauth add 9.1.2.1 255.255.255.0 1
```

Note that the value specified for the netmask determines the range of network addresses affected. The number 0 in the netmask serves as a wildcard to mean all IP addresses for that subnet. See the example that follows.

**Table 21. Using netmask to specify a network range**

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Netmask</th>
<th>Network range affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.2.3</td>
<td>255.255.255.0</td>
<td>9.1.2.[0–255]</td>
</tr>
<tr>
<td>9.1.2.3</td>
<td>255.255.0.0</td>
<td>9.1.[0–255].[0–255]</td>
</tr>
<tr>
<td>9.1.2.3</td>
<td>255.0.0.0</td>
<td>9.[0–255].[0–255].[0–255]</td>
</tr>
</tbody>
</table>

- Require requests from one specific IP address to use a specified authentication strength level.

For example, to require requests from IP address 9.1.2.3 to use authentication strength level 1:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 1
```

To require requests from all IP addresses on subnet 9.1.2.x to use authentication strength level 1:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 1
```

- Disable use of authentication strength level step-up by all requests from a range of network addresses.

The syntax is:

```
pdadmin> pop modify pop_name set ipauth remove network netmask
```

For example, to disable all requests from the range of IP addresses on the 9.1.2.x subnet:

```
pdadmin> pop modify test set ipauth remove 9.1.2.1 255.255.255.0
```

- Allow access to the protected resource based solely on IP address, or range of IP addresses, regardless of the authentication strength level.

This restriction is enforced by specifying the IP address or addresses, and assigning an authentication level of zero (0). For example, to allow requests from IP address 9.1.2.3, regardless of authentication strength level:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 0
```

Likewise, to allow requests from all IP addresses on the 9.1.2.x subnet, regardless of authentication strength level:
padmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 0
- Deny access based solely on IP address, or range of IP addresses, regardless of authentication strength level.

This restriction is enforced by using the key word forbidden as the final parameter.

For example, to restrict only the client at IP address 9.1.2.3 from accessing the protected resource:

padmin> pop modify test set ipauth 9.1.2.3 255.255.255.0 forbidden

Likewise, to restrict requests from all IP addresses on the 9.1.2.x subnet from accessing the resource:

padmin> pop modify test set ipauth 9.1.2.3 255.255.255.0 forbidden

- Prevent requests from all IP addresses from accessing the protected object, unless the IP address has been enabled by a previous pop modify set ipauth add command.

For example, in a use case above, a range of IP addresses were required to access the protected resource by using authentication strength level 1:

padmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 1

The administrator can, in addition, specify that requests from all other IP addresses will be denied, regardless of authentication strength level, in the following padmin command:

padmin> pop modify test set ipauth anyothernw forbidden

The option anyothernw means any other network address, and the option forbidden enforces the denial policy.

**Attach a protected object policy to a protected resource**

After a protected object policy (POP) has been defined and created, it must be attached to the protected resources to which it applies. The syntax for attaching a POP is:

padmin pop attach object_name pop_name

For example, an authentication policy for a WebSEAL deployment could be defined as follows:

- The deployment will use forms authentication and certificate authentication. Forms authentication is the first authentication strength level (1) and certificate authentication is the second (stronger) authentication level (2).
- Users must authenticate using forms authentication or stronger to access the following protected resource (a WebSEAL junction):
  /WebSEAL/hostA/junction
- Users must authenticate using certificate authentication to access the following protected resource (an application):
  /WebSEAL/hostA/junction/applicationA

To implement this policy, the following configuration steps must take place.

1. Modify the WebSEAL configuration file to grant forms authentication an authentication strength of 1 and certificate authentication a strength of 2:

   ```
   [authentication-levels]
   level = unauthenticated
   level = password
   level = ssl
   ```

2. Create a POP for authentication level 1 (forms authentication).
pdadmin> pop create test1
pdadmin> pop modify test1 set ipauth anyothernw 1

3. Create a POP for authentication level 2 (certificate authentication).
   pdadmin> pop create test2
   pdadmin> pop modify test2 set ipauth anyothernw 2

4. Attach the POP test1 to /WebSEAL/hostA/junction.
   pdadmin> pop attach /WebSEAL/hostA/junction test1

5. Attach the POP test2 to /WebSEAL/hostA/junction/application.
   pdadmin> pop attach /WebSEAL/hostA/junction/applicationA test2

**Note:** For more information on the administration of POPs, see the IBM Tivoli Access Manager Base Administration Guide. For information on pdadmin syntax, see the IBM Tivoli Access Manager for e-business Command Reference.

**Enforce user identity match across authentication levels**

Optionally, you can require the user identity that performs the authentication strength (step-up) operation to match the user identity used to perform the initial authentication operation.

When this function is enabled, WebSEAL verifies that the user identity in the new user credential match the user identity in the original credential. If the user identities do not match, WebSEAL denies the authentication step-up, logs an error and returns an error page to the user.

This function is disabled by default. To enable it, edit the WebSEAL configuration file, and set the value of verify-step-up-user to yes:

```
[step-up]
verify-step-up-user = yes
```
Quality of protection POP policy

The quality of protection POP attribute allows you to specify what level of data protection is required when performing an operation on an object.

The quality of protection POP attribute is used to determine whether access will be granted to requested resource. When an ACL check for a resource succeeds, the quality of protection POP is checked. If a quality of protection POP exists, and the resource manager (WebSEAL) cannot guarantee the required level of protection, the request is denied.

```bash
pdadmin> pop modify pop-name set qop {none|integrity|privacy}
```

When QOP level is set to either integrity or privacy, WebSEAL requires data encryption through the use of Secure Socket Layer (SSL).

For example:

```bash
pdadmin> pop modify test set qop privacy
```

Handling unauthenticated users (HTTP / HTTPS)

WebSEAL accepts requests from both authenticated and unauthenticated users over HTTP and HTTPS. WebSEAL then relies on the authorization service to enforce security policy by permitting or denying access to protected resources.

The following conditions apply to unauthenticated users who access WebSEAL over SSL:

- The exchange of information between the unauthenticated user and WebSEAL is encrypted—just as it is with an authenticated user.
- An SSL connection between an unauthenticated user and WebSEAL requires only server-side authentication.

Processing a request from an anonymous client

1. An anonymous client makes a request to WebSEAL (over HTTP or HTTPS).
2. WebSEAL creates an unauthenticated credential for this client.
3. The request proceeds, with this credential, to the protected Web object.
4. The authorization service checks the permissions on the unauthenticated entry of the ACL for this object, and permits or denies the requested operation.
5. Successful access to this object depends on the unauthenticated ACL entry containing at least the read (r) and traverse (T) permissions.
6. If the request fails the authorization decision, the client receives a login form (BA or Forms-based).

Forcing user login

You can force an unauthenticated user to log in by correctly setting the appropriate permissions on the unauthenticated entry in the ACL policy that protects the requested object.

The read (r) and traverse (T) permissions allow unauthenticated access to an object.

To force an unauthenticated user to log in, remove the read (r) permission from the unauthenticated entry in the ACL policy that protects the object. The user receives a login prompt (BA or Forms-based).
Applications of unauthenticated HTTPS

There are many practical business reasons for supporting unauthenticated access to WebSEAL over HTTPS:

- Some applications do not require a personal login, but require sensitive information, such as addresses and credit card numbers. Examples include online purchases of airline tickets and other merchandise.
- Some applications require that you register for an account with the business before you can proceed with further transactions. Again, sensitive information must be passed over the network.
Chapter 6. Authentication

This chapter discusses how WebSEAL maintains session state and handles the authentication process. Successful authentication results in a Tivoli Access Manager identity that represents the user. WebSEAL uses this identity to acquire credentials for that user. Credentials are used by the authorization service to permit or deny access to protected resources.

Topic Index:
- “Overview of the authentication process” on page 130
- “Managing session state” on page 132
- “Authentication configuration overview” on page 140
- “Basic authentication” on page 145
- “Forms authentication” on page 147
- “Client-side certificate authentication” on page 149
- “HTTP header authentication” on page 156
- “IP address authentication” on page 159
- “Token authentication” on page 160
- “Failover authentication” on page 166
- “SPNEGO protocol and Kerberos authentication” on page 182
- “Multiplexing proxy agents” on page 183
Overview of the authentication process

Authentication is the method of identifying an individual process or entity that is attempting to login to a secure domain.

- WebSEAL supports several authentication methods by default and can be customized to use other methods.
- The result of successful authentication to WebSEAL is a Tivoli Access Manager user registry identity.
- WebSEAL uses this identity to obtain a credential for that user.
- The authorization service uses this credential to permit or deny access to protected objects after evaluating the ACL permissions and POP conditions governing the policy for each object.

Note: ACL = access control list policy POP = protected object policy

During authentication, WebSEAL examines a client request for the following information:

- **Session data**
  
  Session data is information that identifies a specific connection between the client and the WebSEAL server. Session data is stored with the client and accompanies subsequent requests by that client. It is used to re-identify the client session to the WebSEAL server and avoid the overhead of establishing a new session for each request.

- **Authentication data**
  
  Authentication data is information from the client that identifies the client to the WebSEAL server. Authentication data types include client-side certificates, passwords, and token codes.

When WebSEAL receives a client request, WebSEAL always looks for session data first, followed by authentication data. The initial client request never contains session data.

**Supported session data types**

WebSEAL supports the following session data types:

- SSL ID (defined by the SSL protocol)
- Server-specific session cookie
- BA header data
- HTTP header data
- IP address

When WebSEAL examines a client request, it searches for session data in the order specified in this list.

**Supported authentication methods**

Although WebSEAL functions independently of the authentication process, WebSEAL uses credentials to monitor all users participating in the secure domain. To obtain the necessary identity information for credentials acquisition, WebSEAL relies on the information gained from the authentication process.

The following table lists the authentication methods supported by WebSEAL for credentials acquisition. When WebSEAL examines a client request, it searches for authentication data in the order specified in this table.
<table>
<thead>
<tr>
<th>Authentication Method</th>
<th>Supported Connection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Failover cookie</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>2. CDSSO ID token</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>3. Client-side certificate</td>
<td>HTTPS</td>
</tr>
<tr>
<td>4. Token passcode</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>5. Forms authentication (username and password)</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>6. SPNEGO (Kerberos)</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>7. Basic authentication (username and password)</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>8. HTTP headers</td>
<td>HTTP and HTTPS</td>
</tr>
<tr>
<td>9. IP address</td>
<td>HTTP and HTTPS</td>
</tr>
</tbody>
</table>

Authentication methods can be independently enabled and disabled for both HTTP and HTTPS transports. If no authentication methods are enabled for a particular transport, the authentication process is inactive for clients using that transport.
Managing session state

A secure connection, or session, between a client and a server requires that the server have the ability to remember—over numerous requests—who it is talking to. The server must have some form of session state information that identifies the client associated with each request.

This section contains the following topics:
- “Session state overview” on page 132
- “GSKit and WebSEAL session cache overview” on page 132
- “Configuring the GSKit SSL session ID cache” on page 133
- “Configuring the WebSEAL session/credentials cache” on page 133
- “Maintaining state with session cookies” on page 135
- “Determining valid session ID data types” on page 138

Session state overview

Without an established session state between client and server, the communication between the client and the server must be renegotiated for each subsequent request. Session state information improves performance by eliminating repeated closing and re-opening of client/server connections. The client can log in once and make numerous requests without performing a separate login for each request.

WebSEAL handles both HTTP and HTTPS communication. HTTP is a “stateless” protocol and does not provide any means of distinguishing one request from another. The SSL transport protocol, on the other hand, is specifically designed to provide a session ID to maintain session state information. HTTP communication can be encapsulated over SSL to become HTTPS.

However, WebSEAL must often handle HTTP communication from unauthenticated clients. And there are times when the SSL session ID is not an appropriate solution. Therefore, WebSEAL is designed to use any of the following information types to maintain session state with a client:
- SSL ID
- Server-specific session cookie
- BA header data
- HTTP header data
- IP address

GSKit and WebSEAL session cache overview

A session cache allows a server to store the session ID information from multiple clients. WebSEAL uses two types of session caches to accommodate both HTTPS and HTTP session state information.
- WebSEAL session/credentials cache
  The WebSEAL session/credentials cache stores any type of session ID information (see the list above) plus the credential information obtained for each client.
  Credential information is cached to eliminate repetitive queries to the user registry database during authorization checks.
- GSKit SSL session ID cache
  The GSKit session cache handles HTTPS (SSL) communication when SSL session ID information is used to maintain session state.
The GSKit cache also maintains session state information for the SSL connection between WebSEAL and the LDAP user registry.

**Configuring the GSKit SSL session ID cache**

The following configuration tasks are available for the GSKit SSL session ID cache:

- Setting the cache entry timeout value
- Setting the maximum concurrent entries value

**Setting the cache entry timeout value**

The parameters for setting the maximum lifetime timeout for an entry in the GSKit SSL session ID cache are located in the [ssl] stanza of the WebSEAL configuration file. There are two parameters: one for SSL V2 connections (ssl-v2-timeout) and one for SSL V3 connections (ssl-v3-timeout).

The default SSL V2 session timeout (in seconds) is 100 (with a possible range of 1-100):

```
[ssl]
ssl-v2-timeout = 100
```

The default SSL V3 session timeout (in seconds) is 7200 (with a possible range of 1-86400):

```
[ssl]
ssl-v3-timeout = 7200
```

**Setting the maximum concurrent entries value**

The ssl-max-entries parameter, located in the [ssl] stanza of the WebSEAL configuration file, sets the maximum number of concurrent entries in the GSKit SSL session ID cache.

This value corresponds to the number of concurrent login sessions. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm to allow new incoming logins.

The default number of concurrent login sessions is 4096:

```
[ssl]
ssl-max-entries = 4096
```

For performance considerations, see the *IBM Tivoli Access Manager for e-business Performance Tuning Guide*.

**Configuring the WebSEAL session/credentials cache**

The following sections describe configuration and use of the WebSEAL session/credentials cache:

- Setting the maximum concurrent entries value
- Setting the cache entry lifetime timeout value
- Setting the cache entry inactivity timeout value
- “Credentials cache limitation” on page 134

**Setting the maximum concurrent entries value**

The max-entries parameter, located in the [session] stanza of the WebSEAL configuration file, sets the maximum number of concurrent entries in the WebSEAL session/credentials cache.
This value corresponds to the number of concurrent login sessions. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm to allow new incoming logins.

The default number of concurrent login sessions is 4096:

```
[session]
max-entries = 4096
```

For performance considerations, see the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

**Setting the cache entry lifetime timeout value**

The `timeout` parameter, located in the `[session]` stanza of the WebSEAL configuration file, sets the maximum lifetime timeout value for all user sessions stored in the WebSEAL session/credentials cache.

WebSEAL caches credential information internally. The session cache timeout parameter dictates the length of time authorization credential information remains in memory on WebSEAL.

The parameter is not an inactivity timeout. The value maps to a "credential lifetime" rather than a "session inactivity timeout". Its purpose is to enhance security by forcing the user to reauthenticate when the specified timeout limit is reached.

The default session cache entry lifetime timeout (in seconds) is 3600:

```
[session]
timeout = 3600
```

**Note:** This parameter is not appropriate for basic authentication (BA). BA supplies authentication data with each request, thereby repeatedly resetting the timeout value.

**Setting the cache entry inactivity timeout value**

The `inactive-timeout` parameter, located in the `[session]` stanza of the WebSEAL configuration file, sets the timeout value for user session inactivity.

The default login session inactivity timeout (in seconds) is 600:

```
[session]
inactive-timeout = 600
```

To disable this timeout feature, set the parameter value to "0".

**Note:** This parameter is not appropriate for basic authentication (BA). BA supplies authentication data with each request, thereby repeatedly resetting the inactive timeout value.

**Credentials cache limitation**

**Limitation:**

When you delete a user from the registry, the credentials of that user in the WebSEAL credentials cache are not removed. If the user has a browser session active at the time the account is deleted, the user can continue to browse, based on the existing credentials in the cache. The credentials of the user are not reevaluated, based on the current information in the user registry, until either a new login occurs or the current credentials expire. The contents of the WebSEAL credentials cache are cleared when the user logs out of the browser session.
Workaround:

As administrator, you can force an immediate halt to user activity in a domain by adding an entry to the default WebSEAL ACL policy for the deleted user with the traverse (T) permission removed. You can also terminate the session manually, using either a command line or using a Tivoli Access Manager administration API function. See [Terminating user sessions](#) on page 345.

Maintaining state with session cookies

One method of maintaining session state between a client and a server is to use a cookie to hold this session information. The server packages the state information for a particular client in a cookie and sends it to the client’s browser. For each new request, the browser re-identifies itself by sending the cookie (with the session information) back to the server.

Session cookies offer a possible solution for situations when the client uses a browser that renegotiates its SSL session after very short periods of time. For example, some versions of the Microsoft Internet Explorer browser renegotiate SSL sessions every two or three minutes.

A session cookie provides reauthentication of a client only to the single, unique server that the client had previously authenticated to within a short time period (around ten minutes). The mechanism is based on a "server cookie" that cannot be passed to any machine other than the one that generated the cookie.

In addition, the session cookie contains only a random number identifier that is used to index the server’s session cache. There is no other information exposed in the session cookie. The session cookie cannot compromise security policy.

Session cookie conditions

WebSEAL uses a secure server-specific session cookie. The following conditions apply to this cookie mechanism:

- Cookie contains session information only; it does not contain identity information
- Cookie resides only in the browser memory (it is not written to the browser cookie jar on the disk)
- Cookie has a limited lifetime
- Cookie has path and domain parameters that prohibit its use by other servers

Session cookies with basic authentication headers

When a client requests access to a protected resource, and WebSEAL is configured to use BA, WebSEAL sends a session cookie to the client.

Clients such as browsers can be configured to either accept or reject cookies. After the client accepts or rejects the cookie, WebSEAL authenticates the user by prompting for username and password.

When a client accepts the session cookie and then successfully logs in, WebSEAL matches, for each additional request by the client, the session ID from the cookie with an existing entry in WebSEAL’s session cache. Thus WebSEAL does not need to reauthenticate the client. This use of the same session ID optimizes server performance.

When the client rejects the session cookie and then successfully logs in, WebSEAL must, for each additional request by the client, establish a new session by
reauthenticating the user. WebSEAL uses BA header information to reauthenticate the user. This reauthentication eliminates possible security vulnerabilities that can occur when BA headers are used without cookies. However, the overhead of reauthenticatoin and session creation lessens server performance.

WebSEAL can be configured to accept BA requests over HTTP or HTTPS (or both). When WebSEAL accepts BA requests over HTTP, WebSEAL always sends a cookie to the client.

When WebSEAL is configured to accept BA requests over HTTPS, administrators can optionally use SSL ids to manage sessions. When administrators choose to not use SSL ids, WebSEAL always sends a cookie to the client.

To use SSL IDs, edit the WebSEAL configuration file as follows:

```plaintext
ssl-id-sessions = yes
```

To not use SSL IDs, set:

```plaintext
ssl-id-sessions = no
```

**Enabling and disabling session ID cookies**

The `ssl-id-sessions` parameter, located in the `[session]` stanza of the WebSEAL configuration file, enables and disables session cookies. This parameter controls whether the SSL session ID is used to maintain the login session for clients accessing over HTTPS. If the parameter is set to "no", session cookies are used for most authentication methods.

```plaintext
[session]
ssl-id-sessions = {yes|no}
```

**Note:** The Opera browser, in its default configuration, does not maintain SSL IDs across SSL connections. When using the Opera browser, `ssl-id-sessions` must be set to `no`.

A configuration setting of `no` for this parameter results in the following conditions for clients accessing over HTTPS:

- The SSL session ID is never used as session ID data.
- Cookies will be used to maintain sessions with clients authenticating with failover cookies, CDSSO ID tokens, forms username and password, token passcode, and client-side certificates.
- For BA clients, a cookie is used to maintain sessions in every configuration except one: If a client connects via HTTPS and `ssl-id-sessions = yes` and `use-same-session = no`, then the SSL ID is used to maintain the session.
- The HTTP header is used as session ID data for clients authenticating with HTTP headers
- The IP address is used as session ID data for clients authenticating with IP addresses.

When `ssl-id-sessions` is set to `yes`, several different values determine the timeout for the session. The session cache entry lifetime timeout is set in the `timeout` entry in the `[session]` stanza, and the session inactivity timeout is set by `inactive-timeout` in the same stanza. SSL timeouts are set in the `[ssl]` stanza, where both `ssl-v2-timeout` and `ssl-v3-timeout` are declared. Thus, when `ssl-id-sessions = yes`, the timeout is set to the *lowest* of the values set for each of the following timeouts:
When you use cookies to maintain session state, the cookie is sent to the browser only once, following a successful login. However, some browsers enforce a limit on the number of in-memory cookies they can store concurrently. In some environments, applications can place a large number of in-memory cookies per domain on client systems. In this case, any configured WebSEAL session cookie or failover cookie can be easily replaced by another cookie.

When you configure WebSEAL to use session cookies (and perhaps failover cookies), you can set the `resend-webseal-cookies` parameter, located in the [session] stanza of the WebSEAL configuration file, to have WebSEAL send the session cookie and the failover cookie to the browser with every response. This action helps to ensure that the session cookie and the failover cookie remain in the browser memory.

The `resend-webseal-cookies` parameter has a default setting of "no":

```
[session]
resend-webseal-cookies = no
```

Change the default setting to "yes" to send WebSEAL session cookies and failover cookies with every response.

**Enabling and disabling same sessions**

You can configure WebSEAL to use the same session ID data when a client logs in over one type of transport (HTTP, for example), disconnects, and re-logs in over another type of transport (HTTPS, for example).

The `use-same-session` parameter, located in the [session] stanza of the WebSEAL configuration file, enables and disables the recognition of the same session ID data. By default, this parameter is set to "no":

```
[session]
use-same-session = no
```

A "yes" configuration setting for this parameter results in the following conditions:

- Session cookies are used to identify the following client types for subsequent logins over another transport:
  - Failover cookies
  - Client-side certificates
  - CDSSO ID token
  - Token passcode
  - Forms username and password
  - basic authentication
- The HTTP header is used for clients accessing with HTTP headers.
- The IP address is used for clients accessing with IP addresses.
- The `ssl-id-sessions` configuration is ignored; the resulting behavior is the same as if `ssl-id-sessions` were set to “no”.

This logic is important because HTTP clients do not have an SSL session ID available as session data.
Because the cookies are available to both HTTP and HTTPS clients, they are not flagged as secure cookies.

**Same session limitation with Netscape 4.7x**

**Problem:**

The same-session feature fails on Netscape Navigator Version 4.7x when requests made to WebSEAL include the port number in the URL, such as: http://webseal:80

**Explanation:**

When WebSEAL is configured for the default HTTP/HTTPS ports, and the port number is not included in the URL, the request succeeds. Requests fail when WebSEAL is configured on non-default ports and the `use-same-session = yes` configuration option is enabled.

Netscape 4.7x does not consider host names with non-standard port numbers to be in the same domain as those with different port numbers. For example, when you access `https://hostname:443`, WebSEAL sets a cookie. When you later access `http://hostname:80`, Netscape does not send the cookie because domain:80 is not the same as domain:443.

**Workaround:**

Upgrade to Netscape Navigator, Version 6.2, or higher.

**Determining valid session ID data types**

The session data type for a client accessing with a particular authentication method is determined by specific combinations of the following configuration parameters:

- Enabling or disabling session cookies (`ssl-id-sessions`)
- Enabling or disabling the ability to use the same session data when a client switches between HTTP and HTTPS (`use-same-session`)

The following tables summarizes the valid session ID data for any given configuration that combines the `ssl-id-sessions` and `use-same-session` parameters:

<table>
<thead>
<tr>
<th>HTTPS Clients</th>
<th>Authentication Method</th>
<th>ssl-id-sessions = yes</th>
<th>ssl-id-sessions = no use-same-session = no</th>
<th>ssl-id-sessions ignored use-same-session = yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failover cookie</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>Certificate</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>CDSSO</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>Token</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td>SSL ID</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>HTTP header</td>
<td>SSL ID</td>
<td>HTTP header</td>
<td>HTTP header</td>
<td></td>
</tr>
<tr>
<td>IP address</td>
<td>SSL ID</td>
<td>IP address</td>
<td>IP address</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP Clients</th>
<th>Authentication Method</th>
<th>use-same-session = no</th>
<th>use-same-session = yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failover cookie</td>
<td></td>
<td>Cookie</td>
<td>Cookie</td>
</tr>
<tr>
<td>Authentication Method</td>
<td>use-same-session = no</td>
<td>use-same-session = yes</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>CDSSO</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>Token</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>BA</td>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
<tr>
<td>HTTP header</td>
<td>HTTP header</td>
<td>HTTP header</td>
<td></td>
</tr>
<tr>
<td>IP address</td>
<td>IP address</td>
<td>IP address</td>
<td></td>
</tr>
</tbody>
</table>
## Authentication configuration overview

You can enable and disable authentication for both HTTP and HTTPS clients on a per-method basis.

The mechanisms for all authentication methods supported by WebSEAL are configured in the [authentication-mechanisms] stanza of the WebSEAL configuration file. Supported authentication method parameters include:

- **Local (built-in) authenticators**
  
  Parameters for local authenticators specify the appropriate built-in shared library (UNIX) or DLL (Windows) files.

- **Custom external authenticators**
  
  WebSEAL provides template server code that you can use to build and specify a custom authentication module.

  An external authentication module specifies the appropriate custom shared library.

### Authentication module parameters

The following parameters specify local built-in authenticators:

*Table 22. Authentication library types specified in the WebSEAL configuration file*

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd-ldap</td>
<td>Library that implements basic authentication and forms authentication with an LDAP user registry.</td>
</tr>
<tr>
<td>cert-ssl</td>
<td>Library that implements certificate authentication.</td>
</tr>
<tr>
<td>token-cdas</td>
<td>Library that implements token authentication.</td>
</tr>
<tr>
<td>http-request</td>
<td>Library that implements HTTP header or IP address authentication.</td>
</tr>
<tr>
<td>sso-create</td>
<td>Library that implements WebSEAL single sign-on token creation.</td>
</tr>
<tr>
<td>sso-consume</td>
<td>Library that implements WebSEAL single sign-on token authentication (consumption).</td>
</tr>
<tr>
<td>passwd-cdas</td>
<td>Library that implements an authentication module library for either basic authentication or forms authentication.</td>
</tr>
<tr>
<td>cred-ext-attrs</td>
<td>Library that implements credential extended attributes authentication.</td>
</tr>
<tr>
<td>su-password</td>
<td>Library that implements switch user authentication for basic authentication or forms authentication.</td>
</tr>
<tr>
<td>su-token-card</td>
<td>Library that implements switch user authentication for token authentication.</td>
</tr>
<tr>
<td>su-certificate</td>
<td>Library that implements switch user authentication for X.509 certificate authentication.</td>
</tr>
<tr>
<td>su-http-request</td>
<td>Library that implements switch user authentication for HTTP header or IP address authentication.</td>
</tr>
<tr>
<td>su-cdsso</td>
<td>Library that implements switch user authentication for cross-domain single sign-on authentication.</td>
</tr>
<tr>
<td>failover-password</td>
<td>Library that implements failover cookie authentication for basic authentication or forms authentication.</td>
</tr>
<tr>
<td>failover-token-card</td>
<td>Library that implements failover cookie authentication for token card authentication.</td>
</tr>
</tbody>
</table>
Table 22. Authentication library types specified in the WebSEAL configuration file (continued)

<table>
<thead>
<tr>
<th>Library Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failover-certificate</td>
<td>Library that implements failover cookie authentication for certificate authentication.</td>
</tr>
<tr>
<td>failover-http-request</td>
<td>Library that implements failover cookie authentication for HTTP header authentication or IP address authentication.</td>
</tr>
<tr>
<td>failover-cdsso</td>
<td>Library that implements failover cookie authentication for cross-domain single sign-on authentication.</td>
</tr>
<tr>
<td>passwd-strength</td>
<td>Library that enforces custom password strength authentication policies.</td>
</tr>
<tr>
<td>cred-ext-attrs</td>
<td>Custom authentication module used to supply extended attribute data to user credential</td>
</tr>
</tbody>
</table>

You use the [authentication-mechanisms] stanza to configure the authentication method and the implementation in the following format:

```
authentication_method_entry = built-in_shared_library
```

**Authentication conversion library**

WebSEAL provides an authentication conversion library that converts authentication data from UTF-8 format to non-UTF-8 format. For Version 5.1, WebSEAL produces authentication data in UTF-8 format. Prior to Version 5.1, WebSEAL produced authentication data in the format of the local code page. Thus, external authentication modules, such as CDASs, that were written for versions of WebSEAL prior to Version 5.1 might need to use the conversion library.

For more information on the conversion library, including configuration instructions see IBM Tivoli Access Manager for e-business Web Security Developer Reference.

For more information on WebSEAL’s use of UTF-8 encoding, see "Multi-locale support with UTF-8" on page 44.

**Default configuration for WebSEAL authentication**

By default, WebSEAL is set to authenticate clients over SSL using basic authentication (BA) user names and passwords (LDAP registry).

WebSEAL is normally enabled for both TCP and SSL access. Therefore, a typical configuration of the [authentication-mechanisms] stanza includes support for username and password (LDAP registry) and support for client-side certificates over SSL.

The following example represents the typical configuration of the [authentication-mechanisms] stanza for Solaris:

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
cert-ssl = libsslauthn.so
```

To configure other authentication methods, add the appropriate parameter with its shared library (or CDAS module).
Configuring multiple authentication methods

You modify the [authentication-mechanism] stanza of the WebSEAL configuration file to specify the shared library to be used for any supported authentication method.

The following conditions apply when you configure multiple authentication methods:

- All authentication methods can function independently from each other. It is possible to configure a shared library for each supported method.
- The cert-cdas method overrides the cert-ssl method when both are configured. You must enable one of these to support client-side certificates.
- Only one password type authenticator is actually used when more than one is configured. WebSEAL uses the following order of priority to resolve multiple configured password authenticators:
  1. passwd-cdas
  2. passwd-ldap
- It is possible to configure the same custom library for two different authentication methods. For example, you could write a custom shared library to process both username/password and HTTP header authentication. For this example, you would configure both the passwd-cdas and http-request parameters with the same shared library. It is the responsibility of the developer to maintain session state and avoid conflicts between the two methods.

Prompting for login

WebSEAL prompts a client for a login under the following conditions:

- An unauthenticated client that fails an authorization check
- A forms or basic authentication client that fails an authorization check
- When an authentication strength (step-up authentication) policy is enforced on the requested resource, and the client has not already authenticated using the required authentication level.

The following client types are presented a "403 failure" error:
- When an authorization check fails:
  – Client-side certificate
  – Failover cookie
  – CDSSO
  – IP address
  – HTTP header
- When a client authenticates with a method that is disabled by WebSEAL

Configuring account expiry notification

WebSEAL returns an error message to a user when a login attempt fails. The error message applies to a variety of situations where the user has supplied invalid authentication information, such as an invalid user name or password.

You can choose to have this same error message returned to a user when the user login fails due to an invalid or expired user account. Alternatively, you can specify that a different error message is sent to the user in this situation. The different error message specifies the exact reason for the failure due to account expiry.
By default, only one error message is returned for all login failures. To specify an account expiry notification message, modify the following setting in the WebSEAL configuration file:

```
[acnt-mgt]
account-expiry-notification = yes
```

The default value is no.

**Logout and change password commands**

Tivoli Access Manager provides the following commands for supporting clients who authenticate over HTTP or HTTPS.

**pkmslogout**

Clients can use the `pkmslogout` command to log out from the current session when they use an authentication method that does not supply authentication data with each request. For example, `pkmslogout` does not work for clients using basic authentication, certificates, or IP address authentication. In this case, you must close the browser to log out.

The `pkmslogout` command is appropriate for authentication using token passcode, forms authentication, and certain implementations of HTTP header authentication.

Run the command as follows:
```
https://www.tivoli.com/pkmslogout
```

The browser displays a logout form defined in the WebSEAL configuration file:

```
[acnt-mgt]
logout = logout.html
```

You can modify the `logout.html` file to suit your requirements.

The `pkmslogout` utility also supports multiple logout response pages when the network architecture requires different exit screens for users logging out of distinctly different back-end systems.

The following expression identifies a specific response file:
```
https://www.tivoli.com/pkmslogout?filename=<custom_logout_file>
```

where `custom_logout_file` is the filename of the logout response. This file must reside in the same `lib/html/C` directory that contains the default `logout.html` file and other sample HTML response forms.

**pkmspasswd**

You can use this command to change your login password when using basic authentication (BA) or forms authentication. This command is appropriate over HTTP or HTTPS.

For example:
```
https://www.tivoli.com/pkmspasswd
```

To assure maximum security when BA is used with WebSEAL, this command has the following behavior for a BA client:
1. The password is changed.
2. The client user is logged out from the current session.
3. When the client makes an additional request, the browser presents the client with a BA prompt.
4. The client must re-log in to continue making requests.

This scenario only applies to a client using basic authentication.

**Post password change processing**
WebSEAL provides support for customized post password change processing. Administrators can write a custom authentication module that will be called after a password is successfully changed through WebSEAL using the pkms password change page, as described in “Logout and change password commands” on page 143. This module can be used to update passwords in an external user registry.

This feature enables passwords in external user registries, which may be unknown to Tivoli Access Manager, to be updated with passwords that the user changed during the course of attempting authentication when challenged by WebSEAL.

For information on implementing a post password change processing module, see *IBM Tivoli Access Manager for e-business Web Security Developer Reference*. 
Basic authentication

Basic authentication (BA) is a standard method for providing a username and password to the authentication mechanism. BA is defined by the HTTP protocol and can be implemented over HTTP and over HTTPS.

By default, WebSEAL is configured for authentication over HTTPS via Basic authentication (BA) username and password.

### Enabling and disabling basic authentication

The `ba-auth` parameter, located in the `[ba]` stanza of the WebSEAL configuration file, enables and disables the basic authentication method.

- To enable the basic authentication method, enter "http", "https", or "both".
- To disable the basic authentication method, enter "none".

For example:

```
[ba]
ba-auth = https
```

### Setting the realm name

The realm name is the text that is displayed in the dialog box that appears when the browser prompts the user for login data. The realm name is also the name of the realm to which the user will be authenticated when the user login succeeds.

The configuration parameter that sets the realm name is located in the `[ba]` stanza of the WebSEAL configuration file.

For example:

```
[ba]
basic-auth-realm = Access Manager
```

The dialog box would display (for example):

Enter username for Access Manager at www.ibm.com:

### Configuring the basic authentication mechanism

The `passwd-ldap` parameter specifies the shared library used to process username and password authentication.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libldapauthn`.
- On Windows, the file that provides the built-in mapping function is a DLL called `ldapauthn`.

#### Table 23. Shared library names for basic authentication

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Shared Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libldapauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libldapauthn.a</td>
</tr>
<tr>
<td>Linux</td>
<td>libldapauthn.so</td>
</tr>
<tr>
<td>HP-UX</td>
<td>libldapauthn.sl</td>
</tr>
<tr>
<td>Windows</td>
<td>ldapauthn.dll</td>
</tr>
</tbody>
</table>
You can configure the username and password authentication mechanism by entering the `passwd-ldap` parameter with the platform-specific name of the shared library file in the `[authentication-mechanism]` stanza of the WebSEAL configuration file. For example:

**Solaris:**
```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

**Windows:**
```
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```

**Configuration conditions**

If forms authentication is enabled for a specific transport, the basic authentication settings for that transport will be ignored.

**Multi-byte UTF-8 logins not supported**

WebSEAL processes login data using UTF-8 encoding. Use of UTF-8 encoding with basic authentication is not supported because the format of browser data for multi-byte BA logins is inconsistent.

Prior to Version 5.1, WebSEAL did not support multi-byte BA logins. For Version 5.1, WebSEAL’s support for handling multi-byte data has been converted by default to use UTF-8 encoding. However, this change does not affect the inconsistent format of browser data. Thus, WebSEAL continues to not support multi-byte BA logins.

**Note:** BA logins are supported for locales using Latin code pages for 8-bit ASCII (ISO-8859-1), such as for French, Spanish, or German. This support is contingent on the server locale also being ISO-8859-1.
Forms authentication

Tivoli Access Manager provides forms authentication as an alternative to the standard basic authentication mechanism. This method produces a custom HTML login form from Tivoli Access Manager instead of the standard login prompt resulting from a basic authentication challenge.

When you use forms-based login, the browser does not cache the username and password information as it does in basic authentication.

Enabling and disabling forms authentication

The **forms-auth** parameter, located in the `[forms]` stanza of the WebSEAL configuration file, enables and disables the forms authentication method.

- To enable the forms Authentication method, enter "http", "https", or "both".
- To disable the forms Authentication method, enter "none".

For example:

```plaintext
[forms]
forms-auth = https
```

Configuring the forms authentication mechanism

The **passwd-ldap** parameter specifies the shared library used to process username and password authentication.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libldapauthn`.
- On Windows, the file that provides the built-in mapping function is a DLL called `ldapauthn`.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Shared Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libldapauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libldapauthn.a</td>
</tr>
<tr>
<td>Linux</td>
<td>libldapauthn.so</td>
</tr>
<tr>
<td>HP-UX</td>
<td>libldapauthn.sl</td>
</tr>
<tr>
<td>Windows</td>
<td>ldapauthn.dll</td>
</tr>
</tbody>
</table>

You can configure the username and password authentication mechanism by entering the **passwd-ldap** parameter with the platform-specific name of the shared library file in the `[authentication-mechanism]` stanza of the WebSEAL configuration file. For example:

**Solaris:**

```plaintext
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

**Windows:**

```plaintext
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```
Configuration conditions

If forms authentication is enabled for a specific transport, the basic authentication settings for that transport will be ignored.

Customizing HTML response forms

Forms authentication requires you to use a custom login form. By default, the sample login.html form is located in the following directory:

```
install_directory/www-webseal_server_instance/locale_dir/lib/html
```

You can customize the content and design of this form.

For detailed information on the available HTML forms you can customize, see “Managing custom account management pages” on page 88.
Client-side certificate authentication

This section contains the following topics:

- “Overview of client-side certificate authentication”
- “Certificate authentication configuration” on page 151

Overview of client-side certificate authentication

Client-side certificate authentication enables a client to use a client-side digital certificate to request an authenticated identity for use within a Tivoli Access Manager secure domain. When authentication is successful, WebSEAL obtains a Tivoli Access Manager identity that is used to build a credential for the user. The credential specifies the permissions and authorities to be granted to the user.

Client-side certificate authentication is disabled by default.

WebSEAL supports client-side certificate authentication in three different modes. The administrator must specify the appropriate mode at configuration time. The following sections describe each mode:

- “Required certificate authentication mode”
- “Optional certificate authentication mode”
- “Delayed certificate authentication mode”

Required certificate authentication mode

WebSEAL always requires a client certificate.

When the user requests access to a resource over SSL, WebSEAL provides its server-certificate, which allows the client to establish an SSL session. WebSEAL then asks the client for a client-side certificate. If the client does not present a certificate, the SSL connection with the client is closed and client-side certificate authentication is not attempted.

Optional certificate authentication mode

WebSEAL requests a client certificate but does not require it.

When the user requests access to a resource over SSL, WebSEAL provides its server-certificate, which allows the client to establish an SSL session. WebSEAL then asks the client for a client-side certificate. If the client presents its client-side certificate, WebSEAL uses it to initiate a certificate-based authentication session. If the client does not present a client-side certificate, WebSEAL allows the SSL session to continue but the user remains unauthenticated to Tivoli Access Manager.

Delayed certificate authentication mode

In this mode, WebSEAL does not request a client certificate for the purpose of client-side certificate authentication until the client attempts to access a protected resource that requires certificate-based authentication.

When the user requests access to a resource over SSL, WebSEAL provides its server-certificate, which allows the client to establish an SSL session. WebSEAL checks the security policy on the requested resource to determine if certificate authentication is required. The security policy is described in the contents of an access control list (ACL) or protected object policy (POP) that has been attached to the protected resource.

If the security policy does not require certificate authentication, WebSEAL does not request a client-side digital certificate.
If the security policy does require certificate authentication, WebSEAL requests the client certificate.

In this mode, the SSL ID cannot be used to track user session activity, because the SSL session will be renegotiated when there is a need to authenticate with certificates. When certificate authentication takes place, WebSEAL adds the SSL ID for the current session into the Certificate SSL ID (renegotiation) cache.

Delayed certificate authentication is used in two scenarios, based on the user’s authentication status at the time that WebSEAL requests the client certificate. In both scenarios, a client can have an unlimited number of exchanges with the WebSEAL server prior to establishing a need to authenticate using certificates.

The two scenarios are:

- **User is unauthenticated**
  In this scenario, the client remains unauthenticated because the client does not attempt to access any resources that require any authentication. When the client eventually attempts to access a resource that requires authentication because of an ACL, WebSEAL presents a certificate login page, and the client can initiate certificate transfer.

  WebSEAL retains the entry in the session cache for the unauthenticated user, but obtains a new SSL ID from GSKit. When the user successfully authenticates, WebSEAL replaces the old unauthenticated user credentials from the session cache data with the new user credentials. The user is now authenticated, and able to request access over HTTPS to resources that require authentication.

- **User has previously authenticated using another authentication method**
  In this scenario, the client was required to authenticate to Tivoli Access Manager during the previous exchanges with WebSEAL. The previous authentication took place through a different authentication method, such as forms authentication or token authentication.

  The client eventually attempts to access a resource that is protected by a protected object policy (POP) that requires client-side certificate authentication in order to access the resource. WebSEAL examines the current WebSEAL authentication strength policy to determine the ranking of the enabled authentication methods. The authentication strength policy ranks the methods in a hierarchy from weakest to strongest.

  When certificate authentication is ranked stronger than the client’s current authentication method, WebSEAL serves to the client a login page with a form for certificate authentication. When the client successfully authenticates using a certificate, the client’s authentication strength level is increased for the duration of the current session. WebSEAL retains the user’s entry in the session cache, but obtains a new SSL ID from GSKit. WebSEAL modifies the user data in the user entry, by replacing the old user credentials (which were based the user’s previous authentication method) with the new user credentials.

  WebSEAL’s authentication strength feature enables a user to move between different authentication levels during a session. Certificate authentication is one of the authentication levels that can be entered when a user needs to increase (step-up) authentication level in order to access protected object resources.

  To enable a user to move up to a certificate authentication level, administrators must modify the WebSEAL configuration file to include certificate authentication in the list of supported levels for authentication strength. For authentication strength configuration instructions, see “Authentication strength policy” on page 117.
Certificate authentication configuration

All of the certificate authentication modes share a common set of configuration tasks. The delayed certificate authentication mode requires additional tasks.

To enable client-side certificate authentication in any of the supported modes, complete the following tasks:

1. “Enable certificate authentication”
2. “Specify the certificate authentication mechanism” on page 152
3. “Specify the certificate login form” on page 152
4. “Specify the certificate login error page” on page 153

When enabling delayed certificate authentication mode, complete the following additional tasks:

1. “Disable SSL session IDs for session tracking” on page 153
2. “Enable and configure the Certificate SSL ID cache” on page 153
3. “Set the timeout for Certificate SSL ID cache” on page 154
4. “Specify an error page for incorrect protocol” on page 154

Note: The WebSEAL server must be stopped and restarted to activate the new configuration settings.

To disable (unconfigure) client-side certificate authentication, complete the following tasks:

- “Disable certificate authentication” on page 155
- “Disable the Certificate SSL ID cache” on page 155

The WebSEAL configuration file settings for certificate authentication are summarized in the following references:

- “Authentication methods” on page 414
- “Authentication libraries” on page 419
- “Account management” on page 443

Enable certificate authentication

Certificate-based authentication is disabled by default. To enable certificate authentication:

1. Edit the WebSEAL configuration file. In the [certificate] stanza, specify a value that instructs WebSEAL how to handle client certificate authentication requests:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept-client-certs = optional</td>
<td>Client can optionally use certificate-based authentication. WebSEAL asks clients for an X.509 certificate. If client supplies a certificate, use certificate-based authentication.</td>
</tr>
<tr>
<td>accept-client-certs = required</td>
<td>Client must use certificate-based authentication. WebSEAL asks clients for an X.509 certificate. If client does not present a certificate, WebSEAL does not allow a connection.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>accept-client-certs = prompt_as_needed</td>
<td>Client is not required to authenticate with a certificate at session start-up. The client can later initiate certificate authentication. This setting enables delayed certificate authentication mode.</td>
</tr>
</tbody>
</table>

For example, to prompt users for a client certificate only when the client encounters a resource that requires certificate authentication, enter:

```
[certificate]
accept-client-certs = prompt_as_needed
```

Note that this setting is used when implementing an authentication strength policy (step-up) for certificate authentication.

2. When accept-client-certs is set to either optional or required, skip this step and continue to the next step.

**Specify the certificate authentication mechanism**

To specify a certificate authentication mechanism, complete the following steps:

1. Verify that certificate authentication is enabled.

   See “Enable certificate authentication” on page 151.

2. Edit the WebSEAL configuration file. In the [certificate] stanza, specify the appropriate certificate authentication built-in shared library as the value for the `cert-ssl` key:

   **Table 25. Certificate authentication shared libraries**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Shared library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libsslauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libsslauthn.a</td>
</tr>
<tr>
<td>HPUX</td>
<td>libsslauthn.sl</td>
</tr>
<tr>
<td>Linux</td>
<td>libsslauthn.so</td>
</tr>
<tr>
<td>Windows</td>
<td>sslauthn.dll</td>
</tr>
</tbody>
</table>

   For example, on a Solaris system:

   ```
   [authentication-mechanisms]
cert-ssl = libsslauthn.so
   ```

   See also:

   - “Authentication methods” on page 414
   - “Authentication libraries” on page 419

**Specify the certificate login form**

WebSEAL provides an HTML page containing a login form, to be presented to users when the need for certificate authentication has been identified.

Administrators can choose to either use the default login form, customize the login form, or specify an entirely different customized login page. Typically, administrators use the default file but customize the contents of the form.

Administrators who choose to create a new HTML file must edit the WebSEAL configuration file to indicate the location of the new file.
The default WebSEAL configuration file entry is:
[acnt-mgt]
certificate-login = certlogin.html

**Specify the certificate login error page**
WebSEAL provides a default HTML page containing an error message that is displayed when a user fails to successfully authenticate using client-side certificate authentication.

Administrators can choose to either use the default error page, customize the error message, or specify an entirely different customized error page. Typically, administrators use the default page but might customize the contents of the error message.

Administrators who choose to create a new HTML error page must edit the WebSEAL configuration file to indicate the location of the new page.

The default WebSEAL configuration file entry is:
[acnt-mgt]
cert-failure = certfailure.html

**Disable SSL session IDs for session tracking**
This configuration step applies only when delayed certificate authentication has been enabled.

Disable the use of SSL session IDs to track session state. Set the following stanza entry in the WebSEAL configuration file:
[ssl]
ssl-id-sessions = no

**Note:** In this case, SSL IDs cannot be used to maintain user sessions because when the user is prompted for a certificate, the user’s SSL ID will change. If ssl-id-sessions is set to yes, WebSEAL generates an error message upon startup and shuts down.

**Enable and configure the Certificate SSL ID cache**
This configuration step applies only when delayed certificate authentication has been enabled.

To configure the cache, complete the following steps:
1. Verify that certificate authentication is enabled. See ["Enable certificate authentication" on page 151]
2. Specify the maximum number of entries allowed in the cache. Edit the WebSEAL configuration file. In the [certificate] stanza, assign a value to
cert-cache-max-entries:
[certificate]
cert-cache-max-entries = 1024

The value corresponds to the maximum number of concurrent certificate authentications. The default value is one quarter of the default number of entries in the SSL ID cache. (Most SSL sessions do not require certificate logins or will only require certificate authentication once for the session). The number of entries in the SSL ID cache is set in the [ssl] stanza:
[ssl]
ssl-max-entries = 4096

Chapter 6. Authentication 153
Thus, the default value for `cert-cache-max-entries` is 1024, which is one quarter of the default value for `ssl-max-entries`, which is 4096.

Note: Most client requests to WebSEAL occur over SSL connections, and all requests over SSL connections without certificates must check the cache. Thus, keeping the cache size smaller can significantly improve performance.

**Set the timeout for Certificate SSL ID cache**

This configuration step applies only when delayed certificate authentication has been enabled.

Complete the following steps:

1. Verify that certificate authentication is enabled.
   See “Enable certificate authentication” on page 151

2. Edit the WebSEAL configuration file. In the `[certificate]` stanza, adjust the value of `cert-cache-timeout` as necessary.

```
[certificate]
cert-cache-timeout = 120
```

The value is the maximum lifetime for an entry in the cache, expressed as a number of seconds.

Use the default value unless your conditions warrant modifying it. Possible reasons to modify the value:
- Systems with memory restrictions may want to reduce the expiration time.
- The expiration time might need to be increased if there is a significant lag between the time when the user initiates a certificate transfer and when the user actually submits the certificate.
- Lower values clean out the cache sooner when no certificate authentications are required. This frees system memory.

**Specify an error page for incorrect protocol**

This configuration step applies only when delayed certificate authentication has been enabled.

WebSEAL provides a default HTML page containing an error message to be displayed when an authenticated user attempts to increase the authentication strength level to client authentication from an HTTP session. Users attempting to increase authentication level to certificate authentication must use the HTTPS protocol.

Administrators can choose to either use the default error page, customize the error message, or specify an entirely different customized error page. Typically, administrators use the default page but might customize the contents of the error message.

Administrators who choose to create a new HTML error page must edit the WebSEAL configuration file to indicate the location of the new page.

The default WebSEAL configuration file entry is:

```
[acnt-mgt]
cert-stepup-http = certstepuphttp.html
```
Disable certificate authentication
To disable certificate authentication:
1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the [certificate] stanza, specify the following key = value pair:
   
   [certificate]
   accept-client-certs = never

3. Restart the WebSEAL server.

Disable the Certificate SSL ID cache
The Certificate SSL ID cache is used only with delayed certificate authentication or authentication strength step-up to certificate authentication.

The disabling of the cache occurs automatically, based on the configuration settings for certificate authentication. To verify that the cache is disabled, examine the value for accept-client-certs in the [certificate] stanza. Verify that the value is one of the following:
- required
- optional
- never

Verify that the value is not prompt_as_needed.
HTTP header authentication

Tivoli Access Manager WebSEAL provides an authentication module that authenticates users based on information obtained from custom HTTP header information supplied by the client or a proxy agent. This module consists of a mapping function that maps header data to a Tivoli Access Manager identity.

WebSEAL trusts that this custom HTTP header data is the result of previous authentication. The WebSEAL authentication module is built specifically to map data obtained from Entrust Proxy headers. When you enable HTTP header authentication using the built-in authentication module, you should disable all other authentication methods. You should accept connections only from the Entrust Proxy. The disabling of other authentication methods eliminates methods that could be used to impersonate custom HTTP header data.

You can optionally customize the HTTP header authentication module to authenticate other types of special header data and, optionally, map this data to a Tivoli Access Manager identity. For information on customizing authentication modules, see the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

Usage notes:
- Session ID cookies are not used to maintain state when ssl-id-sessions = no. The unique header value is used to maintain state.
- When a client encounters an authorization failure, the client receives a “Forbidden” page (HTTP 403).
- Cookie headers cannot be passed to the HTTP header authentication mechanism.

This section contains the following configuration instructions:
- “Enable HTTP header authentication”
- “Specify header types” on page 157
- “Specify the HTTP header authentication mechanism” on page 157
- “Disable HTTP header authentication” on page 158

Enable HTTP header authentication

HTTP header authentication is disabled by default. To enable HTTP header authentication:
1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the [http-headers] stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

<table>
<thead>
<tr>
<th>Protocol to support</th>
<th>Configuration file entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>http-headers-auth = http</td>
</tr>
<tr>
<td>HTTPS</td>
<td>http-headers-auth = https</td>
</tr>
<tr>
<td>Both HTTP and HTTPS</td>
<td>http-headers-auth = both</td>
</tr>
</tbody>
</table>

For example, to support both protocols:
```
[http-headers]
http-headers-auth = both
```
3. Restart the WebSEAL server.

**Specify header types**

The HTTP header types that WebSEAL supports are specified in the [auth-headers] stanza of the WebSEAL configuration file. By default, the built-in shared library is hard-coded to support only Entrust Proxy header data. Thus the only configuration file entry is:

```
[auth-headers]
header = entrust-client
```

You can modify the HTTP header authentication module library to support other header types. To do this, you must replace the existing library with your own implementation of it. For more information on writing your own authentication modules, see *IBM Tivoli Access Manager for e-business Web Security Developer Reference*.

When you write your own authentication module to support other header types, add an entry to the configuration file for each additional supported type:

```
[auth-headers]
header = header_name
```

**Note:** WebSEAL processes only the first header it finds in the user request that matches any one of the values configured in the [auth-headers] stanza. The HTTP header authentication mechanism is not designed to handle more than one HTTP header in a request.

**Specify the HTTP header authentication mechanism**

To specify a HTTP header authentication mechanism, complete the following steps:

1. Verify that HTTP header authentication is enabled.
   
   See “Enable HTTP header authentication” on page 156.

2. Edit the WebSEAL configuration file. In the [authentication-mechanisms] stanza, specify the appropriate HTTP header authentication built-in shared library as the value for the http-request key:

   **Table 27. HTTP header authentication shared libraries**

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Shared library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libhttpauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libhttpauthn.a</td>
</tr>
<tr>
<td>HPUX</td>
<td>libhttpauthn.sl</td>
</tr>
<tr>
<td>Linux</td>
<td>libhttpauthn.so</td>
</tr>
<tr>
<td>Windows</td>
<td>httpauthn.dll</td>
</tr>
</tbody>
</table>

   For example, on a Solaris system:
   
   `[authentication-mechanisms]
   http-request = libhttpauthn.so`

3. By default, authentication information provided within the HTTP header is assumed to be encoded in the local code page. To specify that the HTTP header is encoded in UTF-8, add an option to the authentication mechanism declaration.

   For example, on a Solaris system:
   
   `[authentication-mechanisms]
   http-request = libhttpauthn.so & utf8`
Disable HTTP header authentication

To disable HTTP header authentication:
1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. Set http-headers-auth to none:
   ```
   [http-headers]
   http-headers-auth = none
   ```
3. Restart the WebSEAL server.

Note: HTTP header authentication is disabled by default.
IP address authentication

Tivoli Access Manager supports authentication via an IP address supplied by the client.

Enabling and disabling IP address authentication

The `ipaddr-auth` parameter, located in the `[ipaddr]` stanza of the WebSEAL configuration file, enables and disables the IP address authentication method.  
- To enable the IP address authentication method, enter "http", "https", or "both".  
- To disable the IP address authentication method, enter "none".

For example:
```
[ipaddr]
ipaddr-auth = https
```

Configuring the IP address authentication mechanism

Authentication using an IP address requires a custom shared library. Use the `http-request` parameter to specify this custom shared library.
Token authentication

Tivoli Access Manager supports authentication using a token passcode supplied by the client.

This section contains the following topics:
- “Token authentication concepts”
- “Token authentication configuration” on page 162

Token authentication concepts

This section contains the following topics:
- “Token authentication library”
- “SecurID Token authentication”
- “Authentication workflow for tokens in new PIN mode” on page 161
- “Using token authentication with a password strength server” on page 162
- “RSA SecurID client does not support Linux for zSeries” on page 162

Token authentication library

Two-factor authentication requires users to provide two forms of identification. For example, a single factor of identification, such as a password, plus a second factor in the form of an authentication token. A simple two-factor method — based on something the user knows plus something the user possesses — provides a more reliable level of user authentication than reusable passwords.

Tivoli Access Manager provides a built-in two-factor authentication library, \texttt{xtokenauth}, that is a client of the RSA SecurID token authentication server (ACE/Server) and is written against the RSA Authorization API. WebSEAL provides RSA token authentication client (ACE/Agent) functions, and is certified as SecurID Ready.

By default, this built-in shared library for token authentication is hard-coded to map SecurID (RSA) token passcode data. This default token authentication mechanism expects the user name used by the client to map to an existing user account in the Tivoli Access Manager LDAP registry.

\textbf{Note:} You can also customize this library file to authenticate other types of special token data and, optionally, map this data to a Tivoli Access Manager identity. Refer to the IBM Tivoli Access Manager for e-business Web Security Developer Reference for more information.

SecurID Token authentication

The WebSEAL token authentication process requires the RSA SecurID client, installed and configured on the WebSEAL server machine, to communicate with a remote RSA server. The supported SecurID client version is 5.1.

RSA’s ACE/Servers authenticate several different tokens, including software tokens and hand-held microprocessor-controlled devices. SecurID Software Tokens are binary programs running on a workstation, installed on a smart-card, or running as a plug-in to a Web browser. SecurID Software Tokens can run as an application. The application displays a window into which a user enters a Personal Identification Number (PIN), and the Software Token computes the passcode. The user can then authenticate to WebSEAL by entering the passcode into a login form.
The most typical form of SecurID Token is the hand-held device. The device is usually a key fob or slim card. The token can have a PIN pad, onto which a user enters a PIN, in order to generate a passcode. When the token has no PIN pad, the passcode is created by concatenating the user’s PIN and tokencode. A tokencode is changing number displayed on the key fob. The tokencode is a number generated by the SecurID token at one minute intervals. A user then enters the PIN and tokencode to authenticate to the ACE/Server.

WebSEAL supports both RSA token modes:

- Next tokencode mode
  This mode is used when the user enters the correct PIN but an incorrect tokencode. Typically, the tokencode must be entered incorrectly three times in a row to send the tokencard into next tokencode mode. When the user inputs the correct passcode, the tokencode is automatically changed. The user waits for the new tokencode, and then enters the passcode again.

- New PIN mode
  The token can be in New PIN mode when the old PIN is still assigned. The token is placed in this mode when the administrator wants to enforce a maximum password age policy. The token is also in New PIN mode when the PIN is cleared or has not been assigned. Newly assigned tokens might not yet have a PIN. A PIN can be cleared by an administrator when the user has forgotten it or suspects that it has been compromised.

SecurID PINs can be created in different ways:

- User-defined
- System-generated
- User-selectable

PINs modes are defined by the method of creation, and by rules that specify parameters for password creation and device type.

WebSEAL supports the following types of user-defined PINs:

- 4-8 alphanumeric characters, non-PINPAD token
- 4-8 alphanumeric characters, password
- 5-7 numeric characters, non-PINPAD token
- 5-7 numeric characters, PINPAD token
- 5-7 numeric characters, Deny 4-digit PIN
- 5-7 numeric characters, Deny alphanumeric

WebSEAL does not support the following types of new PINs:

- System-generated, non-PINPAD token
- System-generated, PINPAD token
- User-selectable, non-PINPAD token
- User-selectable, PINPAD token

Token users cannot reset their PIN without an ACE administrator first clearing the token or putting it in new PIN mode. This means users with valid PINs cannot post to pkmspassword.form. Attempts to access this form return an error message.

**Authentication workflow for tokens in new PIN mode**

1. A client (browser) requests a protected Web object requiring token authentication.
2. WebSEAL returns an authentication page, requesting username and passcode.
3. User fills in username and tokencode and submits form to WebSEAL’s authentication library.
   When the user has no PIN, either because the tokencard is new or the administrator reset the PIN, the tokencode is the same as the passcode. When the user has a PIN, but the tokencard is in New PIN mode, the user enters the PIN plus the tokencode.
4. WebSEAL’s token authentication library sends the authentication request to the ACE/Server.
5. The ACE/Server processes the request as follows:
   a. If the authentication is unsuccessful, the result is returned to the WebSEAL token authentication library, which displays an error page to the client (return to step 2).
   b. If the token was not in new PIN mode, the user is authenticated. The WebSEAL token authentication library returns success to the WebSEAL server, which serves the requested protected Web object. (End of authentication workflow).
   c. If the token is in new PIN mode, the ACE/Server returns the NEW_PIN error code to the WebSEAL token authentication library.
6. WebSEAL presents to the user the password expired form.
7. User enters tokencode or passcode and the new PIN and posts it to WebSEAL.
8. WebSEAL checks to see if a password strength server is deployed.
   a. If a password strength server is not deployed, WebSEAL continues to step 9.
   b. If a password strength server is deployed, WebSEAL checks the new PIN. If the PIN is valid, WebSEAL continues to step 9. If the PIN is not valid, WebSEAL returns to step 6.
9. The WebSEAL authentication library sends the tokencode and new PIN to the ACE/Server.
10. The ACE/Server returns a response code.
11. If the PIN set call to the ACE/Server is successful, WebSEAL returns the originally requested protected Web object to the client. If the PIN set call fails, authentication workflow returns to step 6.

**Using token authentication with a password strength server**
WebSEAL also supports a password strength server that is specific to an authentication mechanism. This support enables security architects to develop different password strength policies for different authentication methods while using only WebSEAL authentication mechanisms. A four-digit, numeric PIN, for example, may qualify for the ACE/Server but would fail against a more stringent password strength server.

**RSA SecurID client does not support Linux for zSeries**
The RSA SecurID client is not supported on the Linux for zSeries platforms. This means that the WebSEAL token authentication module cannot support Linux for zSeries platforms.

**Token authentication configuration**
To configure token authentication, the instructions in the following sections must be completed:

- **“Enable token authentication” on page 163**
- **“Specify the token authentication mechanism” on page 163**
Enable token authentication

Token authentication is disabled by default. To enable token authentication:

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the [token] stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

<table>
<thead>
<tr>
<th>Protocol to support</th>
<th>Configuration file entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>token-auth = http</td>
</tr>
<tr>
<td>HTTPS</td>
<td>token-auth = https</td>
</tr>
<tr>
<td>Both HTTP and HTTPS</td>
<td>token-auth = both</td>
</tr>
</tbody>
</table>

   For example, to support both protocols:
   
   ```
   [token]
   token-auth = both
   ```

3. Restart the WebSEAL server.

Specify the token authentication mechanism

To configure a token authentication mechanism, complete the following steps:

1. Stop the WebSEAL server.
2. Ensure that token authentication is enabled.
   
   See "Enable token authentication."

3. Edit the WebSEAL configuration file. In the [token] stanza, enter the appropriate shared library name as the value for the token-cdas key. The shared library names are shown in the following table.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Shared library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libxtokenauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libxtokenauthn.a</td>
</tr>
<tr>
<td>HPUX</td>
<td>libxtokenauthn.sl</td>
</tr>
<tr>
<td>Linux</td>
<td>libxtokenauthn.so</td>
</tr>
<tr>
<td>Windows</td>
<td>xtokenauthn.dll</td>
</tr>
</tbody>
</table>

   For example, on Solaris:
   
   ```
   [authentication-mechanisms]
   token-cdas = libxtokenauthn.so
   ```
4. Restart the WebSEAL server.

See also:

- “Token authentication concepts” on page 160
- “Authentication libraries” on page 419

Enable access to the SecurID client library
For successful communication between the SecurID client and ACE/Server, you must manually set the proper permissions on a SecurID node secret file. You must also set an environment variable that points WebSEAL to the location of the node secret file. The file is called securid. The file is sent upon the first successful authentication between the SecurID client and server. Subsequent RSA client/server communication relies on an exchange of the node secret to verify one another’s authenticity.

To enable WebSEAL to access the SecurID client library, complete the following steps:

1. Change the permissions of the securid and sdconf.rec client configuration files to allow read access by the ivmgr group.
   
   The following example assumes that the location of these files is the /opt/ace/data directory:

   **UNIX:**
   ```
   # cd /opt/ace/data
   # chmod 444 securid
   # chmod 444 sdconf.rec
   ```

   **Windows:**

   Set the Security Properties on the files to "Everyone".

2. Set the VAR_ACE environment variable to inform WebSEAL of the directory location of these two files.
   
   The following example assumes the location of the files is the /opt/ace/data directory:

   **UNIX:**
   ```
   # export VAR_ACE=/opt/ace/data
   ```

   **Windows:**

   Start > Settings > Control Panel > System > Environment

For more information on WebSEAL support for the SecurID client, see “Token authentication concepts” on page 160

Specify a customized password strength library
This configuration entry is required only when a customized password strength library is used.

Edit the WebSEAL configuration file. In the [authentication-mechanisms] stanza, add an entry with the entry keyword passwd-strength. Specify the name of the customized password strength library at the value for the entry.

The following example shows a sample configuration file entry for use of token authentication with a password strength library:
[token]
token-auth = both

[authentication-mechanisms]
token-cdas = libxtokenauthn.so
passwd-strength = libxstrength.so

Enable backwards compatibility for customized token authentication library
In Tivoli Access Manager WebSEAL releases prior to 5.1, WebSEAL did not support the RSA set PIN functions. For these versions, when an authorization check to the ACE/Server returns the New PIN error code, WebSEAL treats it as a failed authentication attempt. If the user attempts to POST a new PIN to the /pkmspasswd page while logged in with token authentication, WebSEAL returns a form with the error message: “Operation not allowed for method: token-card”.

To retain this WebSEAL behavior with release 5.1, users who meet the following conditions will need to add a configuration setting to the WebSEAL configuration file:
• User has deployed a WebSEAL version prior to 5.1 and has enabled token authentication.
• User has used the WebSEAL Authorization Development Kit Password Strength module to develop a customized password strength library
• User does not want to replace the existing customized password strength library with the new xauth_change_password() interface.

To retain backwards compatible behavior with the above software profile, modify the token-cdas entry in the WebSEAL configuration file. For example, on a WebSEAL version prior to 5.1, an example entry on Solaris would be:

[authentication-mechanisms]
token-cdas = /opt/pdweb/lib/libxtokenauthn.so

To retain backwards compatibility on Tivoli Access Manager WebSEAL 5.1, add the NO_NEW_PIN parameter:

[authentication-mechanisms]
token-cdas = /opt/pdweb/lib/libxtokenauthn.so& NO_NEW_PIN

Disable token authentication
To disable token authentication:
1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. Set token-auth to none:
   [token]
   token-auth = none
3. Restart the WebSEAL server.

Note: Token authentication is disabled by default.
Failover authentication

This section contains the following topics:

- “Failover authentication concepts”
- “Failover authentication configuration” on page 173

Failover authentication concepts

WebSEAL provides an authentication method that enables an authenticated session between a client and WebSEAL to be preserved when the WebSEAL server becomes unavailable. The method is called failover authentication. Failover authentication enables the client to connect to another WebSEAL server, and create an authentication session containing the same user session data and user credentials.

This section contains the following topics:

- “Failover authentication scenario”
- “Failover authentication library” on page 167
- “Addition of data to a failover cookie” on page 168
- “Extraction of data from a failover cookie” on page 170
- “Domain-wide failover authentication” on page 171
- “Backwards compatibility” on page 172
- “Upgrading failover authentication” on page 172

Failover authentication scenario

Failover authentication is most commonly used in a scenario where a client (browser) goes through a load balancer to reach a WebSEAL environment. The WebSEAL environment contains two or more replicated WebSEAL servers. The replicated servers are identical. They contain replica copies of the WebSEAL Web protected object space, junction database, and (optionally) dynurl database.

Failover authentication is typically used in a WebSEAL deployment that includes replicated WebSEAL servers. This type of deployment provides two useful features: performance improvements through load balancing, and failover of client sessions between WebSEAL servers.

As part of the failover capability, WebSEAL supports authentication of a user through a failover cookie. The failover cookie is a server-specific cookie or a domain cookie. The failover cookie contains client-specific data, such as user name, cookie-creation time stamp, original authentication method, and an attribute list. The attribute list contains by default the user’s authentication level. WebSEAL can be configured to add specific extended attributes to the attribute list.

WebSEAL encrypts this client-specific data. The replicated WebSEAL servers share a common key that decrypts the cookie information. When the replicated WebSEAL server receives this cookie, it decrypts the cookie, and uses the user name and authentication method to regenerate the client’s credential. WebSEAL can also be configured to copy any extended attributes from the cookie to the user credential. The client can now establish a new session with a replica WebSEAL server without being prompted to log in.

Note: Failover cookies can be used over either HTTP or HTTPS.

The sequence of events for a failover authentication event is:
1. The client (browser) attempts to access a protected resource. The client request goes to a load balancer that controls access to the replicated WebSEAL servers.
2. The load balancer selects a target WebSEAL server and forwards the user request.
3. The client successfully authenticates to WebSEAL using one of the supported authentication methods.
4. WebSEAL creates a failover authentication cookie that contains client authentication information, and sends the cookie to the client.
5. The client sends the cookie through the load balancer to WebSEAL with each subsequent request. The WebSEAL server processes each request.
6. If the load balancer finds that the WebSEAL server is not accessible, the client request is directed to another replicated WebSEAL server.
7. The replicated WebSEAL server is configured to check for the existence of a failover authentication cookie every time it attempts to authenticate a user.
8. WebSEAL uses the information in the cookie to establish a session with the client, without requiring the client to manually log in again. The client’s session data and user credential are built, and the request for the protected resource is processed.
9. The change of session from one WebSEAL server to another WebSEAL server is transparent to the client. Because the WebSEAL servers contain identical resources, the client session continues uninterrupted.

**Note:** For more information on replication of WebSEAL servers, see “Replicated front-end WebSEAL servers” on page 58.

**Failover authentication library**

WebSEAL provides a built-in failover authentication shared library for each of the supported authentication methods. Each failover shared library mimics the shared library for the corresponding authentication method and, additionally, recovers any extended attributes that were originally placed in the user's credential. When a failover authentication event occurs, WebSEAL calls the failover authentication library that matches the last authentication method used by the user before the original WebSEAL server failed.

WebSEAL supplies failover authentication function for the following authentication methods:

- Basic or forms authentication (also known as password authentication)
- Token card authentication
- Certificate authentication
- HTTP request authentication
- Cross-domain single signon (CDSSO)
- Kerberos authentication (SPNEGO)

WebSEAL supplies one standard failover shared library that functions for all the above authentication methods. This library is called libfailoverauthn on UNIX systems, and failoverauthn on Windows.

**Note:** Alternatively, you can supply a custom CDAS library that provides specific authentication capabilities required by your environment.

For example, a WebSEAL server can be configured to support forms authentication and failover authentication. When WebSEAL starts, both the forms authentication shared library and the "failover-forms" authentication library are loaded. The user
authenticates using forms authentication. The WebSEAL server sends a failover authentication cookie to each client (browser). The cookie data specifies that the cookie was created in a forms authentication environment.

When the WebSEAL server becomes unavailable, the failover cookie is sent to a second WebSEAL server. The second WebSEAL server, which is typically a replicated WebSEAL server, also has both the forms authentication shared library and the forms-failover library loaded. The second WebSEAL server receives the failover cookie, and examines it to determine the user’s previous authentication method. The second WebSEAL server calls out to the failover-forms authentication shared library to extract the necessary data from the cookie, and then uses that data to authenticate the user and get a user credential.

For example, when both forms authentication and failover authentication are enabled in a replicated WebSEAL environment, two separate libraries must be configured in the WebSEAL configuration file. One library specifies the forms authentication method library. The other library specifies the failover authentication method library. Example configuration file entries would be:

```
[authentication-mechanisms]
passwd-ldap = /opt/pdweb/lib/libldapauthn.so
failover-password = /opt/pdweb/lib/libfailoverauthn.so
```

In this example, the passwd-ldap stanza entry specifies WebSEAL’s built-in forms authentication library. The failover-password stanza entry specifies WebSEAL’s built-in failover authentication library.

Configuration instructions:

- “Specify the failover authentication library” on page 174
- “Add the authentication level” on page 176

**Addition of data to a failover cookie**

WebSEAL automatically adds specific data from the user session to each failover authentication cookie. WebSEAL can be configured to add additional information from the client data maintained in the credential cache. Also, WebSEAL can be configured to add user-defined data specific to their deployment. For example, user attributes obtained by a custom cross-domain authentication service can be added to the cookie.

By default WebSEAL adds the following data to each cookie:

- **User name**
  
  This name corresponds to the name used to identify the user in the user registry

  **Note:** When an authenticated user has used the WebSEAL switch user function to obtain the effective identity of another user, the identity of the other user is not added to the cookie. Only the original authenticated user identity is added to the cookie.

- **Authentication method**
  
  The authentication method used to authenticate the user to WebSEAL

- **Cookie creation time**
  
  The system time when the cookie was created.

WebSEAL also creates an attribute list containing additional data. By default, the attribute list contains one value:
• **Authentication level**

An integer value that corresponds to the WebSEAL authentication strength level (also an integer value) that is assigned on the local WebSEAL server to the authentication method. Authentication strength, also known as step-up authentication, enables a user to authenticate to a different authenticate method without having to logout.

WebSEAL defines additional user data that can be added to the cookie attribute list:

• **Session lifetime timestamp**

When a user authenticates, WebSEAL tracks the age or lifetime of the user entry in the session cache. The session lifetime timestamp consists of the current time, advanced by the number of seconds configured for the maximum time that a user’s session data can remain in the session cache. When the current system time exceeds the timestamp value, WebSEAL invalidates the user’s entry in the session cache (including the user credentials).

WebSEAL can be configured to add the session lifetime timestamp to the cookie. When this timestamp is added to the cookie, the session lifetime timer can be preserved across failover events. Thus, WebSEAL administrators can choose whether or not to reset the client’s session timer when the client session is established on a replicated server.

Note that successful use of this feature is dependent on synchronization of clocks between replicated WebSEAL servers. If clock skew becomes great, sessions will expire at unintended times.

• **Session inactivity timestamp**

WebSEAL also tracks the amount of time that a user’s entry in WebSEAL’s session cache has been inactive. When a user session is inactive for a period of time longer than the value set for session inactivity, WebSEAL invalidates the user’s session.

The session inactivity timestamp can also be added to the failover authentication cookie. This timestamp differs slightly from the session inactivity timestamp maintained for the WebSEAL session cache. The system inactivity timeout maintained for the cache is calculated by combining two values:

- Current system time
- Maximum number of seconds that a user’s session can remain inactive.

When this value is added to the failover authentication cookie, it is combined with one additional value:

- Maximum number of seconds (interval) between updates to the failover authentication cookie

The setting for the interval between the updating of failover cookies affects performance. Administrators must choose a balance between optimal performance and absolute accuracy of the inactivity timer in the cookie. To keep the inactivity timer most accurate, it should be updated every time the user makes a request. However, frequent updating of cookie contents incurs overhead and decreases performance.

Each administrator must choose an interval that best fits the WebSEAL deployment. In some cases, an update of the failover cookie with every user request is appropriate. In other cases, the administrator might choose to never update the inactivity timer in the failover cookie.

• **Additional extended attributes**
Administrators can configure WebSEAL to insert a customized set of attributes into a failover cookie. Attributes can be specified individually or in a group. To specify a group of attributes, use wildcard pattern matching in configuration file entries. This feature is useful in deployments that also use customized authentication libraries, such as cross-domain authentication servers, to insert special attributes into a user credential. By specifying those attributes in the WebSEAL configuration file, the administrator can ensure that the attributes are available to add to the re-created user credential during failover authentication.

**Note:** The maximum size of a failover authentication cookie is 4 kilobytes (4096 bytes)

**Configuration instructions:**
- “Add the authentication level” on page 176
- “Add the session lifetime timestamp” on page 176
- “Add the session activity timestamp” on page 176
- “Add an interval for updating the activity timestamp” on page 177
- “Add extended attributes” on page 177

**Extraction of data from a failover cookie**
When a failover authentication event occurs, WebSEAL receives a failover authentication cookie and by default extracts the following data from each cookie:

- User name
- Authentication method
- Cookie creation time

WebSEAL first determines if the cookie is valid by subtracting the cookie creation time from the system time, and comparing this value against the WebSEAL configuration file entry for failover cookie lifetime.

If the cookie lifetime has been exceeded, the cookie is not valid, and failover authentication is not attempted. If the cookie lifetime has not been exceeded, WebSEAL uses the user name and authentication method to authenticate the user and build a user credential.

WebSEAL next checks configuration settings to determine if additional cookie data should be extracted and evaluated. Note that the WebSEAL server does not by default extract any other attributes from the failover authentication cookie. Each additional attribute to be extracted must be specified in the WebSEAL configuration file. Wildcard pattern matching can be used to obtain groups of attributes.

WebSEAL can be configured to extract the following defined attributes:

- **Authentication level**
  When this value is extracted, WebSEAL uses it to ensure that the user is authenticated with the authentication method necessary to maintain the specified authentication level.
  Note that WebSEAL can obtain authentication levels from several different places:
  - Failover cookie
  - Failover authentication library
  - Cross-domain authentication service
– Entitlements service

The authentication level extracted from the failover cookie takes precedence over levels obtained from the other places.

• Session lifetime timestamp
WebSEAL can use this timestamp to determine if the user’s entry in the original server’s session cache would have expired. If it would have, WebSEAL discards the cookie and all its potential credential attributes. The session lifetime is not preserved, and the user is prompted to log in.

• Session inactivity timestamp
WebSEAL can use this timestamp to determine if the user’s entry in the original server’s session cache would have been inactive for too long. If it would have, WebSEAL discards the cookie and all its potential credential attributes. The session lifetime is not preserved, and the user is prompted to log in.

Note: Successful use of these timestamps requires synchronization of clocks between replicated WebSEAL servers. If clock skew becomes great, sessions will expire or become inactive at unintended times.

• Additional extended attributes
These include user-defined customized attributes, such as those generated by cross-domain authentication services. WebSEAL adds the attributes to the user credential.

Attributes that are not specified in the WebSEAL configuration file will be ignored and not extracted. In addition, administrators can specify that certain attributes must be ignored during failover cookie extraction. Although ignore is the default behavior, this specification can be useful, for example, to ensure that user attributes are obtained from the user registry instead of from the failover cookie.

Configuration instructions:
• “Extraction of data from a failover cookie” on page 170

Domain-wide failover authentication
WebSEAL supports an optional configuration that enables failover authentication cookies to be marked as available for use during failover authentication to any and all other WebSEAL servers in the Tivoli Access Manager domain. This configuration option enables failover authentication cookies to be used in deployments that do not necessarily have a load balancer and replicated WebSEAL servers.

When a client session goes through a failover authentication event to a replicated WebSEAL server, the client continues to access the same set of protected resources. When a client session goes through a failover authentication event to a WebSEAL server that is not replicated, it is possible that a different set of resources will be available to the client. In large deployments, this partitioning of resources within the Tivoli Access Manager domain is common. This partitioning can be done for performance reasons and for administrative purposes.

Domain-wide failover authentication can be used to redirect a client to another WebSEAL server at a time when the client’s requests have led it to request a resource that is not available through the local WebSEAL server. In this case, the client (browser) is redirected to another WebSEAL server. The receiving WebSEAL server can be configured to look for failover authentication cookies. The WebSEAL server attempts to authenticate the client and recognizes the failover authentication
cookie. By using the cookie, the WebSEAL server does not need to prompt the client for login information, but instead can establish a session with the client and construct a valid set of user credentials.

Configuration instructions:
- "Enable domain-wide failover cookies” on page 179

Backwards compatibility
Failover cookies generated by Version 5.1 WebSEAL servers can be understood and read (consumed) by WebSEAL servers from versions prior to Version 5.1. Likewise, failover cookies generated by older (pre-Version 5.1) WebSEAL servers can be understood and read (consumed) by Version 5.1 WebSEAL servers. CDAS modules written to customize failover cookies for older (pre-Version 5.1) WebSEAL servers will work with Version 5.1 WebSEAL servers.

To ensure complete backwards compatibility, the following features are provided:
- WebSEAL can be configured to authenticate a user based on failover cookie contents when the session lifetime timestamp is not present.
  The session lifetime timestamp is not present in failover authentication cookies prior to Version 5.1.
- WebSEAL can be configured to authenticate a user based on failover cookie contents when the session inactivity timestamp is not present.
  The session inactivity timestamp is not present in failover authentication cookies prior to Version 5.1.
- The algorithm used to encrypt client data in failover authentication cookies was updated for Version 4.1 of WebSEAL. When using WebSEAL servers with versions of WebSEAL prior to Version 4.1, a configuration file setting can be set to enable access to the older-style cookies.
- WebSEAL can be configured to not use UTF-8 encoding on strings in the failover cookie. By not using UTF-8 encoding for cookies created on Version 5.1 WebSEAL servers, the cookies can be understood and read (consumed) by older (pre-Version 5.1) WebSEAL servers.

Configuration instructions:
- "Require validation of a lifetime timestamp” on page 179
- "Require validation of an activity timestamp” on page 180
- "Enable backwards compatibility for encryption prior to Version 4.1” on page 180
- “Specify UTF-8 encoding on cookie strings” on page 175

Upgrading failover authentication
In the WebSEAL configuration file, the [failover-add-attributes] and [failover-restore-attributes] stanzas replace the pre-Version 5.1 stanza [failover-attributes].

During an upgrade from Tivoli Access Manager Version 4.1 to the current Tivoli Access Manager version, the stanza [failover-attributes] and its contents are migrated to the [failover-add-attributes] stanza.

The upgrade is automated, and takes place when WebSEAL is installed. There is no need for manual updating of these entries.
Failover authentication configuration

This section describes how to configure failover authentication.

If you are not familiar with failover authentication concepts, review “Failover authentication concepts” on page 166.

To configure failover authentication, complete the following tasks:

1. Stop the WebSEAL server.

2. To enable failover authentication, complete each of the following tasks:
   a. “Specify the protocol for failover cookie” on page 174
   b. “Specify the failover authentication library” on page 174
   c. “Create an encryption key for cookie data” on page 175
   d. “Specify the cookie lifetime” on page 175
   e. “Specify UTF-8 encoding on cookie strings” on page 175

3. Optionally, you can configure WebSEAL to maintain session state across failover authentication sessions. If this is appropriate for your deployment, complete the following instructions:
   a. “Add the session lifetime timestamp” on page 176
   b. “Add the session activity timestamp” on page 176
   c. “Add an interval for updating the activity timestamp” on page 177

4. Optionally, you can configure WebSEAL to add extended attributes or authentication level to the failover cookie:
   - “Add extended attributes” on page 177
   - “Specify the authentication level attribute after failover authentication” on page 178

5. When you have configured WebSEAL to add attributes to the failover cookie, you must configure WebSEAL to extract the attributes when reading the cookie:
   - “Specify attributes for extraction” on page 178

6. Optionally, you can enable failover authentication cookies for use on any WebSEAL server within the domain. If this is appropriate for your deployment, see
   - “Enable domain-wide failover cookies” on page 179

7. If you need to maintain backwards compatibility with failover authentication cookies generated by WebSEAL servers from versions prior to Version 5.1, complete the following instructions:
   a. “Specify UTF-8 encoding on cookie strings” on page 175
   b. “Require validation of a lifetime timestamp” on page 179
   c. “Require validation of an activity timestamp” on page 180
   d. “Enable backwards compatibility for encryption prior to Version 4.1” on page 180
   e. “Enable backwards compatibility for Version 4.1 cookies” on page 181

8. After completing all the instructions applicable to your deployment, restart the WebSEAL server.

WebSEAL configuration file references:
   - “Authentication failover” on page 424
   - “Authentication methods” on page 414
   - “Authentication libraries” on page 419
**Specify the protocol for failover cookie**

Failover authentication cookies are disabled by default. To enable failover cookies, edit the WebSEAL configuration file:

In the [failover] stanza, specify a value that instructs WebSEAL how to handle client certificate authentication requests. The following table shows the valid values.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failover-auth = http</td>
<td>Failover cookies enabled over HTTP protocol.</td>
</tr>
<tr>
<td>failover-auth = https</td>
<td>Failover cookies enabled over HTTPS (SSL) protocol.</td>
</tr>
<tr>
<td>failover-auth = both</td>
<td>Failover cookies enabled over both HTTP and HTTPS (SSL) protocol.</td>
</tr>
</tbody>
</table>

**Note:** Enabling failover authentication to either HTTP or HTTPS causes cookies to be written to clients connecting over all protocols. The value specified in the failover-auth stanza entry dictates the protocol over which cookies will be accepted for authentication during a failover authentication event.

**Specify the failover authentication library**

Edit the WebSEAL configuration file. In the [authentication-mechanisms] stanza, uncomment the entry for the authentication type (or types) that must support failover cookies. Add the name of the WebSEAL failover cookie library appropriate for the operating system type.

The default configuration file entry is:

```
[authentication-mechanisms]
#failover-password = failover_password_library_filename
#failover-token-card = failover_token_card_filename
#failover-certificate = failover_certificate_filename
#failover-http-request = failover_http_request_filename
#failover-cdsso = failover_cdsso_filename
#failover-kerberosv5 = failover_kerberos_library
```

WebSEAL supplies one standard failover shared library that functions for all the above authentication methods. Refer to the following table for the library names.

<table>
<thead>
<tr>
<th>System</th>
<th>Library Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>libfailoverauthn.so</td>
</tr>
<tr>
<td>Linux</td>
<td>libfailoverauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libfailoverauthn.a</td>
</tr>
<tr>
<td>HP-UX</td>
<td>libfailoverauthn.sl</td>
</tr>
<tr>
<td>Windows</td>
<td>failoverauthn.dll</td>
</tr>
</tbody>
</table>

For example, to enable failover authentication for clients who originally authenticated with forms authentication on Solaris, uncomment the failover-password entry and add the library name:

```
[authentication-mechanisms]
failover-password = libfailoverauthn.so
```

Alternatively, when you have developed a CDAS library that implements a customized version of failover authentication for one or more authentication
methods, insert the name of the custom CDAS as the value for the configuration file keyword. For example, if you developed a custom CDAS for forms authentication, enter the absolute path name:

```
[authentication-mechanisms]
failover-password = /dir_name/custom_cdas_failover_library.so
```

Create an encryption key for cookie data

Use the `cdsso_key_gen` utility to secure the cookie data. WebSEAL provides this utility. This utility generates a symmetric key that encrypts and decrypts the data in the cookie.

**Attention:** If you do not configure WebSEAL to encrypt failover authentication cookies, and you have enabled failover authentication, WebSEAL will generate an error and refuse to start. Failover authentication cookies must be encrypted.

1. Run the utility on one of the replicated servers. From a command line, specify the location of the key file you want to create. You must specify an absolute path name.
   For example:
   - UNIX:
     ```
     /opt/pdweb/bin/cdsso_key_gen absolute_pathname_for_keyfile
     ```
   - Windows:
     ```
     C:\Program Files\Tivoli\PDWeb\bin\cdsso_key_gen absolute_pathname_for_keyfile
     ```
   You can give the key file any appropriate name, such as `/opt/pdweb/lib/ws.key`.

2. Edit the WebSEAL configuration file. In the `[failover]` stanza, specify the keyfile location.
   ```
   [failover]
   failover-cookies-keyfile = absolute_pathname_for_keyfile
   ```

3. Manually copy the key file to each of the remaining replicated servers.

4. On each replicated server, edit the WebSEAL configuration file to supply the correct path name to `failover-cookies-keyfile` in the `[failover]` stanza.

Specify the cookie lifetime

Edit the WebSEAL configuration file. Specify the valid lifetime for the failover cookie.

```
[failover]
failover-cookie-lifetime = 60
```

The default lifetime is 60 minutes.

Specify UTF-8 encoding on cookie strings

Edit the WebSEAL configuration file. Specify whether or not WebSEAL should use UTF-8 encoding on strings within the failover cookies.

```
[failover]
use-utf8 = yes
```

The default value is yes.

UTF-8 should be used when user names or credential attributes in the cookie are not encoded in the same code page as the one that the WebSEAL server is using. By default, WebSEAL servers use UTF-8 encoding. When all WebSEAL servers in the WebSEAL deployment use UTF-8 encoding, leave this value at the default setting of `yes`. 
Backwards compatibility

WebSEAL servers prior to Version 5.1 did not use UTF-8 encoding. Thus, cookies created by these servers do not have UTF-8 encoding on their strings. When a WebSEAL server is operating with WebSEAL servers from versions prior to Version 5.1, WebSEAL should not use UTF-8 encoding.

For backwards compatibility, set use-utf8 to no.

```
[failover]
use-utf8 = no
```

For more information on WebSEAL support for UTF-8 encoding, see “Multi-locale support with UTF-8” on page 44.

Add the authentication level

To specify authentication level in the failover authentication cookie, add the authentication level to the WebSEAL configuration file. You must use the stanza entry keyword AUTHENTICATON_LEVEL:

```
[failover-add-attributes]
AUTHENTICATION_LEVEL = add
```

The actual value for AUTHENTICATION_LEVEL is an integer that WebSEAL tracks internally. You do not need to specify the integer in this stanza.

Add the session lifetime timestamp

WebSEAL calculates the session lifetime timestamp by combining the following values:

- Current system time.
- Maximum lifetime in seconds that an entry is allowed to exist in the WebSEAL credential cache.

This maximum lifetime in seconds is specified in the WebSEAL configuration file [session] stanza:

```
[session]
timeout = 3600
```

To add this value to the failover authentication cookie, add the following entry to the WebSEAL configuration file:

```
[failover-add-attributes]
session-lifetime-timestamp = add
```

Note that this attribute cannot be set by wildcard matching. The exact entry session-lifetime-timestamp must be entered.

Add the session activity timestamp

WebSEAL calculates the session activity timestamp by adding together these values:

- System time.
- Maximum lifetime of inactive entries in the credential cache.

The maximum lifetime for inactive entries is set in the [session] stanza in the WebSEAL configuration file:

```
[session]
inactive-timeout = 600
```

The default value is 600 seconds.

- Interval for updating the failover authentication cookie.
This value is set in the [failover] stanza in the WebSEAL configuration file:

```
[failover]
failover-update-cookie = 60
```

The default value is 60 seconds. For more information, see “Add an interval for updating the activity timestamp.”

To add this value to the failover authentication cookie, add the following entry to the WebSEAL configuration file:

```
[failover-add-attributes]
session-activity-timestamp = add
```

Note that this attribute cannot be set by wildcard matching. The exact entry `session-activity-timestamp` must be entered.

**Note:** When you set `failover-update-cookie` to a number greater than zero, ensure that you also set `session-activity-timestamp` = `add`. If you do not set `session-activity-timestamp` = `add`, WebSEAL will decode the failover cookie on each user access. This could adversely affect performance.

**Add an interval for updating the activity timestamp**

Optionally, the session activity timestamp in the failover cookie can be updated during the user’s session.

This entry contains an integer value for interval (in seconds) between updating the failover cookie’s activity timestamp.

The default entry is:

```
[failover]
failover-update-cookie = 60
```

When `failover-update-cookie` is set to 0, the last activity timestamp is updated with each request.

When `failover-update-cookie` is set to an integer less than 0 (any negative number), the last activity timestamp is never updated.

When `failover-update-cookie` is set to an integer greater than 0, the session activity timestamp in the cookie is updated at intervals of this number of seconds.

The value chosen for this stanza entry can affect performance. See “Addition of data to a failover cookie” on page 168.

**Note:** When you set `failover-update-cookie` to a number greater than zero, ensure that you also set `session-activity-timestamp` = `add`. If you do not set `session-activity-timestamp` = `add`, WebSEAL will decode the failover cookie on each user access. This could adversely affect performance. See “Add the session activity timestamp” on page 176.

**Add extended attributes**

WebSEAL can optionally be configured to place a copy of specified extended attributes from a user credential into a failover authentication cookie. No extended attributes are configured by default.

To add extended attributes, add entries to the [failover-add-attributes] stanza in the WebSEAL configuration file. The syntax is:
The *attribute_pattern* can be either a specific attribute name, or a case-insensitive wildcard expression that matches more than one attribute name. For example, to specify all attributes with the prefix *tagvalue_* add the following entry:

```plaintext
[failover-add-attributes]
tagvalue_* = add
```

The order of the stanza entries is important. Rules that appear earlier in [failover-add-attributes] take priority over those placed later in the stanza.

Attributes that do not match any of the wildcard patterns, or are not explicitly specified, are not added to the failover cookie.

**Specify the authentication level attribute after failover authentication**

When a failover cookie is used to authenticate, an authentication level can be associated with the generated credential. This can be done in the following ways:

- **Using a flag to the failover-* authentication mechanism.**
  
  You can set the authentication level based on the last authentication method used. This is done by passing a flag to the authentication method. The syntax is:

  ```plaintext
  [authentication-mechanisms]
  failover-method_name = WebSEAL_failover_lib -i authentication_level
  ```

  For example:

  ```plaintext
  failover-password = /opt/pdweb/liblibfailoverauth.so& -i 1
  failover-token = /opt/pdweb/liblibfailoverauth.so& -i 2
  failover-certificate = /opt/pdweb/liblibfailoverauth.so& -i 3
  ```

  **Note:** This method can be used only with the built-in WebSEAL failover authentication library. This method is not available to custom authentication modules.

- **Setting it in the [failover-restore-attributes] configuration file stanza entry.**
  
  This attribute can specify whether to use the authentication level from the failover cookie, if it exists:

  ```plaintext
  [failover-restore-attributes]
  AUTHENTICATION_LEVEL = preserve
  ```

**Specify attributes for extraction**

WebSEAL can optionally be configured to extract attributes from a failover authentication cookie and place them into a user credential. No attributes are configured for extraction by default.

Attributes to be extracted are declared in the [failover-restore-attributes] stanza in the WebSEAL configuration file. The syntax is:

```plaintext
[failover-restore-attributes]
attribute_pattern = {preserve|refresh}
```

The value `preserve` tells WebSEAL to extract the attribute and add it to the credential.

The value `refresh` tells WebSEAL to ignore the attribute, and not extract it from the cookie.
The `attribute_pattern` can be either a specific attribute name, or a case-insensitive wildcard expression that matches more than one attribute name. For example, to extract all attributes with the prefix `tagvalue_`, add the following entry:

```
[failover-restore-attributes]
tagvalue_* = preserve
```

Attributes that do not match any patterns specified with the `preserve` value are not extracted from the failover authentication cookie.

The order of the stanza entries is important. Rules that appear earlier in `[failover-restore-attributes]` take priority over those placed later in the stanza.

The following attributes cannot be matched by a wildcard pattern, but must be explicitly defined for extraction:

- Authentication level
  ```
  [failover-restore-attributes]
  AUTHENTICATION_LEVEL = preserve
  ```

- Session lifetime timestamp
  ```
  [failover-restore-attributes]
  session-lifetime-timestamp = preserve
  ```

- Session inactivity timestamp
  ```
  [failover-restore-attributes]
  session-inactivity-timestamp = preserve
  ```

**Enable domain-wide failover cookies**

You can allow a failover authentication cookie to be used by any WebSEAL server within the same domain as the WebSEAL server that creates the cookie. This feature is controlled by a stanza entry in the `[failover]` stanza.

By default, domain-wide failover cookie functionality is disabled:

```
[failover]
enable-failover-cookie-for-domain = no
```

To enable this feature, set `enable-failover-cookie-for-domain` to `yes`:

```
[failover]
enable-failover-cookie-for-domain = yes
```

For information on the effects of enabling this stanza entry, see “Domain-wide failover authentication” on page 171.

**Require validation of a lifetime timestamp**

WebSEAL servers can optionally be configured to `require` that each failover authentication cookie contain a session lifetime timestamp. The session lifetime timestamp is not required by default. The default configuration file entry is:

```
[failover]
failover-require-lifetime-timestamp-validation = no
```

This stanza entry is used primarily for backwards compatibility.

**Attention:** For backwards compatibility with failover cookies created by WebSEAL servers prior to Version 5.1, set this entry to `no`. Failover authentication cookies created by WebSEAL servers prior to Version 5.1 do not contain this timestamp.
• When this value is no, and the session lifetime timestamp is missing from the failover cookie, the receiving server will view the cookie as valid.
• When this value is yes, and the session lifetime timestamp is missing from the failover cookie, the receiving server will view the cookie as not valid.
• When this value is either no or yes, and the session lifetime timestamp is present in the failover cookie, the receiving server evaluates the timestamp. If the timestamp is not valid, the authentication fails. If the timestamp is valid, the authentication process proceeds.

Note: The session lifetime timestamp is configured separately from the session activity timestamp.

Require validation of an activity timestamp
WebSEAL servers can optionally be configured to require that each failover authentication cookie contain a session activity timestamp. The session activity timestamp is not required by default. The default configuration file entry is:

```
[failover]
failover-require-activity-timestamp-validation = no
```

This stanza entry is used primarily for backwards compatibility.

Attention: For backwards compatibility with failover cookies created by WebSEAL servers prior to Version 5.1, set this entry to no. Failover authentication cookies created by WebSEAL servers prior to Version 5.1 do not contain this timestamp.
• When this value is no, and the session activity timestamp is missing from the failover cookie, the receiving server will view the cookie as valid.
• When this value is yes, and the session activity timestamp is missing from the failover cookie, the receiving server will view the cookie as not valid.
• When this value is either no or yes, and the session activity timestamp is present in the failover cookie, the receiving server evaluates the timestamp. If the timestamp is not valid, the authentication fails. If the timestamp is valid, the authentication process proceeds.

Note: The session activity timestamp is configured separately from the session lifetime timestamp.

Enable backwards compatibility for encryption prior to Version 4.1
For Tivoli Access Manager Version 4.1, the level of security for the encryption of the failover authentication cookie was increased. This encryption algorithm is not backward compatible. If you are integrating failover authentication cookies with WebSEAL servers using versions of Tivoli Access Manager prior to Version 4.1, you must specify a configuration file setting in the WebSEAL configuration file to enable backwards compatibility.

Backwards compatibility with the older encryption algorithm is not enabled by default:

```
[server]
pre-410-compatible-tokens = no
```

To enable backwards compatibility, set pre-410-compatible-tokens to yes:

```
[server]
pre-410-compatible-tokens = yes
```
Enable backwards compatibility for Version 4.1 cookies
For Tivoli Access Manager Version 5.1, the format of the encryption of the failover authentication cookie was changed. This encryption algorithm is not backward compatible. If you are integrating failover authentication cookies with Version 4.1 WebSEAL servers, you must specify a configuration file setting in the WebSEAL configuration file to enable backwards compatibility.

Backwards compatibility with the older encryption format is not enabled by default:

```
[server]
pre-510-compatible-tokens = no
```

To enable backwards compatibility, set `pre-510-compatible-tokens` to `yes`:

```
[server]
pre-510-compatible-tokens = yes
```

**Note:** To enable backwards compatibility with WebSEAL servers prior to Version 4.1, you must set an additional parameter. See “Enable backwards compatibility for encryption prior to Version 4.1” on page 180.
SPNEGO protocol and Kerberos authentication

WebSEAL supports the SPNEGO protocol and Kerberos authentication for use with Windows clients to achieve Windows desktop single sign-on. The SPNEGO protocol allows for a negotiation between the client (browser) and the server regarding the authentication mechanism to use. The client identity presented by the browser can be verified by WebSEAL using Kerberos authentication mechanisms.

WebSEAL’s support for Kerberos authentication has been implemented specifically to support a Windows desktop single sign-on solution. This solution requires that the WebSEAL server be configured into an Active Directory domain, and that WebSEAL be able to access a Kerberos Key Distribution Center. In addition, the Internet Explorer client must be configured to use the SPNEGO protocol and Kerberos authentication when contacting WebSEAL.

The configuration steps for this solution combine a series of Windows-specific instructions with WebSEAL server configuration instructions.

WebSEAL’s support for the Windows desktop single sign-on, including the necessary configuration steps, are described in “Windows desktop single sign-on” on page 234.
Multiplexing proxy agents

Tivoli Access Manager provides solutions for securing networks that use a Multiplexing Proxy Agent (MPA).

Standard Proxy Agents (SPA) are gateways that support per-client sessions between clients and the origin server over SSL or HTTP. WebSEAL can apply normal SSL or HTTP authentication to these per-client sessions.

Multiplexing Proxy Agents (MPA) are gateways that accommodate multiple client access. These gateways are sometimes known as WAP gateways when clients access using Wireless Access Protocol (WAP). Gateways establish a single authenticated channel to the origin server and “tunnel” all client requests and responses through this channel.

To WebSEAL, the information across this channel initially appears as multiple requests from one client. WebSEAL must distinguish between the authentication of the MPA server and the additional authentication of each individual client.

![Diagram of Communication over an MPA Gateway]

Figure 5. Communication over an MPA Gateway

Because WebSEAL maintains an authenticated session for the MPA, it must simultaneously maintain separate sessions for each client. Therefore, the session data and authentication method used for the MPA must be distinct (different) from the session data and authentication method used by the client.

Valid session data types and authentication methods

The session data type used by the MPA to WebSEAL must be distinct (different) from the session data type used by the client to WebSEAL. The table below lists the valid session types for the MPA and the client:

<table>
<thead>
<tr>
<th>Valid Session Types</th>
<th>MPA-to-WebSEAL</th>
<th>Client-to-WebSEAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Session ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP Header</td>
<td>HTTP Header</td>
<td></td>
</tr>
<tr>
<td>BA Header</td>
<td>BA Header</td>
<td></td>
</tr>
<tr>
<td>IP Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cookie</td>
<td>Cookie</td>
<td></td>
</tr>
</tbody>
</table>
• The client cannot use an SSL session ID as the session data type.
• As an example, if the MPA uses a BA header for the session data type, the client’s choices for session data type include only HTTP header and cookie.
• If the MPA uses an HTTP header for session data, the client can use a different HTTP header type.
• The server-specific cookie contains session information only; it does not contain identity information.
• If MPA support is enabled, the function of ssl-id-sessions changes. Normally, if ssl-id-sessions = yes, only the SSL session ID is used to maintain sessions for HTTPS clients. To allow the MPA to maintain a session with an SSL session ID and have clients maintain sessions using another method, this restriction is removed. See also “Determining valid session ID data types” on page 138.

The authentication method used by the MPA to WebSEAL must be distinct (different) from the authentication method used by the client to WebSEAL. The table below lists the valid authentication methods for the MPA and the client:

<table>
<thead>
<tr>
<th>MPA-to-WebSEAL</th>
<th>Client-to-WebSEAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic authentication</td>
<td>Basic authentication</td>
</tr>
<tr>
<td>Forms</td>
<td>Forms</td>
</tr>
<tr>
<td>Token</td>
<td>Token</td>
</tr>
<tr>
<td>HTTP Header</td>
<td>HTTP Header</td>
</tr>
<tr>
<td>Certificate</td>
<td></td>
</tr>
<tr>
<td>IP Address</td>
<td></td>
</tr>
</tbody>
</table>

• As an example, if the MPA uses basic authentication, the client’s choices for authentication methods includes forms, token, and HTTP header.
• Certificates and IP address authentication methods are not valid for use by the client.
• Normally, if either forms (or token) authentication is enabled for a particular transport, basic authentication is automatically disabled for that transport (see “Configuring the basic authentication mechanism” on page 145) If MPA support is enabled, this restriction is removed. This allows the MPA to log in, for example, with forms (or token) and clients to log in with basic authentication over the same transport.

Authentication process flow for MPA and multiple clients

1. The WebSEAL administrator performs the following preliminary configuration:
   • Enable support for Multiplexing Proxy Agents
   • Create a Tivoli Access Manager account for the specific MPA gateway
   • Add this MPA account to the webseal-mpa-servers group
2. Clients connect to the MPA gateway.
3. The gateway translates the request to an HTTP request.
4. The gateway authenticates the client.
5. The gateway establishes a connection with WebSEAL with the client request.
6. The MPA authenticates to WebSEAL (using a method distinct from the client) and an identity is derived for the MPA (which already has a WebSEAL account).

7. WebSEAL verifies the MPA’s membership in the webseal-mpa-servers group.

8. A credential is built for the MPA and flagged as a special MPA type in the cache.
   Although this MPA credential accompanies each future client request, it is not used for authorization checks on these requests.

9. Now WebSEAL needs to further identify the owner of the request.
   The MPA is able to distinguish the multiple clients for proper routing of login prompts.

10. The client logs in and authenticates using a method distinct from the authentication type used for the MPA.

11. WebSEAL builds a credential from the client authentication data.

12. Session data type used by each client must be distinct from the session data type used by the MPA.

13. The authorization service permits or denies access to protected objects based on the user credential and the object’s ACL permissions.

**Enabling and disabling MPA authentication**

The **mpa** parameter, located in the [mpa] stanza of the WebSEAL configuration file, enables and disables MPA authentication:

- To enable the MPA authentication method, enter "yes".
- To disable the MPA authentication method, enter "no".

For example:

```plaintext
[mpa]
mpa = yes
```

**Create a user account for the MPA**

Refer to the *IBM Tivoli Access Manager Base Administrator’s Guide* for information on creating user accounts.

**Add the MPA account to the webseal-mpa-servers group**

Refer to the *IBM Tivoli Access Manager Base Administrator’s Guide* for information on managing groups.

**MPA authentication limitations**

- This release of Tivoli Access Manager supports only one MPA per WebSEAL server.
- MPA authentication is not supported with step-up authentication configuration.
- MPA is not supported with `use-same-session = yes`
Chapter 7. Advanced WebSEAL authentication

This chapter contains information that describes advanced WebSEAL authentication functionality.

Topic Index:
- “Switch user authentication” on page 188
- “Server-side request caching” on page 200
- “Configuring reauthentication based on security policy” on page 204
- “Configuring reauthentication based on session inactivity policy” on page 208
- “Automatic redirection during user login” on page 212
- “Configuring post password change processing” on page 214
- “Extended attributes for credentials” on page 215
- “Credential refresh” on page 220
Switch user authentication

The WebSEAL switch user function allows administrators to assume the identity of a user who is a member of the Tivoli Access Manager secure domain. The ability to assume a user’s identity can help an administrator in a Help Desk environment to troubleshoot and diagnose problems. Switch user can also be used to test a user’s access to resources and to perform application integration testing.

Read this entire section to ensure that you understand the switch user function before configuring and using it. This section contains the following topics:

- “Overview of the switch user function”
- “Configuration procedure” on page 190
- “Using switch user” on page 195
- “Developing a custom authentication module for switch user” on page 197

Overview of the switch user function

The switch user implementation is similar to the su command in UNIX environments. In the WebSEAL environment, the administrator acquires the user’s credentials and interacts with resources and back-end applications with the exact same abilities as the actual user. The administrator uses a special HTML form to supply switch user information. WebSEAL processes the form and activates a special authentication mechanism that returns the specified user’s credential without the requirement of knowing the user’s password.

The following sequence describes the switch user process flow:

1. An administrator authenticates to WebSEAL. WebSEAL establishes a session for the administrator, and creates an entry for the administrator in the WebSEAL session cache.

   The session cache entry contains a cache data structure. This data structure stores the administrator’s credential. During the switch user process flow, the cache data will be manipulated.

   For more information on WebSEAL session caches, see “The WebSEAL session/credentials cache structure” on page 10.

2. The administrator connects to a pre-configured switch user HTML form, and completes the form. On the form, the administrator specifies:

   - The name of the user identity that the administrator needs to assume.
   - A destination URL.
   - An authentication method.

   This action results in a POST request being sent to /pkmssu.form.

   The WebSEAL administrator can optionally modify the contents of the switch user HTML form before making it available for use by other administrators. See “Part 3: Configuring the switch user HTML form” on page 193. The administrator can also optionally extend the capabilities of the form. See “Part 4: Designing additional input forms” on page 195.

3. WebSEAL determines whether to grant the switch user request by performing the following checks:

   a. WebSEAL examines the membership of the Tivoli Access Manager su-admins group to determine if the administrator has permission to invoke the switch user function.
Administrators requesting use of switch user authentication must be members of the su-admins group. Membership in this group must be configured before switch user can be used. For more information, see “Part 1: Configuring user access” on page 190.

b. WebSEAL examines the membership of the Tivoli Access Manager su-admins, securitygroup and su-excluded groups to ensure the user identity supplied in the switch user form is not a member of one of these groups.

User identities that belong to any of these groups cannot be accessed by the switch user function. The WebSEAL administrator must configure memberships in these groups before administrators use the switch user function. For configuration instructions and more information on these groups, see “Part 1: Configuring user access” on page 190.

4. When access to the switch user function has been granted, WebSEAL calls the appropriate switch user shared library to perform the special switch user authentication.

WebSEAL supports a number of different authentication mechanisms. Each authentication mechanism has a corresponding switch user authentication mechanism. WebSEAL provides shared libraries that contain the special switch user mechanisms. Before switch user authentication can be used, the WebSEAL administrator must configure WebSEAL to use the necessary shared libraries. For more information, see “Part 2: Configuring switch user authentication mechanisms” on page 191.

Note: Switch user authentication can also be performed by a custom switch user CDAS library. For more information, see ”Developing a custom authentication module for switch user” on page 197.

5. When authentication of the designated user succeeds, the switch user authentication mechanism returns a valid credential for the user — without requiring the user password for input.

6. WebSEAL manipulates the contents of the appropriate entry in the WebSEAL session cache by:
   a. Placing the user’s credential into a new cache data structure.
   b. Removing the administrator’s WebSEAL session cache data and storing it in a separate location.
   c. Inserting the user cache data, including the user’s credential, in place of the administrator’s cache data.

7. WebSEAL sends a redirect to the browser for the destination URL supplied in the switch user form.

The request is processed normally, using the user’s credential, and the URL is accessed.

8. The administrator can continue to make other requests. All authorization decisions for these requests are based on the credential of the user.

When using switch user functionality, administrators might need to establish and manage sessions with additional applications. These sessions need to be established using the identity of the new user. To enable this, the new user credential also contains a new User Session ID. This User Session ID is used, for example, when troubleshooting the user’s ability to access and use additional Web resources.

For more information on WebSEAL session caches, see “GSKit and WebSEAL session cache overview” on page 132 and “The WebSEAL session/credentials cache structure” on page 10.
9. The administrator ends the switch user session using the standard Tivoli Access Manager /pkmslogout utility. Upon successful log out:
   a. The user’s cache data is deleted.
   b. The administrator’s original cache data (and credential) is restored.
   c. The administrator is returned to the original page from which the switch user form was requested.

   The authorization service uses the original credential of the administrator for all subsequent requests.

**Configuration procedure**

The WebSEAL administrator must complete several configuration steps before administrators can use the switch user functionality. To configure switch user, complete the instructions in each of the following sections:

1. “Part 1: Configuring user access”
2. “Part 2: Configuring switch user authentication mechanisms” on page 191
3. “Part 3: Configuring the switch user HTML form” on page 193

   This part is optional.

4. “Part 4: Designing additional input forms” on page 195

   This part is optional.

5. “Part 5: Stopping and restarting WebSEAL” on page 195

**Part 1: Configuring user access**

During WebSEAL installation, the WebSEAL configuration process automatically creates several groups for use by the switch user functionality. The WebSEAL administrator controls switch user capability by adding users to the groups.

To configure user access, complete the following steps:

1. Add users to the **su-admins** group.
   
   To use switch user function, a user must be a member of a special administrative group called **su-admins**. This group is automatically created by default during installation of a WebSEAL server. There are no users in this group by default. The WebSEAL administrator must manually add users to this group. Typically, only administrative users are added to this group.
   
   Users who have been granted membership in **su-admins** can switch user to most other user identities, but cannot switch to the identity of any other user that is also a member of the **su-admins** group. Thus, once an administrator is granted switch user privileges by being added to **su-admins**, the administrator’s account is protected from access by any other user that gains switch user privileges.

2. Add users to the **su-excluded** group
   
   This group contains the names of users whose identities should not be accessed through the switch user capability. During WebSEAL installation, the WebSEAL configuration process automatically creates this group. There are no users in this group by default. WebSEAL administrator typically add to this group the names of users who are not members of the administrative group **su-admins**, but for whom switch user access should still be blocked.

When switch user is used, WebSEAL also checks the memberships of the Tivoli Access Manager group called **securitygroup**. This group contains the name of the Tivoli Access Manager administrative user **sec_master**, plus a number of WebSEAL processes that must be excluded from access through switch user capability.
This group is automatically created by default during installation of a WebSEAL server. The following identities are automatically added to this group during installation:

- sec_master — the Tivoli Access Manager administrator
- acld – the Tivoli Access Manager authorization server daemon
- webseald — the WebSEAL daemon

WebSEAL administrators should not add any users to the securitygroup group. To control user access to switch user, use either su-admins or su-excluded.

**Part 2: Configuring switch user authentication mechanisms**

Tivoli Access Manager supplies a single, built-in, switch user library that implements the switch user authentication mechanism. The switch user library differs from the standard authentication libraries. The library specifies an authentication mechanism that takes a supplied user identity and returns a valid credential for that user without requiring the user password for input.

The built-in switch user shared library provided with Tivoli Access Manager is called libsuauthn (on UNIX systems) and suauthn (on Windows systems). The platform-specific shared library file names are:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris Operating Environment</td>
<td>libsuauthn.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libsuauthn.a</td>
</tr>
<tr>
<td>HP-UX</td>
<td>libsuauthn.sl</td>
</tr>
<tr>
<td>Windows</td>
<td>suauthn.dll</td>
</tr>
</tbody>
</table>

The built-in library supports the following authentication mechanisms:

- su-password
- su-token-card
- su-certificate
- su-http-request
- su-cdsso
- su-kerberosv5

The authentication mechanisms are specified in the [authentication-mechanisms] stanza of the WebSEAL configuration file. There is a separate entry for each supported authentication mechanism.

By default, all of the switch user authentication mechanisms are disabled in the configuration file. For example:

```plaintext
[authentication-mechanisms]
#su-password = <su-password-library>
#su-token-card = <su-token-card-library>
#su-certificate = <su-certificate-library>
#su-http-request = <su-http-request-library>
#su-cdsso = <su-cdsso-library>
#su-kerberosv5 = <su-kerberos-library>
```

The configuration steps for enabling authentication consist primarily of editing the appropriate configuration file name=value entries. However, when the WebSEAL deployment supports more than one authentication mechanism, and the administrator wants to use switch user functions for more than one type of
supported mechanism, an additional step is required. In this case, additional copies of the default switch user shared library must be made.

The following instructions are separated into two parts. The first part describes how to configure a single switch user authentication mechanism. The second part describes how to configure multiple switch user authentication mechanisms. Use the instructions that are appropriate for your deployment.

**Configuring a single switch user authentication mechanism**

To enable a single switch user authentication mechanism, complete the following steps:

1. Edit the appropriate entry in the WebSEAL configuration file. Remove the comment character (＃) at the start of the line.
2. Enter the name of the switch user authentication library.

For example, on a Solaris system, prior to the configuration of switch user, the configuration file for a WebSEAL server that was configured to support only password authentication would contain the following entries (each entry is one continuous line):

```
[authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so & -cfgfile
[/opt/pdweb/etc/webseald-instance_name.conf]
```

The switch user library specified by the su-password entry corresponds to the authentication library specified by passwd-ldap. The entry shown below in **bold** font shows the modified entry (entered as one continuous command line):

```
[authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so & -cfgfile
[/opt/pdweb/etc/webseald-instance_name.conf]
```

```
#su-password = /opt/pdweb/etc/lib/pdwebauthn.so
```

**Configuring multiple switch user authentication mechanisms**

The default switch user shared library, **libsauthn** (UNIX) or **suauthn** (Windows) supports multiple authentication mechanisms. In the configuration file, however, each entry for a configured switch user authentication library must be **uniquely named**, even though the same shared library is used for multiple authentication methods.

In the following example, for a Solaris platform, an existing environment has two authentication methods enabled:

- Forms authentication using the built-in **libldapauthn** library
- Certificate authentication using the built-in **libsslauthn** library
The configuration file entries are:

```
[authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so & -cfgfile
       [/opt/pdweb/etc/webseald-instance_name.conf]
cert-ssl = /opt/PolicyDirector/lib/libsslauthn.so
```

To enable switch user authentication mechanisms for both of these authentication methods, complete the following steps:

1. Make a copy of the switch user shared library for each authentication mechanism. The administrator can choose any name for each copy, as long as each copy is uniquely named.
   
   For example, to support switch user for both forms authentication and certificate authentication:
   
   ```
   # cp libsuauthn.so libsuformauthn.so
   # cp libsuauthn.so libscert.so
   ```

2. Edit the appropriate entries in the WebSEAL configuration file. Remove the comment character (#) at the start of the entry for each supported switch user authentication mechanism.

3. For each uncommented entry, enter the name of the uniquely-named copy of the switch user authentication library.

   The updated configuration file entries for the example above are:

   ```
   [authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so & -cfgfile
       [/opt/pdweb/etc/webseald-instance_name.conf]
cert-ssl = /opt/PolicyDirector/lib/libsslauthn.so & -cfgfile
       [/opt/pdweb/etc/webseald-instance_name.conf]
su-password = /opt/pdwebre/lib/libsuformauthn.so
   su-certificate = /opt/pdwebre/lib/libscert.so
   ```

   The environment is now expanded to support switch user functionality for both authentication methods.

   **Note:** If your environment includes a custom CDAS authentication mechanism, you must provide the same functionality. See “Developing a custom authentication module for switch user” on page 197.

**Part 3: Configuring the switch user HTML form**

This part is optional.

WebSEAL provides a default HTML form that the administrator accesses in order to use the switch user function. The default form can be used without modification. Optionally, the form can be edited for customized appearance and functionality.

The default form is named switchuser.html. The name of this file can be modified.

**Form contents and location**

The form contains requests for:

- User name
  
  The name of the user whose credentials the administrator wants to access.

- Destination URL
  
  This page appears after a successful switch user operation.

- Authentication method
The authentication method parameters specify which authentication mechanism WebSEAL uses to build the user credential.

Each of these entries is required. WebSEAL verifies that all required data is present in the submitted form. If data is missing, the form is returned to the administrator with a descriptive message. When all required data is present, WebSEAL submits data from the switch user form data to the /pkmssu.form action URL.

Note: Only members of the su-admins group can invoke the form. An ACL is not required on this file. WebSEAL performs an internally hard-coded group membership check. WebSEAL returns a 404 “Not Found” error when the group membership check fails.

The full path name for the switch user form is defined in the WebSEAL configuration file. This path name can be modified.

The values for three parameters are combined to build the full path name:
- The server-root parameter located in the [server] stanza specifies the root of the server hierarchy.
- The mgt-pages-root parameter in the [acnt-mgt] stanza specifies the localization sub-directory.
- The switch-user parameter in the [acnt-mgt] stanza specifies the name of the switch user file.

For example, on a UNIX system, the configuration file entries would be:

```plaintext
[server]
server-root = /opt/pdweb/www-instance_name
....
[acnt-mgt]
mgt-pages-root = lib/html/<LANG>
switch-user = switchuser.html
```

The value of the LANG directory is specific to the locale. You can determine the full path to the switch user form by combining the values. For example, on a UNIX system, with a U.S. English locale where the LANG directory is called "C", the full path would be:

```
/opt/pdweb/www-instance_name/lib/html/C/switchuser.html
```

The default value of server-root on Windows is:

```
C:\Program Files\Tivoli\PDWeb\www-instance_name
```

The full path on Windows would be:

```
C:\Program Files\Tivoli\PDWeb\www-instance_name\lib\html\C\switchuser.html
```

How to customize the HTML form

To customize the switch user form, open the form for editing, and complete the following steps:

1. Specify the location and contents of the destination URL.
   You can configure this as hidden input containing an appropriate home page or a successful switch user confirmation page.

2. Specify the authentication methods
   You can configure this field as hidden input. Valid values for the authentication method include:
su-ba
su-forms
su-certificate
su-token-card
su-http-request
su-cdss0

The methods in the list above map directly to authentication mechanisms specified in the WebSEAL configuration file. Note, however, that the **su-ba** and **su-forms** methods both map to the **su-password** authentication mechanism. Both basic authentication (ba) and forms authentication (forms) use the su-password authentication library. Note that a WebSEAL deployment can support basic authentication without supporting forms authentication. Thus separate configuration values are maintained for each authentication type (su-ba and su-forms).

**Part 4: Designing additional input forms**

This part is optional.

You can design additional forms to validate or process data to be submitted to `/pkmssu.form`. These forms can be used to assist the administrator by populating some of the entries on the switch user form.

Some examples are:

- An administrator might have chosen to have different destination URLs, to be accessed based on the user identity. Another form could be written to build and present a list of these URLs, from which the administrator could select the appropriate entry.
- A form could be developed to call another program, such as a CGI script, to supply a list of user identities for whom switch user is allowed. This list could help administrators determine if access to a user identity through switch user is allowed.
- A form could be developed to display a list of user identities for whom switch user is not allowed. This list would be based on the memberships of the su-excluded and securitygroup groups.

**Part 5: Stopping and restarting WebSEAL**

To activate the new configuration changes you must stop and restart WebSEAL. This enables WebSEAL to use the new values that were specified to the WebSEAL configuration file in “Part 1: Configuring user access” on page 190 and “Part 2: Configuring switch user authentication mechanisms” on page 191.

The methods for stopping and restarting the WebSEAL server are described in “Server tasks” on page 72.

**Using switch user**

When the configuration steps in the previous section have been completed, WebSEAL administrators can use the switch user function.

To use the switch user function, complete the following steps:

1. Log in as a user who has permission to access the switch user function. This function is usually accessed by administrators. The user must be a member of the su-admins group.
2. Invoke the switch user HTML form.
The default file name is switchuser.html. For information on the full path name, see “Part 3: Configuring the switch user HTML form” on page 193.

3. On the form, specify:
   • The name of the user identity that you want to assume.
   • A destination URL
   • An authentication method

This action results in a POST request being sent to /pkmsu.form. WebSEAL sends a redirect to the browser for the destination URL supplied in the switch user form. The request is processed using the user’s credential, and the URL is accessed.

4. Make other requests as necessary.
   All authorization decisions for these requests are based on the credential of the user.

5. When finished, end the switch user session by using the standard Tivoli Access Manager /pkmslogout utility.

For more information on how the switch user function works, see “Overview of the switch user function” on page 188.

Additional switch user features
This section describes switch user support for additional features such as reauthentication, step-up authentication, user session management, and auditing.

Session cache timeout
The functionality of the configured WebSEAL session cache inactivity and lifetime timeout values is not affected by the switch user operation. The inactivity and lifetime timers are associated with the administrator’s session cache entry and not the cache data that changes during a switch user operation.

The inactivity timer continues to be reset while the administrator performs requests as the "switched-to" user. When the administrator ends the switch user session, the inactivity is still valid for the re-established administrator session.

The lifetime value is not extended because of a switch user operation. It is possible for the lifetime timeout of the session cache entry to expire during a switch user operation. If this timeout occurs, the session cache is deleted and the administrator is logged off. The administrator must reauthenticate and begin the switch user operation again.

Step-up authentication
The shared library specification can take additional arguments in the form:

<library>& <arg1> <arg2> ..... <argx>

You can designate step-up authentication levels using the -l option followed by the level number. For example:

su-password = /opt/PolicyDirector/lib/libsuformauthn.so& -l 1
su-certificate = /opt/PolicyDirector/lib/libsucert.so& -l 0
su-token-card = /opt/PolicyDirector/lib/libsucustom.so& -l 2

Note: The administrator must know the user’s password to successfully perform step-up authentication.
Reauthentication
WebSEAL reauthentication functionality is recognized by the switch user operation. If reauthentication is required during a switch user operation, the administrator must authenticate as the "switched-to" user.

Note: The administrator must know the "switched-to" user’s password to successfully reauthenticate.

User session management
The switch user operation supports user session management. The administrator has a unique User Session ID. Additionally, during a switch user operation, a unique User Session ID exists for the "switched-to" user. The terminate single user sessions task and terminate all user sessions task perform as expected.

Tag-value
The tag-value capability often used by a CDAS is recognized and supported by the switch user functionality.

Auditing
It is possible to audit the administrator during a switch user operation. The switch user functionality adds an extended attribute to the "switch-to" user credential that identifies the administrator. The extended attribute, as stored in the credential, is called tagvalue_su-admin:

\[
tagvalue_su-admin = <su-admin-name>
\]

This extended attribute is available to any auditing mechanism.

Developing a custom authentication module for switch user
The switch user functionality also supports custom authentication module. This support is important because an existing custom authentication module often returns additional information about the user that is incorporated into the user’s credential. A custom authentication module can be used to perform further checks regarding switch user capability, such as determining which users can switch user to other users’ identities, or specifying time periods when switch user capability is not allowed.

If you are using the switch user feature in such an environment, you must write a special switch user authentication module that emulates the behavior of your existing authentication module while supporting the requirement of returning a credential without requiring the user password for input.

The Tivoli Access Manager external authentication API provides a set of identity components that can be used to pass client authentication information to the switch user shared library. This information is passed using a name/value list format, where the name is an identifier that specifies the value type.

The information is stored in the \texttt{xnlist_t data} type. Values can be accessed by using the utility function \texttt{xnvlist_get()}. 

Identity components appropriate for a switch user authentication module include:

\[
xauthn_su_method
xauthn_admin_name
xauthn_admin_cred
xauthn_existing_cred
\]
xauthn_username
xauthn_qop
xauthn_ipaddr
xauthn_browser_info

The xauthn_browser_info, xauthn_qop, and xauthn_ipaddr identity components represent those of the administrator, not the "switched to" user. This data is supplied for any authentication module that must perform additional validations of the administrator's account.

**Note:** Refer to the IBM Tivoli Access Manager for e-business Web Security Developer Reference for more information on writing a custom authentication module.

### Configuring a custom authentication module for switch user

The following example expands on the example presented in "Part 2: Configuring switch user authentication mechanisms" on page 191. The example adds a custom authentication module to list of enabled authentication mechanisms. The example, for a Solaris platform, shows an existing environment that has three authentication methods enabled:

- Forms authentication using the built-in libldapauthn library
- Certificates authentication using the built-in libsslauthn library
- Token authentication using a custom authentication module

In this example, the administrator wants to be able to use switch user authentication for all three authentication methods. Thus, three additional authentication parameters for switch user must be enabled in the WebSEAL configuration file. The third parameter represents the new custom authentication module library that was written to emulate the existing token authentication and support the requirements of switch user authentication:

The configuration file entries before enabling switch user for the three authentication mechanism are:

```plaintext
[authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so
cert-ssl = /opt/PolicyDirector/lib/libsslauthn.so
token-cdas = /opt/PolicyDirector/lib/libcustom.so
```

Note that the example token custom authentication library is called libcustom.so. The new switch user version of this token custom authentication library will be called libsucustom.so.

After adding the switch user authentication mechanism, the configuration file entries are:

```plaintext
[authentication-mechanisms]
passwd-ldap = /opt/PolicyDirector/lib/libldapauthn.so
cert-ssl = /opt/PolicyDirector/lib/libsslauthn.so
token-cdas = /opt/PolicyDirector/lib/libcustom.so
su-password = /opt/PolicyDirector/lib/libsuformauthn.so
su-certificate = /opt/PolicyDirector/lib/libsuauthn.so
su-token-card = /opt/PolicyDirector/lib/libsucustom.so
```

Notice the following changes:

- The new entry for the authentication module has the name su-token-card. The value for this entry is the full path name to the shared library that has been extended to support switch user.
- For the non-custom authentication methods in this example, remember that:
- The **su-forms** authentication method supplied in the switch user form is mapped to the **su-password** authentication mechanism parameter in the WebSEAL configuration file.

- The supplied `libsuauthn` library has been renamed for both the Forms and certificate mechanisms.
Server-side request caching

This section contains the following topics:

- "Overview of server-side request caching"
- "Configuring server-side caching parameters" on page 201

Overview of server-side request caching

In past versions of WebSEAL using Forms authentication, WebSEAL created a cache entry for the URL of a user request whenever authentication was required. Upon successful authentication, WebSEAL sent an HTTP redirect to the browser that included this URL. The browser then followed the redirect to the original resource location.

The limitation of this implementation became apparent when, for example, a POST request was interrupted by a session timeout that prompted a re-authentication process. Since WebSEAL only cached the URL of the original request, the POST data (including the METHOD and Message Body) were lost during the HTTP redirect. The user had to rebuild the POST request.

WebSEAL now caches a more complete set of request data and uses this cached data to rebuild the request during the HTTP redirect, if a re-authentication requirement interrupts the completion of the request processing. This solution particularly benefits POST and PUT requests, because these requests types can include a variety of information fields.

When an authentication requirement interrupts a request, WebSEAL caches all information necessary to rebuild the request during the HTTP redirect that follows after re-authentication. Cached request data includes URL, METHOD, Message Body, query strings, and all HTTP headers (including cookies). This data is temporarily stored in the WebSEAL credentials/session cache.

Upon successful authentication (or re-authentication), WebSEAL sends an HTTP redirect to the browser. The browser follows the redirect to the original URL contained in the redirect. WebSEAL intercepts the redirect and rebuilds the request using the cached data. The rebuilt request is delivered to the URL destination.

The following diagram illustrates a typical server-side request caching process flow:

1. User successfully logs in (Forms authentication) and submits an HTTP request for a resource involving a CGI-generated data form. WebSEAL creates and caches a session ID for the user.
2. The back-end application server returns the form to the user.
3. During the time it takes the user to fill in the form, the configured session timeout for the user expires. WebSEAL removes the user’s credentials cache entry and session ID.
4. The user eventually submits the completed form (POST). WebSEAL finds no cache entry for the user, creates a new cache entry, and temporarily caches the complete information contained in the POST request.
5. Because WebSEAL finds no credentials for this user, the user must authenticate. WebSEAL sends a login form to the user.
6. The user returns the completed login form to WebSEAL (POST). Authentication is successful. The cache now contains the user’s credentials, as well as the cached request.
7. WebSEAL sends an HTTP redirect back to the browser containing the URL of the originally requested resource.

8. The browser follows the redirect (GET). WebSEAL intercepts the redirect and rebuilds the original request (form) using the cached POST data. The restored request (form) is delivered to the URL designation.

Figure 6. Example WebSEAL request caching process flow

Usage notes
- The server-side caching parameters protect WebSEAL from denial of service attack types that could cause WebSEAL to cache more data than it can handle.
- Server-side request caching will not function correctly if the user session time out value expires during the login process. In this situation, the cache entry is lost.
- Server-side request caching can cause limitations with the browser’s ability to manipulate the resource. The browser is unaware that WebSEAL has rebuilt the HTTP redirect. Therefore the browser’s reload/refresh function and caching ability can be hindered.

Configuring server-side caching parameters
WebSEAL automatically caches requests during Forms authentication. You can modify settings in the [server] stanza of the WebSEAL configuration file to specify limits to the size of the requests that WebSEAL caches. The following sections describe the settings that you can modify:
- “Modifying max-client-read” on page 202
- “Modifying request-max-cache” on page 203
Server-side cache parameters are also summarized in the configuration file reference appendix. See "Server configuration" on page 384.

### Modifying max-client-read
This parameter specifies the maximum number of bytes that WebSEAL holds in internal buffers while reading from a client. It affects the maximum size of URLs, HTTP Headers, and the size of a request that will be cached. This parameter must be set to at least twice the value of request-body-max-read. The minimum value is 32768. If the value is set to a number below 32768, the value is ignored and a value of 32768 is used. The default value is 32768

[server]
max-client-read = 32768

This value can be increased when your deployment needs to cache abnormally large requests. There is no maximum size other than the maximum imposed by the data type. However, increasing the size can possibly adversely affect performance and system security. Allocating larger buffers increases memory usage and thus could possibly decrease performance. More importantly, allocating very large buffers increases the risk of a successful denial-of-service attack by a malicious user. The risk is increased simply because WebSEAL is loading and holding more data into memory, which provides the user with a larger buffer from which to attempt an attack.

### Modifying request-body-max-read
This parameter specifies the maximum number of bytes to read in as content from the body of requests for use in dynurl, authentication, and request caching. This affects the amount of data that WebSEAL caches for users who must authenticate before their request can be fulfilled. This affects all request that have bodies, such as POST and, PUT requests.

This impacts forms authentication, because this limits the size of the POST data that is processed when performing such authentication. To maintain a request body size sufficient for forms authentication, WebSEAL sets a hard minimum of 512 bytes on request-body-max-read. When this value is set below that minimum, the setting will be ignored and the value 512 is used.

This setting also impacts dynamic URL processing because the query portion of a POST request URI is contained in the request body.

**Note:** This setting does not limit the maximum POST size. The maximum POST size is unlimited.

The value of request-body-max-read is related to the value of max-client-read. When request-body-max-read is set to a value higher than 16384, max-client-read must be increased to at least twice the value of request-body-max-read. When the value of max-client-read is less than double the value of request-body-max-read, WebSEAL ignores request-body-max-read and imposes a value of half the value of max-client-read.

The default value is 4096:

[server]
request-body-max-read = 4096

You can set this parameter to zero (0).
When the server-side cache setting for request-body-max-read is exceeded during a request, WebSEAL aborts the request caching process. WebSEAL returns a Request Caching Failed error message to the browser, and writes the error to the log file. You can customize this error message. See “HTTP error message pages” on page 84.

The value of request-body-max-read also affects the value specified for request-max-cache. See “Modifying request-max-cache.”

**Modifying request-max-cache**

When a user is prompted to authenticate before a request can be fulfilled, the data from that request is cached for processing after the completion of the authentication. The maximum amount of data cached per request is specified by the request-max-cache parameter.

To ensure that you cache all of request-body-max-read worth of the body of requests, you must account for the maximum size of all the other request components in this value. For example: If you want to cache 2048 bytes of request bodies, and you anticipate that the maximum size of all request headers and cookies will be 4096 bytes, you would:

1. Set request-body-max-read = 2048
2. Set request-max-cache = 2048 + 4096 = 6144

    [server]
    request-max-cache = 8192

The default value for request-max-cache is 8192.

When the server-side cache setting for request-max-cache is exceeded during a request, WebSEAL aborts the request caching process. WebSEAL returns a Request Caching Failed error message to the browser, and writes the error to the log file. You can customize this error message. See “HTTP error message pages” on page 84.
Configuring reauthentication based on security policy

Tivoli Access Manager WebSEAL can force a user to perform an additional login (reauthentication) to ensure that a user accessing a protected resource is the same person who initially authenticated at the start of the session. Reauthentication can be activated by a Protected Object Policy (POP) on the protected object or by expiration of the inactivity timeout value of a WebSEAL session cache entry.

This section discusses reauthentication based on security policy as dictated by a POP extended attribute.

Background material on the WebSEAL session cache is provided in “The WebSEAL session/credentials cache structure” on page 16.

Conditions affecting POP reauthentication

Forced reauthentication provides additional protection for sensitive resources in the secure domain. Reauthentication based on security policy is activated by a specific extended attribute in a POP that protects the requested resource object. The POP can be directly attached to the object, or the object can inherit the POP conditions from a parent object.

Reauthentication is supported by the following WebSEAL authentication methods:

- Forms (user name and password) authentication
- Token authentication

In addition, a custom user name/password CDAS can be written to support reauthentication.

Reauthentication assumes the user has initially logged in to the secure domain and that a valid credential exists for the user. During reauthentication, the user must log in using the same identity that generated the existing credential. WebSEAL preserves the user’s original session information, including the credential, during reauthentication. The credential is not replaced during reauthentication.

During reauthentication, WebSEAL also caches the request that prompted the reauthentication. Upon successful reauthentication, the cached data is used to rebuild the request. See “Server-side request caching” on page 200.

If reauthentication fails, WebSEAL returns the login prompt again. If reauthentication succeeds, but the ACL check fails for that resource, a 403 “Forbidden” is returned and the user is denied access to the requested resource. In either case, the user is never logged off. Using a still valid credential, the user can abort the reauthentication process (by requesting another URL) and still participate in the secure domain by accessing other resources that do not require reauthentication.

Configuration is available to reset the lifetime timer of WebSEAL session cache entries. In addition, a grace period can be configured to allow sufficient time for the reauthentication process to complete before the lifetime timeout of a session cache entry expires.
Creating and applying the reauthentication POP

Forced reauthentication based on security policy is configured by creating a protected object policy (POP) with a special extended attribute named "reauth". You can attach this POP to any object that requires the extra protection provided by forced reauthentication.

Remember that all children of the object with the POP also inherit the POP conditions. Each requested child object requires a separate reauthentication.

Use the `pdadmin pop create`, `pdadmin pop modify`, and `pdadmin pop attach` commands. The following example illustrates creating a POP called "secure" with the `reauth` extended attribute and attaching it to an object (budget.html):

```
pdadmin> pop create secure
pdadmin> pop modify secure set attribute reauth true
pdadmin> pop attach /WebSEAL/hostA/junction/budget.html secure
```

Anyone attempting to access budget.html is forced to reauthenticate using the same identity and authentication method that generated the existing credential.

If the user requesting the resource is unauthenticated, the POP forces the user to authenticate. No reauthentication is necessary for this resource after successful initial login.

Details about the `pdadmin` command line utility can be found in the *IBM Tivoli Access Manager Base Administrator's Guide*.

Configuring session cache entry lifetime reset and extension

Resetting the session cache entry lifetime value

The user's session cache entry has a limited lifetime, as specified by the `timeout` parameter in the `[session]` stanza of the `webseald.conf` configuration file. The default value, in seconds, is 3600 (1 hour):

```
[session]
timeout = 3600
```

Regardless of session activity or inactivity, the session cache entry is removed when the lifetime value is reached, at which point the user is logged off.

However, you can configure the lifetime of the session cache entry to be reset whenever reauthentication occurs. With this configuration, the user session no longer has a single maximum lifetime value. Each time reauthentication occurs, the lifetime value of the session cache entry is reset.

You can configure session cache entry lifetime reset with the `reauth-reset-lifetime` parameter in the `[reauthentication]` stanza of the `webseald.conf` configuration file:

```
[reauthentication]
reauth-reset-lifetime = yes
```

The default value is "no".

This parameter is also appropriate for reauthentication due to the expiration of the inactivity timeout value for a session cache entry. See "Configuring reauthentication based on session inactivity policy" on page 208.
Extending the session cache entry lifetime value

It is possible for the lifetime value of a session cache entry to expire while the user is performing a reauthentication. This situation occurs under the following conditions:

• The user requests a resource protected by a reauthentication POP
• The user’s session cache entry lifetime value is very near expiration

The lifetime of a session cache entry can expire after the reauthentication login form is sent to the user and before the completed login form is returned. When the session cache entry lifetime value expires, the session cache entry is deleted. When the login form is returned to WebSEAL, there is no longer a session for that user. In addition, all cached user request data is lost.

You can configure a time extension, or “grace period,” for the session cache entry lifetime value if the session cache entry lifetime expires during reauthentication. The reauth-extend-lifetime parameter in the [reauthentication] stanza of the webseald.conf configuration file provides this time extension, in seconds. For example (5 minutes):

[reauthentication]
reauth-extend-lifetime = 300

The default value, "0", provides no extension to the session cache entry timeout value.

The reauth-extend-lifetime parameter applies to users with existing session cache entries and who are required to reauthenticate. For example:

• Users performing reauthentication resulting from POP security policy
• Users performing reauthentication resulting from session cache inactivity
• Users performing step-up authentication

The reauth-extend-lifetime option is intended to be used in conjunction with the reauth-reset-lifetime=yes option.

This parameter is also appropriate for reauthentication due to the expiration of the session cache entry inactivity timeout value. See "Configuring reauthentication based on session inactivity policy" on page 208.

Customizing login forms for reauthentication

WebSEAL supports reauthentication for both forms and token authentication methods. By default, forms authentication uses the login.html page to request user name and password information from the client (see "Custom HTML page descriptions" on page 89). By default, token authentication uses the tokenlogin.html page to request user name and token passcode information from the client (see "Custom HTML page descriptions" on page 89). These same default login pages are also used during reauthentication.

During initial login, both user name and password (passcode) fields are blank in each of these login pages. However, it is possible to have the user name field in these login pages automatically filled in during reauthentication by using the %USERNAME% macro (see "Macro support for account management pages" on page 89). The client needs to complete only the password (passcode) field.

For example, modify the following line in the login.html page:

<TD><INPUT NAME="username" SIZE="15"></TD>
to include the %USERNAME% macro:

\[
<TD><INPUT NAME="username" SIZE="15" VALUE="%USERNAME%"></TD>
\]

During an initial login, the value for the %USERNAME% macro is empty and the user name text field displayed on the login page appears blank. For a reauthenticating client, the %USERNAME% macro now contains the value of the client user name. The user name text field on the login page appears with the user’s name automatically provided.
Configuring reauthentication based on session inactivity policy

Tivoli Access Manager WebSEAL can force a user to perform an additional login (reauthentication) to ensure that a user accessing a protected resource is the same person who initially authenticated at the start of the session. Reauthentication can be activated by a Protected Object Policy (POP) on the protected object or by expiration of the inactivity timeout value for a WebSEAL session cache entry.

This section discusses reauthentication based on the expiration of the inactivity timeout value for a WebSEAL session cache entry.

Background material on the WebSEAL session cache is provided in “The WebSEAL session/credentials cache structure” on page 16.

Conditions affecting inactivity reauthentication

Forced reauthentication provides additional protection for sensitive resources in the secure domain. Reauthentication based on session inactivity policy is enabled by a configuration parameter and is activated by the expiration of the inactivity timeout value of the session cache entry.

Reauthentication is supported by the following supported WebSEAL authentication methods:

- Forms (user name and password) authentication
- Token authentication

In addition, a custom user name/password CDAS can be written to support reauthentication.

Reauthentication assumes the user has initially logged in to the secure domain and that a valid credential exists for the user. During reauthentication, the user must log in using the same identity that generated the existing credential. WebSEAL preserves the user’s original session information, including the credential, during reauthentication. The credential is not replaced during reauthentication.

During reauthentication, WebSEAL also caches the request that prompted the reauthentication. After successful reauthentication, the cached data is used to rebuild the request. See “Server-side request caching” on page 200.

A user’s session is normally regulated by a session inactivity value and a session lifetime value. When WebSEAL is configured for reauthentication based on session inactivity, the user’s session cache entry is “flagged” whenever the session inactivity timeout value expires. The session cache entry (containing the user credential) is not removed. The user can proceed to access unprotected resources. However, if the user requests a protected resource, WebSEAL sends a login prompt. After successful reauthentication, the inactive session “flag” is removed and the inactivity timer is reset.

If reauthentication fails, WebSEAL returns the login prompt again. The session cache entry remains “flagged” and the user can proceed as unauthenticated until the session cache entry lifetime value expires.

If reauthentication succeeds, but the ACL check fails for that resource, a 403 "Forbidden" is returned and the user is denied access to the requested resource.
The session cache entry lifetime value usually determines the maximum session length. When this lifetime value expires, the session is normally terminated regardless of activity. However, WebSEAL can be configured to allow the user to reauthenticate after the session lifetime value has expired. After successful reauthentication, the lifetime value of the session cache entry is reset.

Two other conditions can end a user session: the user can explicitly log out or an administrator can terminate a user session. See “Terminating user sessions” on page 345.

Configuration is available to reset the WebSEAL session cache entry lifetime timer. In addition, a grace period can be configured to allow sufficient time for the reauthentication process to complete before the lifetime timeout of the session cache entry expires.

**Enabling inactivity reauthentication**

To configure WebSEAL to “flag” inactive sessions rather than remove them from the session cache, set the value for the `reauth-for-inactive` parameter to “yes” in the `[reauthentication]` stanza of the `webseald.conf` configuration file:

```
[reauthentication]
reauth-for-inactive = yes
```

The default value for this parameter is “no”.

**Resetting and extending the session cache entry lifetime value**

**Resetting the session cache entry lifetime value**

The user’s session cache entry has a limited lifetime, as specified by the `timeout` parameter in the `[session]` stanza of the `webseald.conf` configuration file. The default value, in seconds, is 3600 (1 hour):

```
[session]
timeout = 3600
```

Regardless of session activity or inactivity, the session cache entry is removed when the lifetime value is reached, at which point the user is logged off.

However, you can configure the lifetime of the session cache entry to be reset whenever reauthentication occurs. With this configuration, the user session no longer has a single maximum lifetime value. Each time reauthentication occurs, the session cache entry lifetime value is reset.

You can reset the lifetime value of session cache entries with the `reauth-reset-lifetime` parameter in the `[reauthentication]` stanza of the `webseald.conf` configuration file:

```
[reauthentication]
reauth-reset-lifetime = yes
```

The default value is “no”.

This parameter is also appropriate for reauthentication due security (POP) policy. See “Configuring reauthentication based on security policy” on page 204.
Extending the session cache entry lifetime value
It is possible for the lifetime value of a session cache entry to expire while the user is performing a reauthentication. This situation occurs under the following conditions:

- The user requests a resource protected by a reauthentication POP
- The user’s session cache entry lifetime value is very near expiration

The session cache entry lifetime can expire after the reauthentication login form is sent to the user and before the completed login form is returned. When the session cache entry lifetime value expires, the session cache entry is deleted. When the login form is returned to WebSEAL, there is no longer a session for that user. In addition, all cached user request data is lost.

You can configure a time extension, or “grace period,” for the session cache entry lifetime if the lifetime of the session cache entry expires during reauthentication. The `reauth-extend-lifetime` parameter in the `[reauthentication]` stanza of the `webseald.conf` configuration file provides this time extension, in seconds. For example (5 minutes):

```
[reauthentication]
reauth-extend-lifetime = 300
```

The default value, "0", provides no extension to the timeout value of the session cache entry.

The `reauth-extend-lifetime` parameter applies to users with existing session cache entries and who are required to reauthenticate. For example:

- Users performing reauthentication resulting from POP security policy
- Users performing reauthentication resulting from session cache inactivity
- Users performing step-up authentication

The `reauth-extend-lifetime` option is intended to be used in conjunction with the `reauth-reset-lifetime=yes` option.

This parameter is also appropriate for reauthentication due to security (POP) policy. See “Configuring reauthentication based on security policy” on page 204

Preventing session removal when the session lifetime expires
It is possible for a user to remain active for the full duration of a session lifetime. When the session lifetime value expires, the session cache entry is normally removed and the user is logged off. To prevent this sudden session termination, you can configure WebSEAL to allow the user to reauthenticate after the session timeout value has expired.

WebSEAL allows resetting of the session lifetime value, after it has expired, under the following conditions:

- Reauthentication based on inactivity policy is enabled (`reauth-for-inactive=yes`)
- The session lifetime value (`timeout`) has expired
- The time extension ("grace period") for the session lifetime is enabled and set to a reasonable value (for example, `reauth-extend-lifetime=300`)
- The user activates the reauthentication prompt by requesting a protected resource before the time extension ("grace period") expires

(WebSEAL does not allow repeated additions of the time extension to an end of session lifetime event.)
• Resetting the session cache lifetime is configured to be true (reauth-reset-lifetime=yes)

At the occurrence of a session lifetime expiration, WebSEAL checks the conditions listed above. If all conditions are met, the lifetime timeout is extended by the reauth-extend-lifetime value and the user’s session cache entry is “flagged” as extended. The session cache entry (containing the user credential) is not removed and the use can proceed to access unprotected resources. When the user requests a protected resource, WebSEAL prompts the user to reauthenticate.

The reauth-extend-lifetime value should be set to a reasonable value so the user has enough time to trigger the reauthentication prompt. Note that if the user does not access a protected object during the “grace period”, the reauthentication process is not activated. In this case, it is possible for the reauth-extend-lifetime value to expire, in which case the session cache entry is removed.

Typically, however, reauthentication policy is implemented to secure an application that is serving predominantly protected resources. A time extension (“grace period”) of 5–10 minutes should be adequate time to allow an active user to trigger the reauthentication process, and thus reset the session lifetime value.

**Customizing login forms for reauthentication**

WebSEAL supports reauthentication for both forms and token authentication methods. By default, forms authentication uses the login.html page to request user name and password information from the client (see "Custom HTML page descriptions" on page 89). By default, token authentication uses the tokenlogin.html page to request user name and token passcode information from the client (see "Custom HTML page descriptions" on page 89). These same default login pages are also used during reauthentication.

During initial login, both user name and password (passcode) fields are blank in each of these login pages. However, it is possible to have the user name field in these login pages automatically filled in during reauthentication by using the %USERNAME% macro (see "Macro support for account management pages" on page 89). The client needs to complete only the password (passcode) field.

For example, modify the following line in the login.html page:

```html
<TD><INPUT NAME="username" SIZE="15"></TD>
```

To include the %USERNAME% macro:

```html
<TD><INPUT NAME="username" SIZE="15" VALUE="%USERNAME%"></TD>
```

During an initial login, the value for the %USERNAME% macro is empty and the user name text field displayed on the login page appears blank. For a reauthenticating client, the %USERNAME% macro now contains the value of the client user name. The user name text field on the login page appears with the user’s name automatically provided.
Automatic redirection during user login

This section contains the following topics:
- “Overview of automatic redirection”
- “Enabling automatic redirection”
- “Disabling automatic redirection” on page 213
- “Limitations” on page 213

Overview of automatic redirection

When a user makes a request for a resource in a WebSEAL domain, WebSEAL sends the resource to the user upon successful authentication and policy checks. As an alternative to this standard response, you can configure WebSEAL to automatically redirect the user to a specially designated home, or welcome page. This forced redirection at login is appropriate, for example, when users enter the WebSEAL domain through a portal page. Automatic redirection also overrides user attempts to directly access specific pages within the domain by selecting user bookmarks.

The automatic redirection following process flow is as follows:

1. The user sends a request and successfully authenticates.
2. WebSEAL builds a custom response and returns it to the browser as a redirect.
   - This redirect response contains the URL value specified by the login-redirect-page parameter in the WebSEAL configuration file.
3. The browser follows the redirect response (containing the configured URL).
4. WebSEAL returns the page located at the configured URL.

Automatic redirection at login is enabled and disabled independently for each authentication method. Automatic redirection is supported for the following authentication methods:
- Basic authentication
- Forms authentication
- Token authentication

Enabling automatic redirection

To configure automatic redirection, complete the following steps:

1. Open the WebSEAL configuration file for editing.
2. Enable automatic redirection for each of the applicable authentication methods by uncommenting the entry for each method in the [enable-redirects] stanza:
   
   [enable-redirects]
   redirect = forms-auth
   redirect = basic-auth
   redirect = token-auth

   The example above enabled automatic redirection for forms authentication, basic authentication, and token authentication.

3. Specify the URL to which the user is redirected after login.
   - The URL can be expressed as an absolute or server-relative path. For example:

   [acct-mgt]
   login-redirect-page = http://www.ibm.com

   or
Disabling automatic redirection
To disable automatic redirection, complete the following steps:
1. Open the WebSEAL configuration file for editing.
2. Disable automatic redirection for each of the applicable authentication methods by commenting or removing the entry for each authentication method in the [enable-redirects] stanza:
   
   ```
   [enable-redirects]
   #redirect = forms-auth
   #redirect = basic-auth
   #redirect = token-auth
   ```
   
   Note that the hash character ( # ) is added to the start of each line. The example above disabled automatic redirection for forms authentication, basic authentication, and token authentication.
3. Stop and restart the WebSEAL server.

Limitations
WebSEAL does not support automatic redirection at login under the following conditions:

- When a Windows client has authenticated using SPNEGO protocol (and Kerberos authentication) as part of Windows desktop single sign-on.
- During reauthentication.
- When the browser is reopened while using basic authentication.

Redirection works as expected the first time a user visits a page with a browser and authenticates with a valid user name and password. However, if that instance of the browser is closed and another opened, the redirected page is not displayed after the user is authenticated.
Configuring post password change processing

WebSEAL can be configured to allow customized processing to occur after a successful password change operation. Password changes can occur voluntarily using the `pkmspasswd` command from the browser, or through mandated actions dictated by password security policies (such as password expiration).

The post password change processing capability allows you, for example, to update one or more external user registries without additional authentication overhead.

The post password change processing functionality relies on an additional authentication mechanism configured in the `webseald.conf` configuration file. If a password change is successful, WebSEAL checks for this additional mechanism. If the password change fails, WebSEAL does not check for the additional mechanism.

The additional authentication mechanism is a custom authentication library, written using the external authentication C API. Refer to the IBM Tivoli Access Manager WebSEAL Developer’s Reference for complete reference information for this API. If the custom mechanism is configured and the password change is successful, the custom library receives the old password, the new password, and the user name.

To configure post password change processing, use the `post-pwd-change` parameter in the `[authentication-mechanisms]` stanza of the `webseald.conf` configuration file to specify the full path to the custom library file:

```
[authentication-mechanisms]
post-pwdchg-process = <full-path-library-name>
```

For example (Solaris):
```
[authentication-mechanisms]
post-pwdchg-process = /opt/PolicyDirector/lib/reg2update.so
```

Post password change processing conditions

- WebSEAL calls a configured custom library only when a password change is successful
- Errors are returned as success or failure. It is the responsibility of the developer to properly handle any errors locally. This success or failure is audited, but not acted on.
- Failure of post password change processing does not cause a successful password change to fail
Extended attributes for credentials

This section contains the following topics:

- “Mechanisms for adding registry attributes to a credential”
- “Registry attribute entitlement service configuration” on page 216
- “Junction handling of extended credential attributes” on page 218

Mechanisms for adding registry attributes to a credential

The WebSEAL authentication process accesses the Tivoli Access Manager user registry and builds a credential for the user. The credential contains user information that is needed to make access decisions. This includes information such as user name and list of groups to which the user belongs.

WebSEAL supports several different mechanisms (services) that allow administrators and application developers to extend the authentication process. When WebSEAL conducts the authentication process, it checks to see if any external services have been implemented and configured. When they have, WebSEAL calls those services. The services can do their own processing to build a list of extended attributes about the user identity. These extended attributes are added to the user credential.

The following types of services are supported:

- Registry attribute entitlement service

This entitlement service is built-in to Tivoli Access Manager by default. This service is an implementation of a class of Tivoli Access Manager entitlement services known as credential attribute entitlement services. This service is called a registry attribute entitlement service because it obtains specified user information from a user registry (such as an LDAP user registry) and inserts the data into an attribute list in the user credential. This built-in registry attribute entitlement service is a generic entitlement service that can be used by many resource managers. This service takes the place of a previous method that required administrators to add "tag/value" entries to the [ldap-ext-creds-tag] stanza in the pd.conf configuration file. In Version 5.1, you should use the built-in entitlement service to obtain LDAP user registry data. For configuration information, see “Registry attribute entitlement service configuration” on page 216.

Note: Note that Tivoli Access Manager provides additional built-in entitlement services that can be used to add additional information. These additional services, however, obtain the additional information from sources other than user registry entries. For example, the extended attribute entitlement service obtains information from ACLs and POPs in the protected resource object space. For more information, see the description of entitlement services in the IBM Tivoli Access Manager for e-business Authorization C API Developer Reference.

- Customized credential attribute entitlement service

When the built-in credential attribute entitlement service cannot provide all of the information needed for your deployment, you can write your own credential attribute entitlement service. This can include your own version of a registry attribute entitlement service. Tivoli Access Manager supports this as part of the authorization API. For more information, see the IBM Tivoli Access Manager for e-business Authorization C API Developer Reference.

- Credential extended attributes external authentication service (CDAS)
WebSEAL provides an external authentication API interface that can be used to develop external authentication services. These services are commonly called CDASs (cross-domain authentication service).

You can use the WebSEAL external authentication API to develop your own external authentication service. This can be used when the need to obtain user authentication information extends beyond entitlement information. The use of a credential extended attributes CDAS is recommended when an application needs to access information available only at authentication time, or when the application needs to map a user ID used at authentication to the Tivoli Access Manager user ID. For more information, see the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

**Registry attribute entitlement service configuration**

Complete the instructions in the following sections:

- “Step 1 — Determine the attributes to be added to the credential”
- “Step 2 — Define your use of the entitlement service”
- “Step 3 — Specify the attributes to be added to the credential” on page 217

**Step 1 — Determine the attributes to be added to the credential**

Each user attribute that you want to add to the user credential must be defined in a Tivoli Access Manager configuration file. Typically, this is done in the WebSEAL configuration file.

Go to the Tivoli Access Manager user registry (for example, an LDAP user registry). Make a list of the names of each user registry entry that you want the credential attributes entitlement service to extract from the registry and place into the user credential. You will need the user DN and group DN also.

**Step 2 — Define your use of the entitlement service**

1. Open the WebSEAL configuration file for editing. Declare a service ID and library name for the registry attribute entitlement service. The service ID is an arbitrary string that you can choose. For example, TAM_CRED_ATTRS_SVC.

   ```
   [aznapi-entitlement-services]
   TAM_CRED_ATTRS_SVC = azn_ent_cred_attrs
   ```

   Note that WebSEAL automatically takes the value azn_ent_cred_attrs and finds the corresponding shared library. For example, on Solaris, libazn_ent_cred_attrs.so

2. Add an authorization API service definition entry to specify your usage of the entitlement service. Add the entry in the [aznapi-configuration]stanza.

   The entry must use the keyword cred-attribute-entitlement-services. The value for this entry must be the service ID you selected previously, such as TAM_CRED_ATTRS_SVC. For example:

   ```
   [aznapi-configuration ]
   cred-attribute-entitlement-services = TAM_CRED_ATTRS_SVC
   ```

**Note:** For more information on configuration of credential attribute entitlement services, see the IBM Tivoli Access Manager for e-business Authorization C API Developer Reference and review the sample configuration file, aznapi.conf. This configuration file is included in the Tivoli Access Manager authorization application development kit (PDAuthADK).
Step 3 — Specify the attributes to be added to the credential

The attributes to be added to the credential are configured in several stanzas. Add this information to the WebSEAL configuration file.

Note: Alternatively, you can define the attributes in a separate file, to be called by the entitlement service. For more information, see the IBM Tivoli Access Manager for e-business Authorization C API Developer Reference.

Review the following example entry.

```
[TAM_CRED_ATTRS_SVC]
eperson = azn_cred Registry_id
group = cn=enterprise, o=tivoli

[TAM_CRED_ATTRS_SVC:eperson]
tagvalue_credattrs_lastname = sn
tagvalue_credattrs_employeetype = employeetype
tagvalue_credattrs_address = homepostaladdress
tagvalue_credattrs_email = mail

[TAM_CRED_ATTRS_SVC:group]
tagvalue_credattrs_businesscategory = businesscategory
```

The stanza name [TAM_CRED_ATTRS_SVC] is the Service ID. Inside this stanza are sources of attributes to be retrieved. The source names, such as user and group are used to identify the source location in the registry. You need to define these. The values for these sources are registry identifiers that exist in the registry. The values can be existing credential attribute names. If this is the case, the service automatically finds and uses the respective values.

Configure the registry attributes for each of the sources under the service stanza in a separate stanza. The syntax of the separate stanza is the service ID library name followed by a colon (:) and then the source name. This connection is necessary because more than one service can be configured in the same file.

The configuration file entries contain mappings of user registry attributes to user-defined credential attributes.

For example, in an LDAP user registry, the DN for a user could be

```
cn=joeuser, o=tivoli
```

For this user, the LDAP user registry entries could be:

```
sn=Smith
employeetype=bankteller
homepostaladdress="3004 Mission St Santa Cruz CA 95060"
email=joeuser@bigco.com
```

For the group `cn=enterprise, o=tivoli`, the LDAP group registry entry could be:

```
businesscategory=finance
```

Using the example configuration entries shown above, the attribute list returned would have the following entries

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Attribute value</th>
</tr>
</thead>
<tbody>
<tr>
<td>credattrs_lastname</td>
<td>Smith</td>
</tr>
<tr>
<td>credattrs_employeetype</td>
<td>bankteller</td>
</tr>
<tr>
<td>credattrs_address</td>
<td>3004 Mission St Santa Cruz CA 95060</td>
</tr>
</tbody>
</table>
Note that the service, source, and attributes can be multi-valued. If you specify the same attribute name as a stanza entry keyword, then the attributes retrieved will be added as a multi-valued attribute even when they come from different sources.

For example, more than one entitlement service can be chained together. This enables values retrieved from one service to be used as input values for another service. Likewise, attributes can be retrieved from more than one DN in the user registry. Thus, using the example above, you could add values from multiple users (DNs) to one credattrs_businesscategory attribute, if you wanted a list of all the businesscategory entries for a group of users.

For example, if you want to build an attribute called myemployeeinfo to add to the credential, and you want this attribute to contain the last name and employee type of everyone that authenticates, you could then define the following:

```
[myID]
source = azn_cred_authzn_id

[myID:source]
myemployeeinfo = lastname
myemployeeinfo = employee_type
```

### Junction handling of extended credential attributes

The user-defined credential information created in the previous section can be placed in an HTTP header of the request that is sent across a junction to a back-end server.

You must configure the junction to extract extended attribute data from the credential and insert the data into the HTTP header of the request. This functionality is achieved by setting a junction extended attribute, called **HTTP-Tag-Value**, on the junction object in the WebSEAL protected object space.

You use the `pdadmin object modify set attribute` command to set extended attributes on a junction object in the WebSEAL protected object space.

```
padmin> object modify object_name set attribute attr_name
          attr_value
```

The above command must be entered as one continuous command line.

An extended attribute (**attr_name**) enables the junction to perform a specific type of functionality. The **HTTP-Tag-Value** extended attribute instructs the junction to extract a particular value from a user’s credential and send the value to the back-end server in an HTTP header. The value of the **HTTP-Tag-Value** extended attribute uses the following format:

```
credential_extended_attribute_name = http_header_name
```

The `credential_extended_attribute_name` entry is the same as the attributed specified in the WebSEAL configuration file but without the “tagvalue_” prefix. The entry is not case-sensitive. The `http_header_name` entry specifies the name of the HTTP header used to deliver the data across the junction.

For example:

```
<table>
<thead>
<tr>
<th>credattrs_email</th>
<th><a href="mailto:joseuser@bigco.com">joseuser@bigco.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>credattrs_businesscategory</td>
<td>finance</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>credattrs_email</th>
<th><a href="mailto:joseuser@bigco.com">joseuser@bigco.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>credattrs_businesscategory</td>
<td>finance</td>
</tr>
</tbody>
</table>
When WebSEAL processes a user request to a back-end application server, it looks for any **HTTP-Tag-Value** attributes configured on the junction object.

In this example, the configured junction looks at the credential of the user making the request, extracts the value of the `tagvalue_ldap-employee-number` credential extended attribute, and places it in an HTTP header as:

```
employee-id:09876
```

In summary:

| Value of **HTTP-Tag-Value** attribute set on the junction object: | ldap-employee-number=employee-id |
| Attribute name and value as they appear in the user credential: | tagvalue_ldap-employee-number:09876 |
| HTTP header name and value: | employee-id:09876 |

If the back-end application is a CGI application, the CGI specification dictates that HTTP headers are made available to CGI programs as environment variables in the form:

```
HTTP_http_header_name
```

For example:

```
HTTP_employee-id=09876
```

Multiple user attribute data can be passed to the junctioned server in HTTP headers by using multiple `pdadmin object modify set attribute` commands to specify multiple **HTTP-Tag-Value** junction attributes (one attribute is specified per command).
Credential refresh

This section contains the following topics:

- “Credential refresh concepts”
- “Credential refresh configuration” on page 224
- “Credential refresh usage” on page 225

Credential refresh concepts

This section contains the following topics:

- “Credential refresh overview”
- “Credential refresh rules” on page 221
- “Refresh of cached credential information” on page 221
- “Configuration file syntax and usage” on page 222
- “Default settings for preserve and refresh” on page 223
- “Limitations” on page 223

Credential refresh overview

When a user authenticates to WebSEAL, the authentication process accesses the Tivoli Access Manager user registry and builds a credential for the user. The credential contains information about the user that is needed by Tivoli Access Manager to decide whether to grant the user access to the requested resource. An example of credential information is a list of groups to which the user belongs.

During a user session, changes in user information can take place. For example, the user may be added to a new group. When this occurs, there might be a need to update or refresh the contents of the user credential, to reflect the new user information. WebSEAL provides a mechanism to enable a credential refresh without requiring the user to log out and then reauthenticate.

You can control how the credential refresh feature occurs. WebSEAL provides configuration settings that enable you to specify credential attributes to refresh (update) and credential attributes to preserve (retain). This enables you to have precise control over how user credentials are manipulated during a user session.

Use of the credential refresh configuration settings can be important when the authentication process on your WebSEAL server includes call outs to mechanisms that provide additional or extended information about a user. These mechanisms include:

- Credential attribute entitlement service.
  This service is built into Tivoli Access Manager by default.
- Customized credential attribute entitlement service
  This service must be written by the application developer.
- Credential extended attributes external authentication module
  This authentication module must be written by the application developer.

For more information on the credential attribute services listed above, see “Mechanisms for adding registry attributes to a credential” on page 215.

When credential refresh occurs, the above services are handled as follows:

- The default credential attribute entitlement services is run.
- The customized credential attribute entitlement services is run.
• Credential extended attributes external authentication modules are not run.

The credential refresh configuration settings enables you to preserve attributes obtained during the initial use of an entitlement service. For example, if an attribute contained a timestamp for the start of the user session, you might want to preserve the timestamp even though the credential was refreshed.

The credential refresh configuration settings also enables you to preserve attributes obtained from a credential extended attribute authentication module. Since custom authentication modules are not run again during the rebuilding of the credential, you use the configuration file settings to specify attributes to be added to the new credential.

**Credential refresh rules**

Credential refresh involves the generation of a new credential for user identity, followed by an evaluation of the contents of the new credential against the contents of the old credential that was obtained during initial user authentication. The contents of the two credentials are combined into a merged credential according to the following rules:

1. When an attribute occurs in the new credential but not the old credential, it is added to the merged credential.

2. The following attributes are added to the merged credential based only on their value in the old credential. These attributes are used by the authorization API. They are not changed by values in the new credential.
   
   ```
   AZN_CRED_AUTHMECH_INFO
   AZN_CRED_BROWSER_INFO
   AZN_CRED_IP_ADDRESS
   AZN_CRED_PRINCIPAL_NAME
   AZN_CRED_AUTH_METHOD
   AZN_CRED_USER_INFO
   AZN_CRED_QOP_INFO
   ```

3. For each attribute in the old credential for which there is a corresponding attribute in the new credential, the following rules apply:
   • When there is an entry in the configuration file that matches it, the attribute in the merged credential is preserved or refreshed according to the value of the entry in the configuration file
   • When there is not an entry in the configuration file that matches it, the attribute in the merged credential is assigned the value from the new credential.

4. For each attribute in the old credential for which there is not a corresponding attribute in the new credential, the following rules apply:
   • When there is a configuration file entry for the attribute specifying refresh, the attribute is not added to the merged credential
   • When there is a configuration file entry for the attribute specifying preserve, the attribute is added to the merged credential
   • When the configuration file does not contain an entry for the attribute, the attribute is not added to the merged credential.

**Refresh of cached credential information**

Some user registries maintain cached information. Cached data is kept for a specific amount of time, and is then discarded. Once the cached data has expired, it is not reloaded into the cache until the next time the user registry is accessed. Thus, when changes are made to user registry data, the data is not immediately cached in memory. Likewise, when using a replicated LDAP user registry, the updates to the replicated registries do not occur immediately.
The default lifetime of data in the WebSEAL user cache is 30 seconds. This lifetime begins when the data first enters the cache, such as when a user first authenticates, or when the cached data has expired and WebSEAL contacts the registry to update the data. WebSEAL contacts the registry to update the data during a credential refresh event. The cached information is valid for 30 seconds after it is first obtained from the registry. After 30 seconds, any credential refresh operations go directly to the user registry. The access to the user registry also causes the user data to be reloaded into the cache.

The following example shows the algorithm for updating the user cache:
1. User authenticates at time auth_time.
2. User is added to a group at time auth_time + 120 seconds
3. User’s credential is refreshed at time auth_time + 130 seconds

Because the user cache data expired at time auth_time + 30 seconds, the new group membership will be added to the user’s credential.

Continuing with the same example:
1. User is then added to another group at time auth_time + 135 seconds
2. User’s credential is refreshed at time auth_time + 140 seconds

When the user credential is refreshed at auth_time + 140 seconds, it does not pick up the new group membership. This is because the user credential is built off cached user data when the cached user data is considered valid (has not expired). Since the user cache data was updated at time auth_time + 130 seconds, it is not scheduled to be updated until auth_time + 160 seconds. Thus, the administrator must wait until time auth_time + 160 seconds to run the refresh command. At that time, the user credential will pick up the new group memberships.

Configuration file syntax and usage
The credential refresh behavior is controlled by entries in the [credential-refresh-attributes] stanza in the WebSEAL configuration file. The format is:

attribute_name_pattern = {preserve|refresh}

The attribute name pattern is used to select a given set of attributes. Wild card matching is supported.

A particular attribute can possibly be matched by many different wild card patterns. Therefore, the order of elements in the configuration file is important. The first pattern that matches a given attribute is the only pattern that applies to that attribute.

Attribute names in attribute_name_pattern should not be case-sensitive because attribute names in credentials are not case-sensitive.

Example – Preserve all of the tag value attributes added by an extended attribute CDAS:
[credential-refresh-attributes]
tagvalue_* = preserve

Example – Update the tagvalue_last_refresh_time attribute with the value from the new credential, but preserve all other attributes that begin with tagvalue_:
[credential-refresh-attributes]
tagvalue_last_refresh_time = refresh
tagvalue_* = preserve
Note that the ordering of attributes in the file is important. In the following example, tagvalue_last_refresh_time will not be refreshed because it is first matched by the tagvalue_* entry, which is set to preserve:

```
[credential-refresh-attributes]
tagvalue_* = preserve
tagvalue_last_refresh_time = refresh
```

Avoid preserving attributes that begin with the letters AZN_. Such attributes are typically used internally by the authorization API during authorization decisions. These attributes are discussed in more detail in the *IBM Tivoli Access Manager for e-business Authorization C API Developer Reference*. In that book, see the discussion on obtaining attribute lists from credentials.

**Default settings for preserve and refresh**
The default settings in the WebSEAL configuration file are:

```
[credential-refresh-attributes]
authentication_level = preserve
tagvalue_* = preserve
```

These settings result in the following behavior:

- The user authentication level is preserved when credentials are refreshed. During a user session, the user authentication level can change when authentication strength policy (step-authentication) is applied. In most cases, you want to preserve the modified authentication level during a credential refresh.
  
If you do not want to preserve the authentication level, change the configuration file entry:
  ```
  authentication_level = refresh
  ```

- The tagvalue_* entry preserves all credential attributes whose name begins with the characters tagvalue_. Attributes with the prefix tagvalue_ are typically supplied by external authentication services (CDASs) that want to add user information to the credential. The prefix is needed to ensure that the credentials are included when WebSEAL inserts credential data into an HTTP header to send across a junction.

**Limitations**

- It is not possible to call extended attribute CDASs during credential refresh. When you have an attribute that must be able to be refreshed during a credential refresh, use the credentials attribute entitlement service to set the attribute, or use a credential refresh rule to preserve the attribute.
- It is not possible to avoid calling the credentials attribute entitlement service during credential refresh. When you have an attribute that should be set only once, during initial authentication, use an extended attribute CDAS to set the attribute.
Credential refresh configuration

To configure credential refresh, complete the following steps:

- **Step 1: Specify attributes to preserve or refresh**
- **Step 2: Enable user session IDs**
- **Step 3: Enable placement of server name into junction header**

### Step 1: Specify attributes to preserve or refresh

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file.
   - Add entries for attributes to preserve. For example:
     ```
     [credential-refresh-attributes]
     my_cred_attribute1 = preserve
     my_cred_attribute2 = preserve
     ```
   - Add entries to refresh:
     ```
     [credential-refresh-attributes]
     my_cred_attribute3 = refresh
     my_cred_attribute4 = refresh
     ```
   - When appropriate use the order of the entries to handle both specific entries and groups of entries. For example, to preserve the attribute `special_cred_attr1`, but refresh all other attributes with the naming construct of `special_cred_attr*`, add the following entries:
     ```
     [credential-refresh-attributes]
     special_cred_attr1 = preserve
     special_cred_attr* = refresh
     ```

### Step 2: Enable user session IDs

Ensure that user session IDs are enabled for the WebSEAL server instance. The credential refresh administration command does not work when user session IDs are not enabled.

```plaintext
[session]
user-session-ids = yes
```

### Step 3: Enable placement of server name into junction header

A header with the URI-encoded authorization API administration server name is passed to all junction servers. When no header name is specified, the header will not be sent to the junction.

The value is set in the default WebSEAL configuration file.

```plaintext
[header-names]
server-name = iv_server_name
```

This setting controls the name of the header used to pass the name of the server to junctioned applications. For example, when `server-name = iv_server_name`, and the WebSEAL server instance is `default-webseald-diamond.subnet1.ibm.com`, WebSEAL passes the following header to the junction:

`iv-server-name:default-webseald-diamond.subnet1.ibm.com`

Typically, the default value `iv_server_name` is used. However, you can replace it with any valid string. Valid strings are limited to the following characters: [A-Z], [a-z], [0-9], hyphen (-), or underscore (_).

WebSEAL accepts a blank value for `server-name`. This is not usually useful, but could be applicable if the junctioned application chooses to have the server name hardcoded instead of obtaining it from the header.
1. Ensure that the server-name key is set in the configuration file for the WebSEAL server instance.
2. Restart the WebSEAL server.

**Credential refresh usage**

Topic index:
- "Refresh credentials for a specified user"
- "Troubleshooting"

**Refresh credentials for a specified user**

Send a command to the WebSEAL server, instructing it to perform a credential refresh operation for all of the sessions of the specified user on the WebSEAL server. The syntax is:

```
pdadmin> server task instance_name-webseald-host_name \ 
      refresh all_sessions user_name
```

Enter the above command as one continuous command line.

To obtain the server name in the correct format, use the `pdadmin server list` command. Then enter the `pdadmin` command to refresh all sessions. For example, when logged in to `pdadmin` as the administrative user `sec_master`:

```
pdadmin sec_master> server list
default-webseald-diamond.subnet1.ibm.com
default-webseald-cmd
pdadmin sec_master> server task default-webseald-diamond.subnet1.ibm.com \ 
      refresh all_sessions brian
DPWMA2043I  The user's credential was updated.
```

Note that the pdadmin server task command must each be entered as one continuous command line.

A warning message is returned if the user is not logged in to the WebSEAL server.

**Usage notes:**

- Configure credential refresh for WebSEAL before using this `pdadmin` command. See "Credential refresh configuration" on page 224.
- You must issue a separate pdadmin command for each user whose credentials are to be refreshed. You cannot refresh credentials for more than one user at a time.
- The user invoking this command must have server admin (The s ACL bit) permission on the `/WebSEAL/hostname_instance_name` server object. This prevents unauthorized users from performing credential refresh operations.

Note that the name of the `hostname_instance_name` server object is different than the server name. To determine the exact name of the server object, use `pdadmin object list`. For example, when logged in to `pdadmin` as the administrative user `sec_master`:

```
pdadmin sec_master> object list /WebSEAL
/WebSEAL/cmd-default
/WebSEAL/diamond.subnet1.ibm.com-default
```

**Troubleshooting**

- **Problem**: When a new group entry is added to a user’s information in a user registry, a credential refresh command does not obtain the new entry.
- **Solution**: Some user registries maintain cached information. The cache is updated periodically. The cache update must take place before the credential
refresh can succeed. Likewise, when using a replicated LDAP user registry, the updates to the replicated registries do not occur immediately. Wait 30 seconds and try credential refresh again. For more information, see “Refresh of cached credential information” on page 221.
Chapter 8. WebSEAL Key Management

This chapter contains information that describes tasks you can perform to manage certificate handling by the WebSEAL server.

Topic Index:
- “WebSEAL Key Management Overview” on page 227
- “Managing client-side and server-side certificates” on page 228

WebSEAL Key Management Overview

The following diagram summarizes the key management configuration required for SSL communication between WebSEAL and other components of the Tivoli Access Manager domain. Configuration stanzas and parameters are located in the WebSEAL configuration file. The GSKit iKeyman utility is used to create key database files and manage the digital certificates stored in these key database files.

![Diagram of key management parameters]

Figure 7. Keyfile management parameters
Managing client-side and server-side certificates

This section describes the administration and configuration tasks required to set up WebSEAL to handle client-side and server-side digital certificates used for authentication over SSL.

WebSEAL requires certificates for the following situations:
- WebSEAL identifies itself to SSL clients with its server-side certificate
- WebSEAL identifies itself to a junctioned back-end server (configured for mutual authentication) with a client-side certificate
- WebSEAL refers to its database of Certificate Authority (CA) root certificates to validate clients accessing with client-side certificates
- WebSEAL refers to its database of Certificate Authority (CA) root certificates to validate junctioned back-end servers

WebSEAL uses the IBM Global Security Kit (GSKit) implementation of SSL to configure and administer digital certificates. GSKit provides the iKeyman utility to set up and manage the certificate key database that contains one or more WebSEAL server/client certificates and the CA root certificates.

WebSEAL includes the following components at installation to support SSL authentication using digital certificates:
- A default key database (pdsrv.kdb)
- A default key database stash file (pdsrv.sth) and password ("pdsrv")
- Several common CA root certificates
- A self-signed test certificate that WebSEAL can use to identify itself to SSL clients

It is recommended that you apply for a commonly recognized certificate from a known Certificate Authority to replace this test certificate.

Configuration for WebSEAL certificate handling includes:
- “Configuring WebSEAL key database parameters” on page 229
- “Using the iKeyman certificate management utility” on page 230
- “Configuring CRL checking” on page 231

GSKit key database file types

The IBM Key Management tool (iKeyman) uses several file types that are summarized in the following table.

A CMS key database consists of a file with the extension .kdb and possibly two or more other files. The .kdb file is created when you create a new key database. A key record in a .kdb file can be either a certificate or a certificate with its encrypted private key information.

The .rdb and .crl files are created when you create a new certificate request. The .rdb file is required throughout the CA certificate request process.

<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.kdb</td>
<td>The &quot;key database&quot; file. Stores personal certificates, personal certificate requests, and signer certificates. For example, the default WebSEAL key database file is pdsrv.kdb.</td>
</tr>
<tr>
<td>.sth</td>
<td>The &quot;stash&quot; file. Stores an obfuscated version of the key database password. The stem name of this file is the same as the associated .kdb file. Also stores private keys, if there are any.</td>
</tr>
<tr>
<td>File Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>.rdb</td>
<td>The &quot;request&quot; database file. Automatically created when you create a .kdb key database file. The stem name of this file is the same as the associated .kdb file. This file contains certificate requests that are outstanding and have not yet been received back from the CA. When a certificate is returned from the CA, the .rdb file is searched for the matching certificate request (based on the public key). If a match is found, the certificate is received and the corresponding certificate request is deleted from the .rdb file. If a match is not found, the attempt to receive the certificate is rejected. Included in the certificate request is the common name, organization, street address, and other information that was specified at the time of the request, as well as the public and private key associated with the request.</td>
</tr>
<tr>
<td>.crl</td>
<td>The “certificate revocation list” file. This file normally contains the list of certificates that have been revoked for one reason or another. However, iKeyman does not provide any support for certificate revocation lists, so it is empty.</td>
</tr>
<tr>
<td>.arm</td>
<td>An ASCII encoded binary file. A .arm file contains a base-64 encoded ASCII representation of a certificate, including its public key, but not its private key. The original binary certificate data is transformed into an ASCII representation. When a user receives a certificate in a .arm file, iKeyman decodes the ASCII representation and places the binary representation into the appropriate .kdb file. Similarly, when a user extracts a certificate from a .kdb file, iKeyman converts the data from binary to ASCII and places it in a .arm file. Note: Any file extension is acceptable to use (other than .arm), as long as the file itself is a Base64 encoded file.</td>
</tr>
<tr>
<td>.der</td>
<td>The &quot;Distinguished Encoding Rules&quot; file. A .der file contains a binary representation of a certificate, including its public key, but not its private key. It is very similar to a .arm file, except that the representation is binary, not ASCII.</td>
</tr>
<tr>
<td>.p12</td>
<td>The &quot;PKCS 12&quot; file, where PKCS stands for &quot;Public-Key Cryptography Standards&quot;. A .p12 file contains a binary representation of a certificate, including both its public and private keys. A .p12 file may also include more than one certificate; for example, a certificate chain. Because a .p12 file contains a private key, it is password protected.</td>
</tr>
</tbody>
</table>

### Configuring WebSEAL key database parameters

**WebSEAL Key Database File:**

At installation, WebSEAL provides a default certificate key database. The `webseal-cert-keyfile` parameter, located in the `[ssl] stanza of the webseald.conf configuration file, identifies the name and location of this file:

```
[ssl]
webseal-cert-keyfile = /var/pdweb/www/certs/pdsrv.kdb
```

You can use the iKeyman utility to create a new key database. However, you must enter the name and location of this new key file in the `webseal-cert-keyfile` parameter so that WebSEAL can find and use the certificates contained in that database.

**Key Database File Password:**

At installation, WebSEAL also provides a default stash file that contains the password for the pdsrv.kdb key file. The `webseal-cert-keyfile-stash` parameter informs WebSEAL of the location of the stash file:

```
webseal-cert-keyfile-stash = /var/pdweb/www/certs/pdsrv.sth
```

The default password encrypted in this stash file is “pdsrv”. You can also express a password as plain text in the `webseal-cert-keyfile-pwd` parameter. For example:

```
webseal-cert-keyfile-pwd = pdsrv
```
At installation, WebSEAL uses the stash file to obtain the key file password. The `webseal-cert-keyfile-pwd` is commented out. By using the stash file you can avoid displaying the password as text in the `webseald.conf` configuration file.

**Note:** Uncomment the specific password parameter you want to use. If both password and stash file are specified, the password value is used.

### WebSEAL Test Certificate:

At installation, WebSEAL provides a non-secure self-signed test certificate. The test certificate, acting as a server-side certificate, allows WebSEAL to identify itself to SSL clients.

To better control how this test certificate is used, the certificate is not installed as a default certificate. Instead, the `webseal-cert-keyfile-label` parameter designates the certificate as the active server-side certificate and overrides any other certificate designated as "default" in the keyfile database.

```
webseal-cert-keyfile-label = WebSEAL-Test-Only
```

**Note:** WebSEAL uses GSKit certificate handling functionality. GSKit allows but does not require that a certificate in keyfile databases be designated the default certificate. For more information on certificate handling, see the GSKit document: *Secure Socket Layer and iKeyman User’s Guide*.

Although this test certificate allows WebSEAL to respond to an SSL-enabled browser request, it cannot be verified by the browser (which does not contain an appropriate root CA certificate). Because the private key for this default certificate is contained in every WebSEAL distribution, this certificate offers no true secure communication.

You must use the iKeyman utility to generate a certificate request that can be sent to a Certificate Authority (CA). Use iKeyman to install and label the returned server certificate.

If you use different certificates for other scenarios (such as `-K` junctions), you can use the iKeyman utility to create, install, and label these certificates. The keyfile label must not contain spaces.

WebSEAL (which by default runs as `user ivmgr`) must have read (r) permission on these key database files.

### Inter-server SSL Communication for Tivoli Access Manager:

The `[ssl]` stanza of the `webseald.conf` configuration file contains four additional parameters used to configure the keyfile used by WebSEAL for internal SSL communication with other Tivoli Access Manager servers. You should only modify these parameters through the `pdconfig` configuration script.

```
[ssl]
ssl-keyfile =
ssl-keyfile-pwd =
ssl-keyfile-stash =
ssl-keyfile-label =
```

### Using the iKeyman certificate management utility

The iKeyman utility is a tool provided with GSKit that you can use to manage digital certificates used by WebSEAL. Use iKeyman to:
• Create one or more key databases
• Change key database passwords
• Create new WebSEAL certificates
• Set a new default WebSEAL certificate
• Create a self-signed certificate for testing
• Request and receive CA root certificates
• Add certificates to and delete certificates from the database
• Copy certificates from one database to another

Refer to the Secure Sockets Layer Introduction and iKeyman User’s Guide for detailed information on using the iKeyman utility.

Configuring CRL checking
The Certificate Revocation List (CRL) is a method of preventing the validation of unwanted certificates. The CRL contains the identities of certificates that are deemed untrustworthy. The GSKit implementation of SSL used by WebSEAL supports CRL checking. GSKit allows WebSEAL to perform CRL checking on client-side certificates and certificates from SSL junctions.

WebSEAL must know the location of this list in order to perform CRL checking. Parameters for the location of the LDAP server that can be referenced for CRL checking during client-side certificate authentication are found in the [ssl] stanza of the webseald.conf configuration file:

```
[ssl]
#crl-ldap-server = <server-name>
#crl-ldap-server-port = <port-id>
#crl-ldap-user = <webseal-admin-name>
#crl-ldap-user-password = <admin-password>
```

Parameters for the location of the LDAP server that can be referenced for CRL checking during authentication across SSL junctions are found in the [junction] stanza of the webseald.conf configuration file:

```
[junction]
#crl-ldap-server = <server-name>
#crl-ldap-server-port = <port-id>
#crl-ldap-user = <webseal-admin-name>
#crl-ldap-user-password = <admin-password>
```

By default, CRL checking is disabled (parameters are commented out). To enable CRL checking during certificate authentication, uncomment each parameter and enter the appropriate values.

A null value for the crl-ldap-user indicates that the SSL authentication mechanism should bind to the LDAP server as an anonymous user.

Configuring the CRL cache
GSKit allows WebSEAL to perform CRL checking on client-side certificates and certificates from SSL junctions. To improve CRL checking performance, you can cache the CRL from a particular Certificate Authority (CA). Subsequent CRL checks are made against this cached version of the list.

The settings for the two webseald.conf configuration file parameters discussed in this section are passed directly to the GSKit utility. For further information about GSKit functionality, refer to the GSKit documentation.
Setting the maximum number of cache entries
The gsk-crl-cache-size parameter specifies the maximum number of entries in the GSKit CRL cache. Each entry represents an entire CRL for a particular Certificate Authority. The default setting is "0". A value greater than "0" is required to activate the cache. When gsk-crl-cache-size and gsk-crl-cache-entry-lifetime are both set to "0" (default), CRL caching is disabled.

[ssl]
gsk-crl-cache-size = 0

Setting the GSKit cache lifetime timeout value
The gsk-crl-cache-entry-lifetime parameter specifies the lifetime timeout value for all entries in the GSKit CRL cache. The value is expressed in seconds and can have a range of 0-86400 seconds. When gsk-crl-cache-size and gsk-crl-cache-entry-lifetime are both set to "0" (default), CRL caching is disabled.

[ssl]
gsk-crl-cache-size = 0
Chapter 9. Client single sign-on solutions

When WebSEAL is implemented as a proxy server to provide protection to a secure domain, there is often a requirement to provide solutions for single sign-on to resources across unique domains and servers.

Topic Index:
- “Windows desktop single sign-on” on page 234
- “Cross-domain single sign-on” on page 245
- “e-community single sign-on” on page 256
Windows desktop single sign-on

This section contains the following topics:

- "Windows desktop single sign-on concepts"
- "Windows desktop single sign-on configuration" on page 237

Windows desktop single sign-on concepts

This section discusses the following topics:

- "SPNEGO protocol and Kerberos authentication"
- "User registry and platform support" on page 235
- "Compatibility with other authentication methods" on page 235
- "Limitations" on page 236

SPNEGO protocol and Kerberos authentication

Microsoft provides an authentication solution that allows Windows clients to use Microsoft Internet Explorer (IE) to access resources on Microsoft Internet Information Servers (IIS) without having to reauthenticate. This single sign-on solution relies on proprietary Microsoft HTTP authentication mechanisms. IBM Tivoli Access Manager WebSEAL provides an equivalent authentication solution that enables IE clients to access WebSEAL servers without having to reauthenticate.

This means that users with an IE browser can access resources protected by Tivoli Access Manager without having to reenter their username and password. The user need only login once to the Windows domain, as is typically done when logging in to Windows on a desktop workstation.

WebSEAL supplies an implementation of same HTTP authentication method used by Microsoft. This implementation involves two pieces:

- Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)
- Kerberos authentication

The SPNEGO protocol mechanism enables WebSEAL to negotiate with the browser to establish the authentication mechanism to use. The browser supplies Kerberos authentication information. WebSEAL knows how to use the user’s Kerberos authentication information when processing a user request to access resources protected by Tivoli Access Manager.

On WebSEAL, this implementation is called Windows desktop single sign-on.

Deployment of this single sign-on solution requires enabling and configuring the SPNEGO protocol on the WebSEAL server. In addition, the WebSEAL server must be configured as a client into an Active Directory domain, and must have connectivity to an Active Directory domain controller. The Active Directory domain controller must act as a Kerberos Key Distribution Center (KDC). WebSEAL servers running on UNIX systems must use the Active Directory domain controller as their Kerberos KDC.

The WebSEAL configuration steps vary depending on the operating system platform and type of Tivoli Access Manager user registry.

Note: Use of SPNEGO requires that a time synchronization service be deployed across the Active Directory server, the WebSEAL server, and any clients (browsers) that will authenticate using SPNEGO.
WebSEAL and IIS handle session management differently. IIS maintains session state with clients by reauthenticating each new TCP connection using the SPNEGO protocol. SPNEGO and Kerberos are both designed for secure authentication over insecure networks. In other words, they are supposed to provide for secure authentication even when using an insecure transport such as HTTP.

The IIS method of maintaining session state can potentially have an adverse effect on performance. WebSEAL avoids this problem by using different session state methods. The WebSEAL session state methods are based on a security model that expects WebSEAL to be deployed either over a secure network or using a secure transport such as SSL. WebSEAL optimizes performance by maintaining state using SSL session IDs or HTTP cookies. Also, WebSEAL provides a scalable, secure environment by supporting junctions between WebSEAL and backend servers. Thus, single signon solutions using SPNEGO to WebSEAL should only be deployed over a secure network or over a secure transport such as SSL.

**User registry and platform support**

WebSEAL SPNEGO support provides single-sign-on from Internet Explorer running on Windows workstations configured into Active Directory domain to WebSEAL.

WebSEAL provides SPNEGO support for use with the following user registries:
- IBM Directory Services (LDAP)
- SunOne LDAP
- Microsoft Active Directory

The above user registries are supported on the following operating system releases:
- IBM AIX 5.1 and 5.2
- Windows 2000 Advanced Server
- Windows 2000 Server
- Sun Solaris Operating Environment 8 and 9

When Active Directory is not the Tivoli Access Manager user registry, users must be replicated between the Active Directory registry and the Access Manager user registry.

Supported Windows client platforms:
- Windows 2000 SP2 (or greater)
- Windows XP
- Internet Explorer 5.0.1 or greater
- Internet Explorer 5.5 SP2
- Internet Explorer 6.0 SP1 (on Windows 2000)

Internet Explorer must be configured to participate in the Windows desktop single signon solution.

**Compatibility with other authentication methods**

WebSEAL support for SPNEGO authentication is compatible with the following WebSEAL authentication methods:
- Basic authentication
- Forms
- HTTP header
Failover authentication
The failover cookie failover mechanism supports SPNEGO authenticated users

Cross domain single signon

SSL certificate authentication

Switch user authentication
Switching user identity to the SPNEGO authenticated user is supported.

When SPNEGO is configured along with another authentication method, WebSEAL simultaneously sends both an SPNEGO challenge and an HTML form login back to the browser. Browsers which support SPNEGO respond with SPNEGO authentication. Browsers that do not support SPNEGO display the login form.

Compatibility between SPNEGO authentication and WebSEAL e-community single signon is limited. A WebSEAL server can be an e-community master authentication server (MAS) and support SPNEGO. However, a WebSEAL server cannot be an e-community slave and also support SPNEGO.

WebSEAL authentication strength policy (step-up authentication) from SPNEGO authentication to other authentication methods is supported.

When SPNEGO authentication is enabled, only the following methods of maintaining session state are supported:

- SSL session IDs
- HTTP cookies

SPNEGO authentication is compatible with the automatic tag-value retrieval support provided by the Tivoli Access Manager entitlements service. Thus, it is possible to add extended attributes to a user’s credential after the user has authenticated with SPNEGO.

Limitations
The following WebSEAL features are not supported with SPNEGO authentication:

- POP or session timer based reauthentication of SPNEGO authenticated clients.
- Password change using pkmspasswd.
- Mapping of a username through a CDAS.
- Use of a customized credential extended attribute CDAS to add extended attributes to a user credential.
- SPNEGO clients cannot log out of WebSEAL. Clients must log out from the workstation. Clients that access WebSEAL pkms command pages (excepting switch user) receive the PKMS help page.
- Reauthentication when the inactive session timer expires for SPNEGO clients.
  The user cache entry is deleted but the session ID is retained. Information in the header received from the SPNEGO client is used to reauthenticate. The client does not have to log in again, but the client receives a new session cache entry.
  - Reauthentication when a user accesses an object with a reauthentication policy attached
    In this case access is denied, and user receives a message stating that reauthentication is required.

The following limitation also applies:
- Microsoft NT LAN Manager (NTLM) authentication is not supported.
However, the Tivoli Access Manager Web Plug-in for IIS supports NTLM. WebSEAL can be deployed in an e-community single signon solution that uses the Web Plug-in for IIS to accomplish SPNEGO authentication using NTLM.

- Using SPNEGO simultaneously with other authentication methods does not work with Netscape 4.7

Windows desktop single sign-on configuration

This section contains the configuration steps that must be completed to implement Windows desktop single signon using SPNEGO authentication with WebSEAL. Not all steps are required on each platform.

To configure SPNEGO authentication, complete each of the following steps:

- “Step 1: Configure WebSEAL server into Active Directory domain”
- “Step 2: Map Kerberos principal to Active Directory user” on page 238
- “Step 3: Install Kerberos runtime client (UNIX only)” on page 240
- “Step 4: Configure Kerberos client (UNIX only)” on page 240
- “Step 5: Verify authentication of Web server principal (UNIX only)” on page 241
- “Step 6: Verify WebSEAL authentication using the keytab file (UNIX only)” on page 242
- “Step 7: Enable SPNEGO for WebSEAL” on page 242
- “Step 8: Add service name and keytab file entries (UNIX only)” on page 242
- “Step 9: Restart WebSEAL” on page 243
- “Step 10: Configure the Internet Explorer client” on page 243

Troubleshooting information:

- “Troubleshooting tips” on page 243

Step 1: Configure WebSEAL server into Active Directory domain

To participate in a Kerberos exchange with IE, a WebSEAL server must have an identity in the Active Directory Kerberos domain. This requires that WebSEAL be registered with the Active Directory domain controller. This enables the IE browser to obtain a Kerberos ticket from the Active Directory domain controller in order to access the WebSEAL server.

See the Microsoft documentation for instructions on how to add an identity for the WebSEAL server host into an Active Directory domain.

Notes:

- On Windows, the default WebSEAL server (first server instance) uses the Local Service Account identity when contacting the Active Directory domain controller. Additional WebSEAL server instances must be configured to use a different user account. The user account must match a unique Active Directory user for the WebSEAL server identity.
- On UNIX, ensure that the user name matches the host name of the WebSEAL server host. Do not use the full domain name. For example, for the system diamond.subnet2.ibm.com, create a user diamond. Do not require the user to change password at next log in. Do not set the password to expire.
- Ensure that DNS is configured properly for each of the participating machine. One way to confirm this is to run forward and reverse nslookup on each machine.

Multiple WebSEAL instances
Each WebSEAL instance must have a separate Kerberos identity

- On Windows, create a separate Windows administrative user for each instance, and modify the services configuration so that the instance is started under that user account.
- On UNIX, create a separate Active Directory user for each instance.

**Note:** Multiple WebSEAL instances are supported with SPNEGO when each WebSEAL server has a unique IP address and host name. Multiple instances are not supported when the instances listen on different ports but share the same IP address.

**Step 2: Map Kerberos principal to Active Directory user**

The IE client request to the Active Directory domain controller requests access to a Kerberos principal of name:

```
HTTP/DNS_name_of_WebSEAL_server@Active_Directory_domain_name
```

This name must be mapped to the Active Directory user that represents the WebSEAL server instance, as created in Step 1 above.

This mapping requires the Windows `ktpass` utility. The `ktpass` utility might not be loaded on the Windows system by default. It can be obtained from the Windows Support Tools package on the Windows CDs.

**Windows**

The configuration steps differ between configuring the first (default) WebSEAL server and configuring multiple server instances. Use the appropriate instructions that follow.

Default (first) WebSEAL server:

1. Register the service principal name for the WebSEAL server. On the Active Directory domain controller, run the `ktpass` command. For example, when the WebSEAL host is `diamond.subnet2.ibm.com`, and the Active Directory domain is `IBM.COM`, the command is:

   ```
ktpass -princ HTTP/diamond.subnet2.ibm.com@IBM.COM -mapuser diamond$
```

   **Note:** The value to the `-mapuser` option has a dollar sign ($) character at the end. The user with the dollar sign ($) represents the local service account.

Multiple WebSEAL server instances:

1. Register the service principal name for the WebSEAL server. On the Active Directory domain controller, run the `ktpass` command. For example, when the WebSEAL instance name is `web1`, host name is `web1.subnet2.ibm.com`, and the Active Directory domain is `IBM.COM`, the command is:

   ```
ktpass -princ HTTP/web1.subnet2.ibm.com@IBM.COM -mapuser web1
```

   **Note:** The user specified by `-mapuser` must match the user created in Step 1 above. The `-mapuser` option does not create a user account.

2. Modify the service for the instance so that it starts using the new user you just created. Open the Services control panel right click on the service for a new WebSEAL server, and select Properties. Select the Log On tab, and select the **This account** radio button. For the account name, enter the name of the account you just created. For example, `web1@subnet1.ibm.com`. Enter the password for the account.
3. Grant the new account Administrator privileges for the local machine. On the client machine, select:

Control Panels > Administrative Tools > Computer Management

Then select:

Local Users and Groups > Groups

Right click on Administrators and select Add To Group. Press the Add button.

4. Select the **Look in** dropdown menu. Find the name of the Active Directory domain. Find the user you just created. Double-click the user name. Press OK. Finish the configuration by pressing OK to each screen as needed.

**UNIX**

Complete the following steps:

1. On UNIX systems, in addition to mapping the user, you must create a keytab file for use when signing in to the Kerberos domain. The syntax is:

   `ktpass -princ DNS_name_of_WebSEAL_server@Active_Directory_domain_name \  
   -pass your_password -mapuser WebSEAL_server_instance \  
   -out full_path_to_keytab_file -mapOp option`

   Enter the above command as one continuous command line

   For example, given the following values:

<table>
<thead>
<tr>
<th>Configuration setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSEAL host system</td>
<td>diamond.subnet2.ibm.com</td>
</tr>
<tr>
<td>User identity for the WebSEAL server instance</td>
<td>diamond</td>
</tr>
<tr>
<td>Active Directory domain</td>
<td>IBM.COM</td>
</tr>
<tr>
<td>password</td>
<td>mypassword</td>
</tr>
<tr>
<td>Keytab file</td>
<td>C:\Program Files\Tivoli\PDWeb\keytab-diamond\diamond_HTTP.keytab</td>
</tr>
<tr>
<td>Suppress error message when Kerberos principal is already associated with the user ( -mapOp )</td>
<td>set</td>
</tr>
</tbody>
</table>

   the `ktpass` command would be:

   `ktpass -princ HTTP/diamond.subnet2.ibm.com@IBM.COM  
   -pass mypassword -mapuser diamond  
   -out C:\Program Files\Tivoli\PDWeb\keytab-diamond\diamond_HTTP.keytab  
   -mapOp set`

   Enter the above command as one continuous command line

   The user identity is the Active Directory user created in the previous step. The password specified here resets the password for the Active Directory user. A
highly secure password, such as a randomly generated password, is preferred. The location of the keytab file is arbitrary. Retain this password, for use in a later step to test your Kerberos configuration (when testing authentication from a UNIX machine to the Active Directory Key Distribution Center).

2. Transfer the keytab file to the UNIX system. Ensure that a secure transfer method is used. The recommended location is:
   
   /var/pdweb/keytab-instance_name/keytab_file_name

3. For best security practice, delete the keytab file from the Windows system.

4. On the UNIX system, assign ownership of the file to ivmgr, and restrict permissions on the keytab file so that only the owner can access it. For example:

   ```
   # chown ivmgr keytab_file
   # chgrp ivmgr keytab_file
   # chmod 600 keytab_file
   ```

5. For UNIX servers, repeat the above steps for each WebSEAL server instance.

**Step 3: Install Kerberos runtime client (UNIX only)**

The WebSEAL server system must have a Kerberos runtime installed. On Windows systems, the Kerberos runtime client is part of the operating system. No additional packages are required.

On UNIX systems, install the appropriate package:

- **AIX**
  
  IBM Network Authentication Service Client.
  
  This client can be found in the AIX expansion pack.

- **Solaris**
  
  - IBM Network Authentication Service Client.
    
    This client is included on the Tivoli Access Manager Web Security CD. Use pkgadd to install.
  
  - SUN Kerberos Client
    
    SUNWkr5cl. This is required by the IBM Network Authentication Service Client.
    
    This package is part of the SEAM package, and can be downloaded from the Sun Web site.

**Step 4: Configure Kerberos client (UNIX only)**

The Kerberos client installed in the previous step must be configured. This requires creation or modification of a Kerberos configuration file. On Solaris and AIX the file is:

```
/etc/krb5/krb5.conf
```

Complete the instructions that apply to your operating system:

**AIX**

Use the `mkkrb5clnt` utility. This utility creates and completes `/etc/krb5/krb5.conf`.

1. Run `mkkrb5clnt`. The syntax is:

   ```
   mkkrb5clnt -r Active_Directory_domain -c Active_Directory_controller_DNS -s Active_Directory_controller_DNS -d local_DNS_domain
   ```

   For example:

   ```
   mkkrb5clnt -r IBM.COM -c dc1.ibm.com -s dc1.ibm.com -d dns.com
   ```
2. Manually edit krb5.conf to remove a cryptographic setting not supported by Active Directory.

    [libdefaults]
    default_tkt_enctypes = des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des-cbc-md5 des-cbc-crc

    This step removes des3-cbc-sha1.

**Solaris**

Manually edit krb5.conf. Customize the following information for your domain:

- Realm
  
  For example: IBM.COM

- Active Directory controller server name
  
  For example, dc1.

- Domain name
  
  For example, ibm.com.

- DNS name
  
  For example, ibm.com.

Using the example values above, the contents of the Kerberos configuration file would include the following entries:

```
/etc/krb5/krb5.conf – partial listing
[libdefaults]
    default_realm = IBM.COM
    default_tkt_enctypes = des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des-cbc-md5 des-cbc-crc

[realms]
    IBM.COM
        kdc = dc1.ibm.com:88
        admin_server = dc1.ibm.com:749
        default_domain = ibm.com
    }

[domain_realm]
    dc1.ibm.com = IBM.COM
    .ibm.com = IBM.COM
```

The last line in the example file above ( .ibm.com = IBM.COM ) represents the DNS domain in which the WebSEAL server operates and to which user connect. Note the dot (.) in front of the IBM domain in the last line. This acts as a wildcard for all hosts in the ibm.com domain.

**Step 5: Verify authentication of Web server principal (UNIX only)**

Use the `kinit` program to verify that the Kerberos principal for the WebSEAL server can authenticate. Use the password. specified when you ran `ktpass` in Step 2:

```
# /usr/krb5/bin/kinit diamond@IBM.COM
Password for diamond@IBM.COM: server_password
# klist
```

You should see some output from `klist` showing the credentials for diamond@IBM.COM
Note: The location of the kinit utility might vary depending on the operating system platform.

Step 6: Verify WebSEAL authentication using the keytab file (UNIX only)
Verify that the WebSEAL server can authenticate using the keytab file created in Step 2. Enter the following kinit command as one continuous command line:

```
# kinit -k -t /var/pdweb/keytab-diamond/diamond_HTTP.keytab HTTP/diamond.subnet2.ibm.com@IBM.COM
# klist
```

You should see some output from klist showing the credentials for HTTP/diamond.subnet2.ibm.com@IBM.COM

Step 7: Enable SPNEGO for WebSEAL
Modify the WebSEAL configuration file to enable SPNEGO. Complete the following steps:
- Stop the WebSEAL server.
- Enable SPNEGO over SSL:

  ```
  [spnego]
  spnego-auth = https
  
  [authentication-mechanisms]
  kerberosv5 = fully_qualified_path to the library
  ```

- Specify the location of the Kerberos authentication library:

  Table 32. Kerberos authentication library location

<table>
<thead>
<tr>
<th>Platform</th>
<th>File location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win32</td>
<td>Tivoli_Access_Manager_install_dir\bin\stliauthn.dll</td>
</tr>
<tr>
<td>AIX</td>
<td>/opt/PolicyDirector/lib/libstliauthn.a</td>
</tr>
<tr>
<td>Solaris</td>
<td>/opt/PolicyDirector/lib/libstliauthn.so</td>
</tr>
</tbody>
</table>

  ```
  [authentication-mechanisms]
  kerberosv5 = fully_qualified_path to the library
  ```

Step 8: Add service name and keytab file entries (UNIX only)
Modify the WebSEAL configuration file to add the Kerberos service name and the location of the keytab file:

```
[spnego]
# UNIX ONLY
spnego-krb-service-name = HTTP@fully_qualified_hostname_of_webseal_server
# UNIX ONLY
spnego-krb-keytab-file = fully_qualified_keytab_path
```

For example:

```
[spnego]
# UNIX ONLY
spnego-krb-service-name = HTTP@diamond.subnet1.ibm.com
# UNIX ONLY
spnego-krb-keytab-file = /var/pdweb/keytab-diamond/diamond_HTTP.keytab
```

When configuring multiple WebSEAL server instances, be sure to enter the information for the instance. For example, using the prior example (in Step 2) for a server instance named web1, enter:
[spnego]
# UNIX ONLY
spnego-krb-service-name = HTTP@web1.subnet1.ibm.com
# UNIX ONLY
spnego-krb-keytab-file = /var/pdweb/keytab-web1/web1_HTTP.keytab

**Step 9: Restart WebSEAL**

UNIX

pdweb start

Windows

Use the Services Control Panel.

On Windows, WebSEAL *must* be running as a service for SPNEGO authentication to work properly. Otherwise, it runs using the identity of the logged in user.

**Step 10: Configure the Internet Explorer client**
The Internet Explorer client must be configured to use the SPNEGO protocol to negotiate authentication mechanisms. Consult the Microsoft Internet Explorer documentation for complete configuration instructions.

Notes:

- The IE browser must recognize the WebSEAL server as an *Intranet* or *Trusted* site. When this is not configured, the IE client does not automatically send user name and password to the server. The IE client should add the WebSEAL server to the *Intranet Sites* list, or add the WebSEAL server to the "Do not require a proxy for these sites" list.
- Internet Explorer 6 must be specifically configured to enable single sign-on. Use the menu item for Internet Options... and select the Advanced tab.
- The Windows client must use the correct DNS name to access the WebSEAL server. When an incorrect DNS name is used, IE may attempt to use NT LAN Manager (NTLM) protocol to contact WebSEAL. WebSEAL does not support NTLM.

**Troubleshooting tips**

**Kerberos configuration**

- **Problem:** When testing the keytab created for a UNIX server using kinit, you get the error "Clock skew too great while getting initial credentials."

  **Solution:** You must keep clocks synchronized when using Kerberos. For a permanent solution, deploy some kind of time synchronization service on your machines. For a temporary solution, adjust the clocks on the machines so they are within one minute of each other.

- **Problem:** When testing the keytab created for a UNIX server using kinit, you get the error "Preauthentication failed while getting initial credentials" or "Password incorrect while getting initial credentials".

  **Solution:** The key in the keytab file is incorrect. Make sure you generated the keytab file correctly, with the correct principal name, Active Directory user name, and path.

- **Problem:** kinit crashes when running kinit -k -t

  **Solution:** Some versions of kinit don’t deal properly with problems when an entry is not found in a keytab file. Double-check that the keytab file has the exact same entry you are passing to kinit.

**WebSEAL configuration**
When a problem occurs, consider enabling trace for SPNEGO. Add an entry to the routing file. The routing file is located under the installation directory, in etc/routing. Example entry:

```
bst:*:9:TEXTFILE:WebSEAL_installation_directory/log/spnegotrace.log
```

On UNIX, the default WebSEAL installation directory is /opt/pdweb. Substitute the path for your installation directory.

Stop and restart WebSEAL. Look for error messages in the trace file.

- **Problem:** The WebSEAL server will not start. The log file contains an error saying "Authentication method (kerberosv5) is not configured."

  **Solution:** Enable the kerberosv5 authentication method in the [authentication-mechanisms] stanza in the WebSEAL configuration file.

- **Problem:** The WebSEAL server will not start. The error message is "The security service function gss_import_name returned major error code 131072 and minor error code -1765328168."

  **Solution:** The principal name specified in the configuration file was invalid. It should have the form "HTTP@host_name" where host_name is the fully qualified DNS name of a computer which is configured into the Kerberos realm.

- **Problem:** The WebSEAL server does not start. The error message is: "The security service function gss_acquire_cred returned major error code 851968 and minor error code 39756033."

  **Solution:** The principal name in the configuration file does not match any of the keys in the specified keytab file. The keys in the keytab file have names like HTTP/host_name @REALM. The principal name should have the format HTTP/host_name

- **Problem:** The WebSEAL server does not start. The error message is "HPDST0129E The security service function gss_acquire_cred returned major error code 851968 and minor error code 486484225. (pd / bst)" or "HPDST0129E The security service function gss_acquire_cred returned major error code 851968 and minor error code 39756033. (pd / bst)"

  **Solution:** These errors are caused by DNS reverse lookup problems. Verify that your reverse lookup works properly.

- **Problem:** When a user attempts to access WebSEAL they receive an error saying "HPDIA0100E An internal error has occurred." The WebSEAL trace log file contains a message saying "The security service function gss_accept_sec_context returned major error code 851968 and minor error code -1765328347."

  **Solution:** the system clock on the client machine is out of sync with the system clock on the WebSEAL server. You must keep clocks synchronized when using Kerberos. For a permanent solution, deploy some kind of time synchronization service on your machines. For a temporary solution, adjust the clocks on the machines so they are within one minute of each other.
Cross-domain single sign-on

The Tivoli Access Manager Cross-Domain Single Sign-on (CDSSO) provides a default mechanism for transferring user credentials between unique servers and domains. CDSSO allows Web users to perform a single sign-on and move seamlessly between two separate secure domains when requesting a resource. The CDSSO authentication mechanism does not rely on a Master Authentication Server or MAS (see “e-community single sign-on” on page 256).

CDSSO supports the goals of scalable network architecture by allowing the integration of multiple secure domains. For example, a large corporate extranet can be set up with two or more unique domains—each with its own users and object space. CDSSO allows movement of users between the domains with a single sign-on.

When a user makes a request to a resource located in another domain, the CDSSO mechanism transfers an encrypted user identity token from the first domain to the second domain. The identity information in this token indicates to the receiving domain that the user is successfully authenticated in the first domain. The identity does not contain password information. The receiving server uses this token to build credentials in its own cache for that user. The user is not forced to perform an additional login.

Customizing single sign-on authentication

Cross-domain single sign-on solutions employ authentication tokens that convey an encoded version of the user identity to the destination server. The construction of these tokens by the initial server is called "token creation." The decoding and use of the token by the destination server is called "token consumption." You can create custom token create and consume libraries to meet the specific requirements of your network and Tivoli Access Manager implementation. Complete information and API reference material for cross-domain external authentication can be found in the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

In many CDSSO scenarios, the default one-to-one mapping between users in different domains might not suit all deployment requirements.

The cross-domain mapping frameWork (CDMF) is a programming interface that allows you to build a custom shared library that can handle extended user attributes and provide mapping services for the user identity.

The CDMF programming interface allows flexibility in customizing the mapping of user identities and the handling of user attributes. Complete information and API reference material for CDMF can be found in the IBM Tivoli Access Manager for e-business Web Security Developer Reference.

Authentication process flow for CDSSO with CDMF

The following process flow description is illustrated in Figure 8.

1. Any user who wants to participate in multiple domains must have a valid user account in the initial domain and an identity that can be mapped into a valid account in each of the participating remote domains.
   A user cannot invoke the CDSSO functionality without initially authenticating to an initial secure domain (A) that contains the user’s account.
2. The user makes a request to access a resource in domain B via a custom link on a Web page.
The link contains a special CDSSO management page expression:

/pkmscdsso?<destination-URL>

For example:

3. The request is first processed by the WebSEAL server in domain A. The websealA server builds an authentication token that contains the user’s credentials, including the Tivoli Access Manager identity (short name), the current domain (“A”), additional user information, and a time stamp. This process is performed by the “token create” function of the built-in single sign-on authentication mechanism (library).

The additional user information (extended attributes) can be obtained by a call out to the customized CDMF shared library (cdmf_get_usr_attributes). This library has the ability to supply user attributes that can be used by domain B during the user mapping process.

WebSEAL triple-DES encrypts this token data with the symmetric key generated by the cdssos_key_gen utility. This key file is shared and specified in the [cdsso-peers] stanza of the WebSEAL configuration file on both domain A and domain B WebSEAL servers.

The token contains a configurable time stamp (auth_token-lifetime) that defines the lifetime of the token. The time stamp, when properly configured, can prevent replay attacks.

The token is contained in a redirected request to the destination server, using the URL contained in the pkmscdsso link. For example:

4. The websealA server redirects the request containing the encrypted token back to the browser and then to the websealB server (HTTP redirection).

5. The websealB server decodes and validates the token as coming from the referring domain. This process is performed by the “token consume” function of the built-in single sign-on authentication mechanism (library).

6. The token consume functionality can further call out to a customized CDMF library which performs the actual user mapping (cdmf_map_usr). The CDMF library passes the user’s identity, and any extended attribute information, back to the token consume library. The token consume library uses this information to build a credential.

7. The websealB authorization service permits or denies access to protected objects based on the user’s credential and the specific ACL permissions associated with the requested objects.
Configuring CDSSO authentication

CDSSO conditions and requirements

- All WebSEAL servers participating in CDSSO must have machine times synchronized. Authentication between servers can fail when machine time differences are too great.
- For CDSSO to function successfully, each participating WebSEAL server must reveal its fully qualified host name to the other participating servers in the cross-domain environment. If any host name does not include a domain, CDSSO cannot be enabled and an error message is logged in msg_webseal1d.log. When setting up a CDSSO environment, ensure that the machine-specific networking setup for each participating server is configured to identify the server with a fully qualified host name.
- Because some WebSEAL configuration requires machine host names to be described as fully qualified host names, you must ensure that your system and network can resolve machine names into fully qualified host names. For example, using fully qualified host names allows for many host names (IP addresses) per machine, as in the case of multiple WebSEAL instances.

Resolving machine names
CDSSO can be disabled upon WebSEAL startup because the machine itself is not adequately configured to resolve machine names. The machine on which WebSEAL resides needs to be able to fully resolve an IP address. Because this functionality is very operating system-specific, it is not the role of this document to provide instructions. Always consult your system administrator if you are not sure your system has the proper capabilities.

The following general Solaris-specific information is provided only as an example:

Goal: Configure the machine to first look to DNS before checking the local /etc/hosts file for DNS information.

1. Make sure that /etc/resolv.conf has valid DNS server entries.
2. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:
   - hosts
dns
   files

  Alternative goal: Configure the machine to first use local DNS information (/etc/hosts) before checking DNS.

  1. Configure the machine to check /etc/hosts before looking to DNS. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:
     - hosts
dns
   files

  2. Enter appropriate DNS information in /etc/hosts:
     - webseal1.fully.qualified.com 1.11.111.111
     - webseal2.fully.qualified.com 2.22.222.222

  The following general Windows-specific information is provided only as an example:

  1. Use DNS and specify 2 IP addresses:
     - Network Connections > LAN > Properties > TCP/IP

  2. Specify a valid DNS server under the Advanced settings:
     - Network Connections > LAN > Properties > TCP/IP > Advanced > DNS > Add...

  3. In this same window, specify the primary DNS suffix for this connection:
     - Network Connections > LAN > Properties > TCP/IP > Advanced > DNS > Add...

  4. In your system properties, specify the computer name and its DNS suffix:
     - My Computer > Properties > Network ID > Properties > Computer name
     - My Computer > Properties > Network ID > Properties > More > Primary DNS suffix

  **CDSSO configuration summary**

  The following configuration steps are explained in detail in the remaining sections of this CDSSO chapter division.

  **Configuring default CDSSO token create functionality**

  1. Enable WebSEAL to generate CDSSO tokens ([cdsso-create]).

  2. Configure the built-in single sign-on authentication mechanism (library) for token create ([sso-create]).

  3. Create the key file used to encode and decode the token. Copy the key file to all appropriate participating servers ([cdsso-peers] stanza).

  4. Configure the token time stamp ([authtoken-lifetime])

  5. Configure the token label parameter ([cdsso-argument]).

  **Configuring default CDSSO token consume functionality**

  1. Enable WebSEAL to consume CDSSO tokens ([cdsso-auth]) for authentication.

  2. Configure the built-in single sign-on authentication mechanism (library) for token consume ([sso-consume]).

  3. Assign the appropriate key file ([cdsso-peers] stanza).

  4. Configure the token time stamp ([authtoken-lifetime])

  5. Configure the token label parameter ([cdsso-argument]).
1. Enabling and disabling CDSSO authentication

To enable CDSSO authentication, specify values for two settings in the WebSEAL configuration file. There is a setting for the token create (sso-create) library and a separate setting for the token consume (sso-consume) library.

To enable the token create library, set the following entry:

```
[cdsso]
cdsso-create = {none|http|https|both}
```

The value both specifies both HTTP and HTTPS protocols. The value none disables the token create library.

To enable the token consume library, (this means that the server will accept CDSSO tokens for authentication) set the following entry:

```
[cdsso]
cdsso-auth = {none|http|https|both}
```

The value both specifies both HTTP and HTTPS protocols. The value none disables the token consume library.

**Note:** You must stop and restart the WebSEAL server in order to activate changes to the WebSEAL configuration file. Complete all of the applicable configuration steps in this section and then restart WebSEAL.

2. Configuring the single sign-on authentication mechanism

The default CDSSO configuration requires that you enable the sso-create and sso-consume single sign-on authentication mechanisms. The sso-create mechanism is required by the initial WebSEAL server for creating the CDSSO token and building the redirected request. The sso-consume mechanism is required by the receiving WebSEAL server to decode the token and build the user credentials from the identity information contained in the token. For the default CDSSO configuration, each parameter specifies a built-in CDSSO token library file. One library contains the code for the token create functionality and the other library contains the code for the token consume functionality.

- On UNIX, the library files are called libssocreate. {so | a | sl} and libssoconsume. {so | a | sl}.
- On Windows, the library files are DLL files called ssocreate.dll and ssoconsume.dll.

<table>
<thead>
<tr>
<th>Authentication Mechanism</th>
<th>Single Sign-on Token Library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solaris</td>
</tr>
<tr>
<td>sso-create</td>
<td>libssocreate.so</td>
</tr>
<tr>
<td>sso-consume</td>
<td>libssoconsume.so</td>
</tr>
</tbody>
</table>

You can configure the CDSSO authentication mechanism by specifying the sso-create and sso-consume parameters with the full path name of the appropriate platform-specific default library file in the [authentication-mechanism] stanza of the WebSEAL configuration file.

For example:

**Solaris:**
[authentication-mechanisms]
sso-create = /opt/pdwebrte/lib/libssocreate.so
sso-consume = /opt/pdwebrte/lib/libssosconsume.so

Windows:
[authentication-mechanisms]
sso-create = C:\Program Files\Tivoli\PDWebRTE\bin\ssocreate.dll
sso-consume = C:\Program Files\Tivoli\PDWebRTE\bin\ssoconsume.dll

3. Encrypting the authentication token data

WebSEAL must encrypt the authentication data placed in the token using a key generated by the \texttt{cdsso\_key\_gen} utility. You must "synchronize" this key by sharing the key file with each participating WebSEAL server in each participating domain. Each participating WebSEAL server in each domain needs to use the same key.

The generated key is a triple DES 192 bit key. You cannot specify a life span time on this key.

\textbf{Note:} The distribution of key files is not a part of the Tivoli Access Manager CDSSO process.

The \texttt{cdsso\_key\_gen} utility requires that you specify the location (absolute path name) of the key file when you run the utility. You must also use a full path name to run this utility:

\texttt{UNIX:}
\[ # /opt/pdwebrte/bin/cdsso\_key\_gen <keyfile\_location> \]

\texttt{Windows:}
\[ MSDOS> C:\Program Files\Tivoli\PDWebRTE\bin\cdsso\_key\_gen <keyfile\_location> \]

Enter this key file location in the \texttt{[cdsso\_peers]} stanza of the WebSEAL configuration file of the participating WebSEAL server in each domain. The format must include the fully qualified host name of the WebSEAL server and the absolute path name to the key file location:

\texttt{[cdsso\_peers]}
\[ <fully-qualified-host-name> = <full-keyfile-pathname> \]

\textbf{Configuration example for server websealA in domain A:}

\texttt{[cdsso\_peers]}
\[ websealB.domainB.com = <pathname>/A-B.key \]

This setting specifies what key websealA uses to encrypt a token destined for websealB in domain B.

\textbf{Configuration example for server websealB in domain B:}

\texttt{[cdsso\_peers]}
\[ websealA.domainA.com = <pathname>/A-B.key \]

This setting specifies what key websealB (in domain B) uses to decrypt a token received from websealA in domain A.

In the above example, the \texttt{A-B.key} file is generated on one machine (websealA, for example) and manually (and securely) copied to the other machine (websealB, for example).
4. Configuring the token time stamp

The token contains a configurable time stamp that defines the lifetime of the identity token. After the time stamp has expired, the token is considered invalid and is not used. The time stamp is used to help prevent replay attacks by setting a value short enough to prevent the token from being stolen and replayed within its lifetime.

The `authtoken-lifetime` parameter, located in the `[cdsso]` stanza of the WebSEAL configuration file, sets the token lifetime value. The value is expressed in seconds. The default value is 180:

```plaintext
[cdsso]
authtoken-lifetime = 180
```

You must take into account any clock skew among the participating domains. Clock skew means that the system times differ on the relevant servers in each domain. When this difference approaches the value of `authtoken-lifetime`, the effective lifetime of the token is greatly reduced. When this difference exceeds the value of `authtoken-lifetime`, tokens from one domain cannot be valid for the other domain. Administrators should adjust `authtoken-lifetime` accordingly. However, when clock skew requires that `authtoken-lifetime` be set to a large value, the risk of replay attacks increases. In this case, administrators should consider synchronizing the system time on the relevant servers in each domain.

See “Cross-domain single sign-on” on page 428.

5. Configuring the token label name

The authentication information used for a single sign-on transaction is placed in the redirected request as an encrypted token query string argument to the request. This token string requires a name, or label, to identify it. The label name uniquely identifies the request to the receiving WebSEAL server as a single sign-on request to be handled by the CDSSO token consume mechanism (library).

You must configure this token label identically on both WebSEAL servers participating in the single sign-on functionality. To configure the token label, use the `cdsso-argument` parameter located in the `[cdsso]` stanza of the WebSEAL configuration file. For example (default):

```plaintext
[cdsso]
cdsso-argument = PD-ID
```

See “Cross-domain single sign-on” on page 428.

Creating the CDSSO HTML link

The HTML link (located on the original server) that connects the user to a resource on the destination server must use a special CDSSO expression that directs the request to a CDSSO management page `pkmscdsso`:

`/pkmscdsso?<destination-URL>`

For example:


The token create library creates and encodes an authentication token (containing the user’s identity information) and includes this token in a redirected request to the resource using the destination URL information from the CDSSO link. For example:
Protecting the authentication token

While the authentication token does not contain authentication information (such as user name and password), it does contain a user identity that is trusted within the receiving domain. The token itself must therefore be protected against theft and replay.

The token is protected against theft off the wire through the use of SSL to secure communications between the WebSEAL servers and the users. The token could conceivably be stolen from the user’s browser history. The time stamp on the token should be short enough to make it unlikely that the token could be stolen and replayed during the lifetime of the token.

However, a token that has expired with respect to its time stamp is still vulnerable to cryptographic attacks. If the key used to encrypt the token is discovered or otherwise compromised, malicious users could build their own tokens.

These tokens could then be inserted into a "pseudo-CDSSO flow." They would be indistinguishable from real authentication tokens to the WebSEAL servers participating in the CDSSO domain. For this reason, the keys used to protect the tokens must also be carefully managed and changed on a regular basis.

UTF-8 encoding of tokens for cross domain single sign-on

The use of UTF-8 encoding for strings within tokens used for cross domain single sign-on is specified in the WebSEAL configuration file.

```
[cdsso]
use-utf8 = {true|false}
```

The default value is true.

When `use-utf8` is set to `false`, strings are encoded using the local code page. Use this value when implementing cross domain single sign-on with older (pre-Version 5.1) WebSEAL servers. WebSEAL servers from versions prior to 5.1 do not use UTF-8 encoding for tokens. When deploying an environment that includes these older servers, configure the Version 5.1 WebSEAL server to not use UTF-8 encoding. This setting is necessary for backwards compatibility.

For more information on WebSEAL support for UTF-8 encoding, see "Multi-locale support with UTF-8" on page 44.

Enabling compatibility with tokens prior to Version 4.1

In the Version 4.1 release of Tivoli Access Manager, the level of security for the encryption of the authentication token was increased. The Version 4.1 encryption algorithm is not backward compatible. If you are integrating CDSSO with servers using versions of Tivoli Access Manager prior to Version 4.1, you must enable the `pre-410-compatible-tokens` parameter in the [server] stanza of the WebSEAL configuration file. For example:

```
pre-410-compatible-tokens = yes
```

The default setting for this parameter is "no".
Enable backwards compatibility for Version 4.1 tokens

For Tivoli Access Manager Version 5.1, the format of the encryption of the authentication token was changed. This encryption algorithm is not backward compatible. If you are integrating authentication tokens with Version 4.1 WebSEAL servers, you must specify a configuration file setting in the WebSEAL configuration file to enable backwards compatibility.

Backwards compatibility with the older encryption format is not enabled by default:

[server]
pre-510-compatible-tokens = no

To enable backwards compatibility, set pre-510-compatible-tokens to yes:

[server]
pre-510-compatible-tokens = yes

Note: To enable backwards compatibility with WebSEAL servers prior to Version 4.1, you must set an additional parameter. See "Enabling compatibility with tokens prior to Version 4.1" on page 252.

Specify extended attributes to add to token

In the WebSEAL configuration file, you can specify extended attributes from a user credential to add to the cross-domain single sign-on token. Extended attributes consist of information about a user identity that is added to an extended attribute list when a user credential is created. Extended attributes can be added by a number of authentication mechanisms, including custom authentication modules. The custom authentication modules can be used, for example, to obtain user information from a registry that is external to Tivoli Access Manager.

You can use this setting to customize the contents of the cross-domain single sign-on token. This feature enables you to tailor the token contents to match the needs of the destination domain. When you use this feature to add an attribute to a token, you must also configure the WebSEAL configuration file for the server in the destination domain. For the destination server, the stanza [cdsso-incoming-attributes] is used to specify the handling (extract or ignore) of each attribute.

You can specify extended attributes by name, or you can declare a pattern that matches multiple attribute names. You can use standard Tivoli Access Manager wildcard-matching characters.

Table 33. Supported wildcard matching characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>The character that follows the backslash is part of a special sequence. Can be used to escape the other pattern matching characters: (? * [ ] ^ ). To match the backslash character, use &quot;&quot;.</td>
</tr>
<tr>
<td>?</td>
<td>Wildcard that matches a single character. For example, the string “abcde” is matched by the expression “ab?de”</td>
</tr>
<tr>
<td>*</td>
<td>Wildcard that matches zero or more characters.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Defines a set of characters, from which any can match. For example, the string “abcde” is matched with the regular expression “ab[cty]de”.</td>
</tr>
<tr>
<td>^</td>
<td>Indicates a negation. For example, the expression [^ab] matches anything but the ‘a’ or ‘b’ characters.</td>
</tr>
</tbody>
</table>
Each entry in the WebSEAL is assigned the name of the domain for which the token is intended. This name takes one or more arguments that specify names or patterns.

The syntax is:

```
[cddsso-token-attributes]
domain_name = pattern1 [pattern2], ... [patternN]
<default> = pattern1 [pattern2], ... [patternN]
```

The `<default>` entry is optional. When WebSEAL does not find an entry that matches the domain name, WebSEAL looks for a `<default>` entry. If the configuration file contains a `<default>` entry, WebSEAL uses the assigned attribute patterns for the current domain. The string `<default>` is a keyword, and must be specified exactly as shown above, including the "<" and ">" characters.

Example: You are creating a cross-domain single sign-on solution between two domains: example1.com and example2.com. Users log in to example1.com but can get redirected to example2.com during the user session. Your deployment includes a customized CDAS that inserts information into each user credential. The information includes a fixed name attribute "job_category" and a variable number of attributes, each prefixed with the characters "my_cdas_attr_". This information needs to be added to the cross-domain token. The configuration file entries would be:

```
example2.com = job_category, my_cdas_attr_*
```

**Specify extended attributes to extract from token**

In the WebSEAL configuration file, you can specify how the token consume library handles extended attributes that have been added to a cross-domain single sign-on token. The attributes can either be extracted or ignored. In some cases, you must extract the attributes because there is no way for a server in the destination domain to generate them. In other cases, you do not want to extract the tokens, because the server in the destination domain can use an independent process to gather the same extended attributes. For example, the attribute could reflect a timestamp that needs to reflect the system time on the destination server.

In the token consume library, attributes that are extracted from the token are passed through to the cross-domain mapping framework library. The default cross-domain mapping framework library passes attributes directly through to the user credential. Customized cross-domain mapping framework libraries can manipulate attributes as needed before passing them to the user credential.

The syntax for the entries is as follows:

```
[cddsso-incoming-attributes]
attribute_pattern = (preserve|refresh)
```

Typically, the names of the extended attributes (attribute_pattern) match the names of the attributes specified in the [cddsso-token-attributes] stanza of the configuration file for a WebSEAL server that generates the tokens. The value must be one of the following keywords:

- **preserve**
  - Extract all attributes that match attribute_pattern.
- **refresh**
  - Do not extract attributes that match attribute_pattern.
Extended attributes in the token that do not match an entry in 
[cdsso-incoming-attributes] are preserved (extracted).

The order of the entries in the stanza is important. The first entry that matches an 
attribute name is used. Other entries are ignored. For example, if you want to 
extract the attribute named my_special_attr1 but want to ignore all other entries 
with the prefix my_special_attr_, the stanza entries should be:

[cdsso-incoming-attributes]
my_special_attr1 = preserve
my_special_attr_* = refresh

Using the examples shown above in “Specify extended attributes to add to token” 
on page 253, the entries in the WebSEAL configuration file for a server that 
operates in the example2.com domain could be:

[cdsso-incoming-attributes]
job_category = preserve
my_cdas_attr_1 = preserve
my_cdas_attr_* = refresh

In this example, the attributes job_category and my_cdas_attr_1 are extracted from 
the token. The remainder of the attributes with the prefix my_cdas_attr_ are 
ignored.
e-community single sign-on

E-community single sign-on is another implementation of cross-domain authentication in an Tivoli Access Manager environment. The goal of cross-domain authentication is to allow users to access resources across multiple servers in multiple domains without re-authentication.

An "e-community" is a group of distinct domains (Tivoli Access Manager or DNS) that participate in a business relationship. These participating domains can be configured as part of one business (and perhaps using different DNS names for geographic reasons) or as disparate businesses with a shared relationship (for example, company headquarters, a life insurance company, and a financial management company).

In either scenario, there is always one domain that is designated the "home" or "owner" domain. In the case of participating businesses, the home domain owns the business agreements that govern the e-community.

In both scenarios, authentication information about the users who participate in the e-community (including the user names and passwords used for authentication) is maintained in the home domain. This arrangement allows a single point of reference for administration issues, such as help desk calls within the e-community that all refer to the home domain.

Alternatively, you can use the Tivoli Access Manager Web Portal Manager to delegate the management of this information such that participating domains have responsibility for the administration of their own users.

The diagram below illustrates a sample e-community with two participating domains: domain A (dA.com) and domain B (dB.com). In this example, domain A represents the home or owner domain. Domain B is a participating, or "remote", domain.
The home domain “owns” the users—that is, it controls the user’s authentication information. Regardless of where a user makes a request for resources, the home domain is always where the user must authenticate.

Authentication occurs against a master authentication server (MAS)—a server (or set of replica servers) that is located in the home domain and configured to authenticate all users. The diagram represents the MAS as mas.dA.com. The duty of the MAS should be restricted to providing authentication services. The MAS should not contain resources that are available to users.

After a user has successfully authenticated to the MAS, the MAS generates a “vouch for” token. This token is passed back to the server where the user is making the request. The server treats this “vouch for” token as proof that the user has successfully authenticated to the MAS and can participate in the e-community.

The transfer of information between e-community domains is described in detail in the section “e-community process flow” on page 258.

**e-community features and requirements**

- The model supports access via direct URLs (bookmarks) to resources. This feature contrasts with the CDSSO model that relies on a specially configured pkmscdsso link (see “Cross-domain single sign-on” on page 245).
- All users who are participating in the e-community authenticate against a single master authentication server (MAS) located in the home domain.
- The e-community implementation allows for “local” authentication in remote domains if the user does not have a valid account with the MAS (for example, users who belong to domain B but do not participate in the domain A-domain B e-community).
A user who fails authentication with the MAS when requesting a resource in a non-MAS (but participating) domain is given the option to authenticate to the local server where the request is being made.

- The MAS (and eventually other selected servers in the remote domains) "vouch for" the user's authenticated identity.
- Domain-specific cookies are used to identify the server that can provide "vouch for" services. This allows servers in a remote domain to request "vouch for" information locally. The encrypted contents of e-community cookies do not contain user identity or security information.
- Special tokens are used to pass encrypted "vouched for" user identity. The "vouch for" token does not contain actual user authentication information. Integrity is provided by shared secret key (triple-DES). The token contains a time-out (lifetime) value to limit the duration of the token validity.

**e-community process flow**

An e-community consists of a master authentication WebSEAL server (MAS) and additional WebSEAL servers located in the home domain and remote domains. The MAS can exist as a single instance of a WebSEAL server, or a set of WebSEAL replicas located behind a load balancer (where the load balancer is identified as the MAS).

All participating local and remote WebSEAL servers need to be configured to use the home domain MAS for initial client authentication. This is a hard requirement for servers in the home domain, and a soft requirement for servers in remote domains. For example, some servers in remote domains can be configured to handle their own authentication. These servers, and the resources they protect, can operate independently of the e-community, even if they are located in a participating e-community domain.

The e-community implementation is based on a "vouch for" system. Normally, when a user requests a resource from a WebSEAL server where the user has not established a valid session, WebSEAL prompts the user for authentication information. In an e-community configuration, the WebSEAL server identifies a "vouch for" server and requests verification from this "vouch for" server that the user has authenticated.

The "vouch for" server has valid credential information for that user. For the user's first request, the "vouch for" server is always the MAS. The MAS continues to serve as the "vouch for" server for resources located in the home domain. As the user continues with resource requests across the e-community, an individual server in each remote domain can build its own credential for the user (based on user identity information from the MAS) and assume the role of "vouch for" server for resources in its domain.

The verification requested of the "vouch for" server takes the form of a "vouch for" token. The "vouch for" server creates the token and returns it to the requesting WebSEAL server. The user identity information in the token is encrypted. The token contains a lifetime limit.

Upon receipt of the "vouch for" token, the requesting server builds credentials and a local session for that user. The user can now access the requested resource based on normal authorization controls. The user benefits from not having to re-authenticate—a goal of the e-community model.
Refer to the following diagram as you follow the e-community process flow in the remainder of this section. The process flow describes two possible FIRST access scenarios (1 and 2). These are followed by two possible NEXT access scenarios (3 and 4) which follow immediately after 2 or 3. Scenario 5 occurs any time after the initial access.

![Diagram](image)

Figure 10. e-community process flow

"Vouch For" Servers

- The MAS is always used to authenticate a user accessing any part of the e-community for the first time.
  
The MAS should perform only as an authentication server and not as a resource provider. The MAS should not be configured to act as a master authentication server and, simultaneously, protect resources. This recommendation addresses performance concerns and is not a security requirement.
- The MAS is always the "vouch for" server for the home domain (domain A in this example).
- A domain-specific e-community cookie is used to identify the "vouch for" server for all other servers within a given domain. The "vouch for" server is the first server in a domain that requests a "vouch for" token from the MAS. The "vouch for" server provides "vouch for" information for the user within the domain. Subsequent requests for "vouch for" services in a given remote domain can be made locally by this server, rather than accessing the out-of-domain MAS. In the home domain, the e-community cookie identifies the MAS as the "vouch for" server.

(1) FIRST e-Community Local (Domain A) Access: WebSEAL 1

1. User requests a resource protected by WebSEAL 1 (within the same domain as MAS). The browser contains no e-community cookie for this domain. WebSEAL 1 has no cached credentials for the user.
2. WebSEAL 1 configuration has e-community authentication enabled and specifies the location of the MAS. WebSEAL 1 redirects the browser to a special "vouch for" URL on the MAS.
3. The MAS receives the "vouch for" request and, failing to find credentials for that user, prompts the user to login.
4. After successful login, the MAS builds a credential for the user, stores it in the cache, and redirects the browser back to the originally requested URL on WebSEAL 1 with an encrypted "vouch for" token. In addition, a domain A-specific e-community cookie is placed on the browser to identify the "vouch for" server for this domain (in this case, the MAS).

If the login attempt is unsuccessful, the MAS returns a "vouch for" token that indicates a failure status. This token is constructed to be indistinguishable from a success status "vouch for" token. The requesting server reacts to a failure status token as if the user had failed local authentication.

5. WebSEAL 1 decrypts the token and builds its own credential for the user.

Note: Identity mapping should not be required within the same domain. If identity mapping is required, WebSEAL 1 must use the Cross-domain Mapping Framework (CDMF).

6. The authorization service permits or denies the request.

(2) FIRST e-Community Remote (Domain B) Access: WebSEAL 3

1. User requests a resource protected by WebSEAL 3 (remote domain B). The browser contains no e-community cookie for this domain. WebSEAL 3 has no cached credentials for the user.

2. WebSEAL 3 configuration has e-community authentication enabled and specifies the location of the MAS. WebSEAL 3 redirects the browser to a special "vouch for" URL on the MAS.

3. The MAS receives the "vouch for" request and, failing to find credentials for that user, prompts the user to login.

4. After successful login, the MAS builds a credential for the user, stores it in the cache, and redirects the browser back to the originally requested URL on WebSEAL 3 with an encrypted "vouch for" token. In addition, a domain A-specific e-community cookie is placed on the browser to identify the "vouch for" server for this domain (in this case, the MAS).

If the login attempt is unsuccessful, the MAS returns a "vouch for" token that indicates a failure status. This token is constructed to be indistinguishable from a success status "vouch for" token. The user fails authentication at the MAS, then the user is prompted for a local authentication at WebSEAL 3. If the user's account exists on this server, authentication then succeeds.

5. WebSEAL 3 decrypts the token and builds its own credential for the user.

6. WebSEAL 3 creates and sets a second e-community cookie (valid for domain B) on the browser, identifying WebSEAL 3 as the "vouch for" server for domain B.

7. The authorization service permits or denies the request.

(3) NEXT e-Community Local (Domain A) Access: WebSEAL 2

1. User requests a resource protected by WebSEAL 2 (within the same domain as MAS). The browser contains a domain A e-community cookie identifying the MAS as the "vouch for" server. WebSEAL 2 receives this cookie. WebSEAL 2 has no cached credentials for the user.

2. WebSEAL 2 configuration has e-community authentication enabled and specifies the location of the MAS. The presence of the domain A e-community cookie overrides the WebSEAL 2 configuration for the MAS location. The scenario provides WebSEAL 2 with the identity of the "vouch for" server. (If scenario 2 occurred first, there would also be a domain B cookie maintained on the browser that would not be sent to a domain A server.)
3. WebSEAL 2 redirects the browser to a special "vouch for" URL on the domain A "vouch for" server identified in the cookie (in this case the MAS, because WebSEAL 2 is in domain A).

4. The MAS receives the "vouch for" request and finds credentials for that user in the cache (this occurred in scenario 1 and 2).

5. The MAS redirects the browser back to the originally requested URL on WebSEAL 2 with an encrypted "vouch for" token.

6. WebSEAL 2 decrypts the token and builds its own credential for the user.

7. The authorization service permits or denies the request.

(4) NEXT e-Community Remote (Domain B) Access: WebSEAL 4

1. User requests a resource protected by WebSEAL 4 (remote domain B). If scenario 2 occurred first, the browser contains a domain B e-community cookie identifying WebSEAL 3 as the "vouch for" server. WebSEAL 4 has no cached credentials for the user.

2. WebSEAL 4 configuration has e-community authentication enabled and specifies the location of the MAS. The presence of a domain B e-community cookie overrides the WebSEAL 4 configuration for the MAS location. The cookie provides WebSEAL 4 with the identity of the "vouch for" server. (If scenario 1 occurred first, there would only be a domain A cookie maintained on the browser that would not be sent to a domain B server. The configured MAS location would be used instead. WebSEAL 4 would then become the "vouch for" server for domain B.)

3. If scenario 2 occurred first, WebSEAL 4 redirects the browser to a special "vouch for" URL on the domain B "vouch for" server identified in the domain B cookie (in this case WebSEAL 3).

4. WebSEAL 3 receives the "vouch for" request and finds credentials for that user in the cache (this occurred in scenario 2).

5. WebSEAL 3 redirects the browser back to the originally requested URL on WebSEAL 4 with an encrypted "vouch for" token.

6. WebSEAL 4 decrypts the token and builds its own credential for the user.

7. The authorization service permits or denies the request.

(5) ANOTHER e-Community Local (Domain A) Access: WebSEAL 2

1. User connects to WebSEAL 2 (domain A) with a request. If scenario 3 occurred, WebSEAL 2 has cached credentials for the user.

2. The authorization service permits or denies the request.

Logout from the e-Community

- If the user logs out by closing the browser, all SSL sessions and all e-community cookies are cleared.
- If the user logs out via the /pkmslogout page, the SSL session and e-community cookie for that domain are cleared.

Understanding the e-community cookie

- The e-community cookie is a domain-specific cookie set by one WebSEAL server, stored in the memory of the user’s browser, and transmitted to other WebSEAL servers (in the same domain) in subsequent requests.
• The domain-specific cookie contains the name of the "vouch for" server, the e-community identity, a location (URL) of the "vouch for" server and functionality, and a lifetime value. The cookie contains no user or security information.

• The e-community cookie allows servers in participating domains to request "vouch for" information locally. The e-community cookie for the domain where the MAS resides plays a less significant role.

• The cookie has a lifetime (timeout) value that is set in the WebSEAL configuration file. This lifetime value specifies how long a remote server is able to provide "vouch for" information for the user. When the cookie lifetime has expired, the user must be redirected to the MAS for authentication.

• The cookie is cleared from memory when the browser is closed. If the user logs out of a specific domain, the e-community cookie is overwritten as empty. This action effectively removes it from the browser.

Understanding the "vouch for" request and reply

The e-community "vouch for" operation requires dedicated functionality accessed through two specially constructed URLs: the "vouch for" request and the "vouch for" reply.

The "vouch for" Request

The "vouch for" request is triggered when a user requests a resource from a target server (configured for e-community) that contains no credential information for that user. The server sends a redirect to the "vouch for" server (either the MAS or a delegated "vouch for" server identified in an e-community cookie).

The "vouch for" request contains the following information:
https://<vouch-for-server>/pkmsvouchfor?<ecommunity-name>&<target-URL>

The receiving server checks the ecommunity-name to validate the e-community identity. The receiving server uses the target-URL in the "vouch for" reply to redirect the browser back to the originally requested page.

The pkmsvouchfor "vouch for" URL is configurable.

For example:

The "vouch for" Reply

The "vouch for" reply is the response from the "vouch for" server to the target server. The "vouch for" server is either the MAS or a delegated "vouch for" server in a domain remote from the MAS domain.

The "vouch for" reply contains the following information:

The PD-VFHOST label identifies the server that performed the "vouch for" operation. The receiving (target) server uses this information to select the correct key required to decrypt the "vouch for" token. The PD-VF label identifies the encrypted token in the "vouch for" reply URL.

For example:
https://w5.db.com/index.html?PD-VFHOST=mas.dA.com&PD-VF=3qhe9fjkp...ge56wgb
Understanding the “vouch for” token

In order to achieve cross-domain single sign-on, some user identity information must be transmitted between servers. This sensitive information is handled using a redirect that includes the identity information encrypted as part of the URL. This encrypted data is called a “vouch for” token.

- The token contains the "vouch for" success or failure status, the user’s identity (if successful), the fully qualified name of the server that created the token, the e-community identity, and a creation time value.
- The holder of a valid "vouch for" token can use this token to establish a session (and set of credentials) with a server without explicitly authenticating to that server.
- The token is encrypted using a shared triple-DES secret key so that its authenticity can be verified.
- Encrypted token information is not stored on the browser.
- The token is passed only once. The receiving server uses this information to build user credentials in its own cache. The server uses these credentials for future requests by that user during the same session.
- The token has a lifetime (timeout) value that is set in the WebSEAL configuration file. This value can be very short (seconds) to reduce the risk of a re-play attack.

Configuring e-community single sign-on

e-community conditions and requirements

- The e-community implementation requires a consistent configuration across all WebSEAL servers in all domains participating in the e-community.
- For e-community to function successfully, each participating WebSEAL server must reveal its fully qualified host name to the other participating servers in the cross-domain environment. If any host name does not include a domain, e-community cannot be enabled and an error message is logged in msg_webseal1d.log. When setting up an e-community environment, ensure that the machine-specific networking setup for each participating server is configured to identify the server with a fully qualified host name.
- All WebSEAL servers participating in e-community must have machine times synchronized. Authentication between servers can fail when machine time differences are too great.
- The e-community implementation is supported on both HTTP and HTTPS.
- The following e-community scenario is not supported for production or testing purposes: a MAS WebSEAL instance and a participating WebSEAL server instance configured to use the same network interface on the same machine.
- Individual e-community domains manage their own user identities and associated privileges. You can use the Cross-domain Mapping Function (CDMF) API to map a user from a remote domain to a valid user in the local domain.
  
  If the e-community domains share global user identities, those users could be distinguished by different passwords in the different domains. For example, a user "abc" can exist in both domain A and domain B, using different passwords for each domain.
- Configuration for e-community is set in the WebSEAL configuration file of each participating WebSEAL server.
- If the originally requested URL is not redirected back to the browser from the MAS (or "vouch for" server), there could be a problem with page caching if the
browser is Microsoft Internet Explorer. If this is the case, configure the browser to always check for newer versions of stored pages:

Tools > Internet Options > General > Temporary Internet Files > Settings

- Do not configure the MAS server on the same interface (IP address) of another participating WebSEAL server instance.
- Because some WebSEAL configuration requires machine host names to be described as fully qualified host names, you must ensure that your system and network can resolve machine names into fully qualified host names. For example, using fully qualified host names allows for many host names (IP addresses) per machine, as in the case of multiple WebSEAL instances.

Resolving machine names

E-community can be disabled upon WebSEAL startup because the machine itself is not adequately configured to resolve machine names. The machine on which WebSEAL resides needs to be able to fully resolve an IP address. Because this functionality is very operating system-specific, it is not the role of this document to provide instructions. Always consult your system administrator if you are not sure your system has the proper capabilities.

The following general Solaris-specific information is provided only as an example:

Goal: Configure the machine to first look to DNS before checking the local /etc/hosts file for DNS information.
1. Make sure that /etc/resolv.conf has valid DNS server entries.
2. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:

```
hosts dns files
```

Alternative goal: Configure the machine to first use local DNS information (/etc/hosts) before checking DNS.
1. Configure the machine to check /etc/hosts before looking to DNS. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:

```
hosts files dns
```
2. Enter appropriate DNS information in /etc/hosts:

```
webseal1.fully.qualified.com 1.11.111.111
webseal2.fully.qualified.com 2.22.222.222
```

The following general Windows-specific information is provided only as an example:

1. Use DNS and specify 2 IP addresses:

```
Network Connections > LAN > Properties > TCP/IP
```

2. Specify a valid DNS server under the Advanced settings:

```
Network Connections > LAN > Properties > TCP/IP > Advanced > DNS > Add...
```

3. In this same window, specify the primary DNS suffix for this connection:

```
Network Connections > LAN > Properties > TCP/IP > Advanced > DNS > Add...
```

4. In your system properties, specify the computer name and its DNS suffix:

```
My Computer > Properties > Network ID > Properties > Computer name
My Computer > Properties > Network ID > Properties > More > Primary DNS suffix
```

E-community configuration summary

An e-community is configured under the following conditions and guidelines:
The "vouch for" server (the MAS or a delegated "vouch for" server) always has the token create responsibility.

- The receiving server (where the requested resource is located) always has the token consume responsibility.
- A delegated "vouch for" server (for all domains remote from the MAS domain) must have both token create and token consume capabilities.

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division:

**Configuring default token create functionality on the "vouch for" server**

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division.

1. Enable e-community authentication to process single sign-on requests by communication type ([e-community-sso-auth]).
2. Specify the unifying name of the e-community for all participating servers ([e-community-name]).
3. Configure the built-in single sign-on authentication mechanism (library) for token create ([sso-create]).
4. Create the key file used to encode and decode the "vouch for" token. Copy the key file to all appropriate participating servers ([e-community-domain-keys] stanza).
5. Configure the token label parameter used in the "vouch for" reply ([vf-argument]).
6. Specify if this server is the MAS or not the MAS ([is-master-authn-server]).
7. Specify the "vouch for" URL used in the "vouch for" request ([vf-url]).
8. Configure token and ec-cookie lifetime values ([vf-token-lifetime] and [ec-cookie-lifetime]).

**Configuring default token consume functionality on the receiving server**

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division.

1. Enable e-community authentication to process single sign-on requests by communication type ([e-community-sso-auth]).
2. Specify the unifying name of the e-community for all participating servers ([e-community-name]).
3. Configure the built-in single sign-on authentication mechanism (library) for token consume ([sso-consume]).
5. Configure the token label parameter used in the "vouch for" reply ([vf-argument]).
6. Specify that this server is not the MAS ([is-master-authn-server]).
7. Specify the "vouch for" URL used in the "vouch for" request ([vf-url]).
8. Configure token and ec-cookie lifetime values ([vf-token-lifetime] and [ec-cookie-lifetime]).
1. Enabling and disabling e-community authentication

The e-community-sso-auth parameter, located in the [e-community-sso] stanza of the WebSEAL configuration file, enables and disables the e-community authentication method, and processes single sign-on requests by communication type.

- To enable the e-community authentication method, enter "http", "https", or "both".
  The values "http", "https", and "both" specify the type of communication used by e-community participants.
- To disable the e-community authentication method, enter "none".
  The value "none" disables e-community for that server. The default setting is "none".

For example:

```
[e-community-sso]
e-community-sso-auth = https
```

**Note:** You must stop and restart the WebSEAL server in order to activate changes to the WebSEAL configuration file. Complete all of the applicable configuration steps in this section and then restart WebSEAL.

2. Specifying an e-community name

The e-community-name parameter identifies the unifying name of the e-community for all participating servers in all participating domains. For example:

```
[e-community-sso]
e-community-name = companyABC
```

The e-community-name value must be the same for all WebSEAL servers in all domains that are participating in the e-community.

3. Configuring the single sign-on authentication mechanism

The default e-community configuration requires that you enable the sso-create and sso-consume single sign-on authentication mechanisms. The sso-create mechanism is required by the initial WebSEAL server for creating the "vouch for" token and building the redirected request. The sso-consume mechanism is required by the receiving WebSEAL server to decode the "vouch for" token and build the user credentials from the identity information contained in the token. For the default e-community configuration, each parameter specifies a built-in "vouch for" token library file. One library contains the code for the token create functionality and the other library contains the code for the token consume functionality.

- On UNIX, the library files are called libssocreate. {so | a | sl} and libssoconsume. {so | a | sl}.
- On Windows, the library files are DLL files called ssocreate.dll and ssoconsume.dll.

<table>
<thead>
<tr>
<th>Authentication Mechanism</th>
<th>Single Sign-on Token Library</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solaris</td>
</tr>
<tr>
<td>sso-create</td>
<td>libssocreate.so</td>
</tr>
<tr>
<td>sso-consume</td>
<td>libssoconsume.so</td>
</tr>
</tbody>
</table>
You can configure the e-community authentication mechanism by specifying the **sso-create** and **sso-consume** parameters with the full path name of the appropriate platform-specific default library file in the [authentication-mechanism] stanza of the WebSEAL configuration file.

For example:

**Solaris:**

```properties
[authentication-mechanisms]
sso-create = /opt/pdwebrte/lib/libssocreate.so
sso-consume = /opt/pdwebrte/lib/libssoconsume.so
```

**Windows:**

```properties
[authentication-mechanisms]
sso-create = C:\Program Files\Tivoli\PDWebRTE\bin\ssocreate.dll
sso-consume = C:\Program Files\Tivoli\PDWebRTE\bin\ssoconsume.dll
```

### 4. Encrypting the "vouch for" token

WebSEAL must encrypt the authentication data placed in the token using a key generated by the **cdsso_key_gen** utility. You must "synchronize" this key by sharing the key file with each WebSEAL server in each participating domain. Each participating WebSEAL server in each domain needs to use the same key.

**Note:** The distribution of key files is not a part of the Tivoli Access Manager e-community process. You must manually and securely copy keys to each participating server.

The **cdsso_key_gen** utility requires that you specify the location (absolute pathname) of the key file when you run the utility. You must also use a full path name to run this utility:

**UNIX:**

```bash
# /opt/pdwebrte/bin/cdsso_key_gen <keyfile-location>
```

**Windows:**

```cmd
MSDOS> C:\Program Files\Tivoli\PDWebRTE\bin\cdsso_key_gen <keyfile-location>
```

The location of the key files used to secure tokens sent between servers participating in the e-community is specified in the [e-community-domain-keys] stanza.

```properties
[e-community-domain-keys]
<domain-name> = <full-keyfile-pathname>
<domain-name> = <full-keyfile-pathname>
```

**E-community Domain Keys**

The location of the key files required for encrypting and decrypting the tokens exchanged among the servers participating in the e-community is specified in the [e-community-domain-keys] stanza. You must specify fully qualified domain names for the servers and absolute path names for the key file locations.

The following example provides the MAS (domain A) with key files for communicating with two remote domains (dB and dC) and a key for communicating with other servers in domain A:
[e-community-domain-keys]
da.com = /abc/xyz/key.fileA-A
dB.com = /abc/xyz/key.fileA-B
dC.com = /abc/xyz/key.fileA-C

In this example, key.fileA-A identifies the key file used between all of the servers in domainA.

key.fileA-B identifies the key file used between domain A and domain B.

key.fileA-C identifies the key file used between domain A and domain C.

Each remote server needs to have a copy of the appropriate key file used by the MAS. To exchange tokens with the MAS (domain A), all servers in domain B require copies of key.fileA-B:

[e-community-domain-keys]
da.com = /efg/hij/key.fileA-B

To exchange tokens with the MAS (domain A), all servers in domain C require copies of key.fileA-C:

[e-community-domain-keys]
da.com = /efg/hij/key.fileA-C

Any servers in domain A which use authentication services provided by the MAS must have a copy of key.fileA-A:

[e-community-domain-keys]
da.com = /efg/hij/key.fileA-A

In this example, key.fileB-B identifies the key file used between all of the servers in domainB. Also, key.fileC-C identifies the key file used between all of the servers in domainC

[e-community-domain-keys]
dB.com = /efg/hij/key.fileB-B
dC.com = /efg/hij/key.fileC-C

5. Configuring the "vouch for" token label name

The authentication information used for a single sign-on transaction is placed in the redirected request as an encrypted token query string argument to the request. This token string requires a name, or label, to identify it. The label name uniquely identifies the request to the receiving WebSEAL server as a single sign-on request to be handled by the single sign-on token consume mechanism (library).

You must configure this token label on both WebSEAL servers participating in the single sign-on functionality. To configure the token label, use the vf-argument parameter located in the [e-community-sso] stanza of the WebSEAL configuration file. For example (default):

[e-community-sso]
vf-argument = PD-VF

See "e-community single sign-on" on page 431.

6. Specifying the master authentication server (MAS)

You must specify which server machine in the e-community is to function as the master authentication server (MAS). You must also specify if a server machine is not the MAS.
is-master-authn-server

Use the **is-master-authn-server** parameter to specify whether a server is the MAS or not. Values include "yes" or "no".

For example:

```
[e-community-sso]
is-master-authn-server = yes
```

Multiple WebSEALs can be configured to act as master authentication servers and then placed behind a load balancer. In this scenario, the load balancer is “recognized” as the MAS by all other WebSEAL servers in the e-community.

If the server you are configuring is not the MAS, use the **master-authn-server** to specify to this server the location of the MAS.

**master-authn-server**

If the **is-master-authn-server** parameter is set to ”no”, this parameter must be uncommented and specified. The parameter identifies the fully qualified domain name of the MAS.

For example:

```
[e-community-sso]
master-authn-server = mas.dA.com
```

Additionally, you can specify the HTTP and HTTPS listening ports used by the MAS if these port values are other than the default (port 80 for HTTP and port 4443 for HTTPS).

**master-http-port**

If **e-community-sso-auth** enables HTTP e-community authentication and the master authentication server listens for HTTP requests on a port other than the standard HTTP port (port 80), the **master-http-port** parameter identifies the non-standard port. This parameter is ignored if this server is the master authentication server. By default, this parameter is disabled.

```
[e-community-sso]
master-http-port = <port-number>
```

**master-https-port**

If **e-community-sso-auth** enables HTTPS e-community authentication and the master authentication server listens for HTTPS requests on a port other than the standard HTTPS port (port 443), the **master-http-port** parameter identifies the non-standard port. This parameter is ignored if this server is the master authentication server. By default, this parameter is disabled.

```
[e-community-sso]
master-https-port = <port-number>
```

### 7. Specifying the "vouch for" URL

**vf-url**

The **vf-url** parameter specifies the “vouch for” URL. The value must begin with a forward-slash (/). The default value is `/pkmsvouchfor`. 
For example:

```
[e-community-sso]
vf-url = /pkmsvouchfor
```

You can also express an extended URL:
```
vf-url = /ecommA/pkmsvouchfor
```

The extended URL is used when the client is communicating with a MAS that is not a WebSEAL server. This use of vf-url enables the client to specify access to a MAS with specialized authentication library, such as a customized token consumption library.

8. Configure token and ec-cookie lifetime values

**vf-token-lifetime**

The **vf-token-lifetime** parameter sets the lifetime timeout value (in seconds) of the "vouch for" token. This value is checked against the creation time stamped on the cookie. The default value is 180 seconds. You must take into account clock skew among participating servers.

For example:
```
[e-community-sso]
vf-token-lifetime = 180
```

**ec-cookie-lifetime**

The **ec-cookie-lifetime** parameter specifies the maximum lifetime (in minutes) of the e-community domain cookie. The default value is 300 minutes.

For example:
```
[e-community-sso]
ec-cookie-lifetime = 300
```

You must take into account any clock skew among the participating domains. Clock skew means that the system times differ on the relevant servers in each domain. When this difference approaches the value of vf-token-lifetime, the effective lifetime of the token is greatly reduced. When this difference exceeds the value of vf-token-lifetime, tokens from one domain cannot be valid for the other domain. Administrators should adjust vf-token-lifetime accordingly. However, when clock skew requires that vf-token-lifetime be set to a large value, the risk of replay attacks increases. In this case, administrators should consider synchronizing the system time on the relevant servers in each domain.

See “e-community single sign-on” on page 431.

**Enabling unauthenticated access**

You can control whether unauthenticated users are allowed access to unprotected resources on e-community SSO slave servers. When authenticated users are allowed this access, the slave server can serve the resource without requiring that the user authenticate through the master authentication server. When this policy is configured, the slave server will redirect to the master authentication server only when the client requests access to a protected resource.

This policy is set through a setting in the WebSEAL configuration file:
ecsso-allow-unauth = {yes|no}

When ecsso-allow-unauth is set to yes, unauthenticated access is enabled. The default setting is yes.

When ecsso-allow-unauth is set to no, unauthenticated access is disabled. In this case, clients must authenticate through the master authentication server when requesting access to a resource (protected or not protected) on an e-community SSO slave server.

Note: The default behavior changed for WebSEAL Version 5.1. In prior versions, unauthenticated access was disabled. To retain backwards compatible behavior with older versions of WebSEAL, set ecsso-allow-unauth = no.

UTF-8 encoding of tokens for e-community single sign-on

The use of UTF-8 encoding for strings within tokens used for e-community single sign-on is specified in the WebSEAL configuration file.

use-utf8 = {yes|no}

The default value is yes.

When use-utf8 is set to no, strings are encoded using the local code page. Use this value when implementing e-community single sign-on with older (pre-Version 5.1) WebSEAL servers. WebSEAL servers from versions prior to 5.1 do not use UTF-8 encoding for tokens. When deploying an environment that includes these older servers, configure the Version 5.1 WebSEAL server to not use UTF-8 encoding. This setting is necessary for backwards compatibility.

For more information on WebSEAL support for UTF-8 encoding, see "Multi-locale support with UTF-8" on page 44.

Enabling compatibility with tokens prior to Version 4.1

In the Version 4.1 release of Tivoli Access Manager, the level of security for the encryption of the authentication token was increased. The Version 4.1 encryption algorithm is not backward compatible. If you are integrating ECSSO with servers using versions of Tivoli Access Manager prior to Version 4.1, you must enable the pre-410-compatible-tokens parameter in the [server] stanza of the WebSEAL configuration file. For example:

pre-410-compatible-tokens = yes

The default setting for this parameter is "no".

Enable backwards compatibility for Version 4.1 tokens

For Tivoli Access Manager Version 5.1, the format of the encryption of the authentication token was changed. This encryption algorithm is not backward compatible. If you are integrating authentication tokens with Version 4.1 WebSEAL servers, you must specify a configuration file setting in the WebSEAL configuration file to enable backwards compatibility.

Backwards compatibility with the older encryption format is not enabled by default:
To enable backwards compatibility, set `pre-510-compatible-tokens` to yes:

```
[server]
pre-510-compatible-tokens = yes
```

**Note:** To enable backwards compatibility with WebSEAL servers prior to Version 4.1, you must set an additional parameter. See “Enabling compatibility with tokens prior to Version 4.1” on page 271.

### Specify extended attributes to add to token

In the WebSEAL configuration file, you can specify extended attributes from a user credential to add to the cross-domain single sign-on token. Extended attributes consist of information about a user identity that is added to an extended attribute list when a user credential is created. Extended attributes can be added by a number of authentication mechanisms, including external authentication services (CDASs). The CDASs can be used, for example, to obtain user information from a registry that is external to Tivoli Access Manager.

You can use this setting to customize the contents of the e-community single sign-on token. This feature enables you to tailor the token contents to match the needs of the destination domain. When you use this feature to add an attribute to a token, you must also configure the WebSEAL configuration file for the server in the destination domain. For the destination server, the stanza `[ecsso-incoming-attributes]` is used to specify the handling (extract or ignore) of each attribute.

You can specify extended attributes by name, or you can declare a pattern that matches multiple attribute names. You can use standard Tivoli Access Manager wildcard-matching characters.

**Table 34. Supported wildcard matching characters**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>The character that follows the backslash is part of a special sequence. Can be used to escape the other pattern matching characters: ( ? * [ ] ^ ). To match the backslash character, use &quot;\&quot;.</td>
</tr>
<tr>
<td>?</td>
<td>Wildcard that matches a single character. For example, the string “ab\cd” is matched by the expression “ab?de”</td>
</tr>
<tr>
<td>*</td>
<td>Wildcard that matches zero or more characters.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Defines a set of characters, from which any can match. For example, the string “ab\cd” is matched with the regular expression “ab[cty]de”</td>
</tr>
<tr>
<td>^</td>
<td>Indicates a negation. For example, the expression [^ab] matches anything but the ‘a’ or ‘b’ characters.</td>
</tr>
</tbody>
</table>

Each entry in the WebSEAL is assigned the name of the domain for which the token is intended. This name takes one or more arguments that specify names or patterns.

The syntax is:

```
[ecsso-token-attributes]
domain_name = pattern1 [pattern2], ... [patternN]
<default> = pattern1 [pattern2], ... [patternN]
```

272  IBM Tivoli Access Manager for e-business: WebSEAL Administration Guide
The <default> entry is optional. When WebSEAL does not find an entry that matches the domain name, WebSEAL looks for a <default> entry. If the configuration file contains a <default> entry, WebSEAL uses the assigned attribute patterns for the current domain. The string <default> is a keyword, and must be specified exactly as shown above, including the < and > characters.

Example: You are creating an e-community single sign-on solution between two domains: example1.com and example2.com. Users log in to example1.com but can get redirected to example2.com during the user session. Your deployment includes a customized CDAS that inserts information into each user credential. The information includes a fixed name attribute "job_category" and a variable number of attributes, each prefixed with the characters "my_cdas_attr_". This information needs to be added to the cross-domain token. The configuration file entries would be:

example2.com = job_category, my_cdas_attr_*

Specify extended attributes to extract from token

In the WebSEAL configuration file, you can specify how the token consume library handles extended attributes that have been added to an e-community single sign-on token. The attributes can either be extracted or ignored. In some cases, you must extract the attributes because there is no way for a server in the destination domain to generate them. In other cases, you do not want to extract the tokens, because the server in the destination domain can use an independent process to gather the same extended attributes. For example, the attribute could reflect a timestamp that needs to reflect the system time on the destination server.

In the token consume library, attributes that are extracted from the token are passed through to the cross-domain mapping framework library. The default cross-domain mapping framework library passes attributes directly through to the user credential. Customized cross-domain mapping framework libraries can manipulate attributes as needed before passing them to the user credential.

The syntax for the entries is as follows:

[ecsso-incoming-attributes]
attribute_pattern = {preserve|refresh}

Typically, the names of the extended attributes (attribute_pattern) match the names of the attributes specified in the [ecsso-token-attributes] stanza of the configuration file for a WebSEAL server that generates the tokens. The value must be one of the following keywords:

• preserve
  Extract all attributes that match attribute_pattern.

• refresh
  Do not extract attributes that match attribute_pattern.

Extended attributes in the token that do not match an entry in [ecsso-incoming-attributes] are preserved (extracted).

The order of the entries in the stanza is important. The first entry that matches an attribute name is used. Other entries are ignored. For example, if you want to extract the attribute named my_special_attr1 but want to ignore all other entries with the prefix my_special_attr_, the stanza entries should be:
Using the examples shown above in “Specify extended attributes to add to token” on page 272, the entries in the WebSEAL configuration file for a server that operates in the example2.com domain could be:

```
[ecsso-incoming-attributes]
job_category = preserve
my_cdas_attr_1 = preserve
my_cdas_attr_* = refresh
```

In this example, the attributes job_category and my_cdas_attr_1 are extracted from the token. The remainder of the attributes with the prefix my_cdas_attr_ are ignored.
Chapter 10. WebSEAL junctions

A WebSEAL junction is an HTTP or HTTPS connection between a front-end WebSEAL server and a back-end Web application server. Junctions logically combine the Web space of the back-end server with the Web space of the WebSEAL server, resulting in a unified view of the entire Web object space. A junction allows WebSEAL to provide protective services on behalf of the back-end server. WebSEAL performs authentication and authorization checks on all requests for resources before passing those requests across a junction to the back-end server. Junctions also allow a variety of single sign-on solutions between a client and the junctioned back-end application.

You can create WebSEAL junctions with the `pdadmin` command-line utility or with the Web Portal Manager. This chapter discusses the details of the many options for configuring WebSEAL junctions.

Topic Index:
- “WebSEAL junctions overview” on page 276
- “Managing junctions with Web Portal Manager” on page 278
- “Using pdadmin to create junctions” on page 279
- “Configuring a basic WebSEAL junction” on page 280
- “Mutually authenticated SSL junctions” on page 282
- “Creating TCP and SSL proxy junctions” on page 285
- “WebSEAL-to-WebSEAL junctions over SSL” on page 286
- “Modifying URLs to back-end resources” on page 287
- “Additional junction options” on page 297
- “Technical notes for using WebSEAL junctions” on page 307
- “Using query_contents with third-party servers” on page 309
WebSEAL junctions overview

You can create the following WebSEAL junction types:

- WebSEAL to back-end server over TCP connection
- WebSEAL to back-end server over SSL connection
- WebSEAL to back-end server over TCP connection via HTTP proxy server
- WebSEAL to back-end server over SSL connection via HTTPS proxy server
- WebSEAL to WebSEAL over SSL connection

You must address the following two concerns when creating any junction:

1. Decide where to junction (mount) the Web application server(s) in the WebSEAL object space.
2. Choose the type of junction.

Junction database location and format

WebSEAL junction information is now stored in XML-formatted database files. The location of the junction database directory is defined in the [junction] stanza of the WebSEAL configuration file. The directory is relative to the WebSEAL server root (server-root parameter in the [server] stanza):

[junction]
junction-db = jct

- Each junction is defined in a separate file with a .xml extension.
- Use pdadmin utility to create and manage junctions and options.
- The XML format allows you to manually create, edit, duplicate, and backup junction files.

Applying coarse-grained access control: summary

1. Use the pdadmin utility or the Web Portal Manager to create a junction between WebSEAL and the back-end server.
2. Place an appropriate ACL policy on the junction point to provide coarse-grained control to the back-end server.

Applying fine-grained access control: summary

1. Use the pdadmin utility or the Web Portal Manager to create a junction between WebSEAL and the back-end server.
   
   WebSEAL cannot automatically "see" and understand a third-party file system. You must inform WebSEAL of the third-party object space using a special application, called query_contents, that inventories the third-party Web space and reports the structure and contents to WebSEAL.

2. Copy the query_contents program to the third-party server.
3. Apply ACL policy to appropriate objects in the unified object space.

Guidelines for creating WebSEAL junctions

The following guidelines summarize the "rules" for junctions:

- You can add a junction anywhere in the primary WebSEAL object space
- You can junction multiple replica back-end servers at the same mount point
  Multiple replica back-end servers mounted to the same junction point must be of the same type—TCP or SSL
- ACL policies are inherited across junctions to third-party servers
• The junction point name should be unique and not match any directory in the Web space of the local WebSEAL server. For example, if WebSEAL has resources of the form /path/..., do not create a junction point with the name /path.

• The junction point should not match any directory in the Web space of the back-end server if HTML pages from that server contain programs (such as Javascript or applets) with server-relative URLs to that directory. For example, if pages from the back-end server contain programs with a URL of form /path/..., do not create a junction point of name /path.

• Do not create multiple WebSEAL junctions that point to the same back-end application server/port. This type of configuration can cause unpredictable control of access to resources and therefore is not a recommended or supported Tivoli Access Manager configuration strategy.

Each WebSEAL junction can be secured by a unique set of access controls (ACLs). However, the ACL policy of each newly created junction overlays the policies of previously created junctions attached to the same back-end server/port. Subsequent junctions secured with more permissive ACLs can compromise previous junctions secured with less permissive ACLs. WebSEAL and the Tivoli Access Manager authorization model cannot guarantee secure access control with this type of junction implementation.

• WebSEAL supports HTTP 1.1 across junctions.

Note: You can also use the Tivoli Access Manager Web Portal Manager graphical user interface to create junctions. For more information, see the Web Portal Manager online help screens.

Additional references for WebSEAL junctions

See "Understanding WebSEAL junctions" on page 11 for a conceptual overview of WebSEAL junctions.

See Appendix B, “WebSEAL junction reference,” on page 491 for complete information on junction command options.
Managing junctions with Web Portal Manager

You can use the Tivoli Access Manager Web Portal Manager graphical user interface to create, list, and delete junctions.

Create a junction using Web Portal Manager

To create a junction using Web Portal Manager:
1. Log in to the domain.
2. Click WebSEAL → Create Junction.
3. Select the WebSEAL Server Name instance.
4. Type the name of the Junction Point.
5. Select a junction Type.
   For online help on supported types, click the ? icon in the upper-right corner.
6. Provide the configuration information, as required by the junction type you chose. Note that the fields in the Web Portal Manager window change based on the junction type. Select the appropriate check boxes, and type the requested values, in the following sections:
   - Server Information
   - Client Identity Headers
   - General Options
   - Basic Authentication
   For online help for each configuration section, click the ? icon in the upper-right corner.
7. When configuring single sign-on using LTPA to WebSphere, supply values for the WebSphere Single Sign-on section.
8. When configuring multiple junctions, you can control the allocation of worker threads by specifying values for the Junction Fairness section.

Note: The default value for Soft Limit is 90%. The online help incorrectly states that the default is 100%. For more information on junction fairness, see “Managing worker thread allocation” on page 41.

List junctions using Web Portal Manager

To list the configured junctions using Web Portal Manager:
1. Log in to the domain.
2. Click WebSEAL → List Junctions
3. Select the WebSEAL Server Name instance.
4. Click Show Junctions.

Delete junctions using Web Portal Manager

To delete one or more configured junctions using Web Portal Manager:
1. Log in to the domain.
2. Click WebSEAL → List Junctions
3. Select the WebSEAL Server Name instance.
4. Click Show Junctions.
5. Select the check box next to the junction name and click Delete.
   You can delete multiple junctions at the same time.
Using pdadmin to create junctions

Before using pdadmin, you must login to a secure domain as the sec_master administration user.

Note: You can also use the Tivoli Access Manager Web Portal Manager graphical user interface to create junctions. For more information, see the Web Portal Manager online help screens.

For example:

UNIX:

# pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
padmin>

Windows:

MSDOS> pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
padmin>

To create WebSEAL junctions, you use the pdadmin server task create command:

```
padmin> server task server_name-host_name create options
```

For example, if the configured name of a single WebSEAL server is default, the server_name is default-webseald followed by -host_name. For example, the server name would be:

default-webseald-cruz.dallas.ibm.com

If you install multiple WebSEAL server instances on the same machine, the Access-Manager-server is the configured name of the WebSEAL server instance, followed by webseald, followed by the host name:

```
instance_name-webseald-host-name
```

For example, if the configured names for two additional WebSEAL instances are webseal2 and webseal3, the server identifications appear as:

webseal2-webseald-cruz
webseal3-webseald-cruz

Use the server list command to verify server identification:

```
padmin> server list
default-webseald-cruz
webseal2-webseald-cruz
webseal3-webseald-cruz
```

For more information, see the reference page for pdadmin server task create (WebSEAL) in the IBM Tivoli Access Manager for e-business Command Reference.
Configuring a basic WebSEAL junction

WebSEAL supports both standard TCP (HTTP) and secure SSL (HTTPS) junctions between WebSEAL and back-end Web application servers.

The junction between WebSEAL and the back-end server is independent of the type of connection (and its level of security) between the client and the WebSEAL server.

The mandatory command options required to create a basic WebSEAL junction using `pdadmin` include:
- Host name of the back-end application server (–h option)
- Junction type: tcp, ssl, tcp proxy, ssl proxy, local (–t option)
- Junction point (mount point)

```
pdadmin> server task instance_name-webseald-host-name \    create -t type -h host_name jct_point
```

For example:

```
pdadmin> server task web1-webseald-cruz create -t tcp -h doc.tivoli.com /pubs
```

**Note:** A "best practices" recommendation is to always use the fully qualified domain name of the back-end server when specifying the argument to the –h option.

Creating TCP type junctions

A WebSEAL junction over a TCP connection provides the basic properties of a junction but does not provide secure communication across the junction.

To create a secure TCP junction and add an initial server, use the `create` command with the –t tcp option:

```
pdadmin> server task webseald-instance-name create -t tcp -h host-name \ [-p port] jct_point
```

The default port value for a TCP junction (if not specified) is 80.

Creating SSL type junctions

SSL junctions function exactly like TCP junctions, with the added value that all communication between WebSEAL and the back-end server is encrypted.

SSL junctions allow secure end-to-end browser-to-application transactions. You can use SSL to secure communications from the client to WebSEAL and from WebSEAL to the back-end server. The back-end server must be HTTPS-enabled when you use an SSL junction.

To create a secure SSL junction and add an initial server, use the `create` command with the –t ssl option:

```
pdadmin> server task instance_name-webseald-host-name create -t ssl -h host_name \ [-p port] jct_point
```

The default port value for an SSL junction (if not specified) is 443.

Verifying the back-end server certificate

When a client makes a request for a resource on the back-end server, WebSEAL, in its role as a security server, performs the request on behalf of the client. The SSL
protocol specifies that when a request is made to the back-end server, that server must provide proof of its identity using a server-side certificate.

When WebSEAL receives this certificate from the back-end server, it must verify its authenticity by matching the certificate against a list of root CA certificates stored in its certificate database.

Tivoli Access Manager uses the IBM Global Security Kit (GSKit) implementation of SSL. You must use the GSKit iKeyman utility to add the root certificate of the CA who signed the back-end server certificate to the WebSEAL certificate keyfile (pdsvr.kdb).

**Examples of SSL junctions**

Junction host `sales.tivoli.com` at junction point `/sales` over SSL:

```bash
pdadmin> server task web1-webseald-cruz create -t ssl -h sales.tivoli.com /sales
```

**Note:** In the above example, the `-t ssl` option dictates a default port of 443.

Junction host `travel_svr` on port 4443 at junction point `/travel` over SSL:

```bash
pdadmin> server task web1-webseald-cruz create -t ssl -p 4443 -h travel_svr /travel
```

**Disabling SSL protocol versions for junctions**

You can optionally disable one or more SSL protocol versions for junction connections. By default, the supported SSL versions are enabled. The WebSEAL configuration file provides the following entries by default:

```
[junction]
disable-ssl-v2 = no
disable-ssl-v3 = no
disable-tls-v1 = no
```

To disable an SSL protocol version for junctions, set the corresponding entry to yes.

**Adding back-end servers to a junction**

To increase high availability of the resources protected by Tivoli Access Manager, you can junction multiple replica back-end servers to the same junction point.

- Multiple back-end servers junctioned to the same point must have identical WebSEAL versions and identical Web document spaces.
- Multiple back-end servers junctioned to the same point must use the same connection type (TCP or SSL).
- WebSEAL uses a least busy algorithm to determine which back-end server replica has the fewest number of request connections and forwards any new request to that server.

Create the initial junction. For example:

```bash
pdadmin> server task web1-webseald-cruz create -t tcp -h server1 /sales
```

Add an additional back-end server replica. For example:

```bash
pdadmin> server task web1-webseald-cruz add -h server2 /sales
```
Mutually authenticated SSL junctions

WebSEAL supports mutual authentication between a WebSEAL server and a back-end server over an SSL junction (–t ssl or –t sslproxy). The following outline summarizes the supported functionality for mutual authentication over SSL (command options are listed where appropriate):

1. WebSEAL authenticates the back-end server (normal SSL process)
   - WebSEAL validates the server certificate from the back-end server. See “WebSEAL validates back-end server certificate.”
   - WebSEAL verifies the Distinguished Name (DN) contained in the certificate (–D) (optional, but highly recommended) See “Distinguished name (DN) matching.”

2. Back-end server authenticates WebSEAL (two methods)

The command options that control mutual authentication over SSL provide the following features:
- You can specify client certificate or BA authentication method.
- You can apply authentication methods on a per-junction basis.

Special considerations for combining the –b options (for handling BA information) with mutual authentication over SSL are described in “Handling client identity information across junctions” on page 284.

WebSEAL validates back-end server certificate

WebSEAL verifies a back-end server certificate according to the standard SSL protocol. The back-end server sends its server certificate to WebSEAL. WebSEAL validates the server certificate against a pre-defined list of root Certificate Authority (CA) certificates.

The Certificate Authority (CA) certificates that form the trust chain for the application server certificate (from the signing CA up to and including the root certificate) must be included in the key database in use by WebSEAL.

You use the iKeyman utility to create and manage the database of root CA certificates.

Distinguished name (DN) matching

You can enhance server-side certificate verification through Distinguished Name (DN) matching. To enable server DN matching, you must specify the back-end server DN when you create the SSL junction to that server. Although DN matching is an optional configuration, it is highly recommended that you implement this feature with mutual authentication over SSL junctions.

During server-side certificate verification, the DN contained in the certificate is compared with the DN defined by the junction. The connection to the back-end server fails if the two DNs do not match.
To enable the server DN matching, specify the back-end server DN when you create the SSL junction using the -D "DN" option. To preserve any blank spaces in the string, surround the DN string with double quotation marks. For example:

-D "/C=US/O=Tivoli/OU=SecureWay/CN=Access Manager"

The -D option is appropriate only when used with the -K or -B option.

**WebSEAL authenticates with client certificate**

Use the -K option to enable WebSEAL to authenticate to the junctioned back-end server using its client certificate.

-K "key_label"

The conditions for this scenario include:

- The back-end server is set up to require verification of WebSEAL’s identity with a client certificate.
- Using the iKeyman utility to create, label, and store a special key that is used solely as WebSEAL’s client certificate when authenticating to a junctioned back-end server.
- It is also highly recommended that you configure the junction for DN matching (-D).

The -K option uses an argument that specifies the key-label of the required certificate as stored in the GSKit key database. Use the iKeyman utility to add new certificates to the key database.

You must surround the key-label argument with quotation marks. For example:

-K "cert1_Tiv"

If the key resides on cryptographic hardware, you must specify the WebSEAL token device with the key label.

-K "token_name:key-label"

For example:

-K "websealtoken:junctionkey"

See "Cryptographic hardware for encryption and key storage” on page 32.

See "Configuring WebSEAL key database parameters” on page 229.

**WebSEAL authenticates with BA header**

Use the -B -U "username" -W "password" option to enable WebSEAL authentication via Basic Authentication.

-B -U "username" -W "password"

The conditions for this scenario include:

- The back-end server is set up to require verification of WebSEAL’s identity with a BA header.
- Do not configure the junction with any -b option. (Internally, however, the -B option uses -b filter.)
- WebSEAL is configured to pass its identity information in a BA header to authenticate to the back-end server.
It is highly recommended that you also configure the junction for DN matching (-D).

You must surround the user name and password arguments with double quotation marks. For example:
-U "WS1" -W "abCde"

**Handling client identity information across junctions**

A junction can be set up to specify client identity information in BA headers. The -b option allows four possible arguments: filter, supply, ignore, gso. You can find detailed information about these arguments in "Configuring BA headers for single sign-on solutions" on page 316.

The -b option has an impact on the junction settings for mutual authentication and you must consider the correct combination of options.

**Using –b supply**

- WebSEAL authentication via BA header is not allowed with this option. This option uses the BA header for the original client user name and a "dummy" password.
- WebSEAL authentication via client certificate is allowed with this option.

**Using –b ignore**

- WebSEAL authentication via BA header is not allowed with this option. This option uses the BA header for the original client user name and password.
- WebSEAL authentication via client certificate is allowed with this option.

**Using –b gso**

- WebSEAL authentication via BA header is not allowed with this option. This option uses the BA header for user name and password information supplied by the GSO server.
- WebSEAL authentication via client certificate is allowed with this option.

**Using –b filter**

- Internally, the –b filter option is used when WebSEAL authentication is set to use BA header information.
  The WebSEAL BA header is used for all subsequent HTTP transactions. To the back-end server, WebSEAL appears logged on at all times.
- WebSEAL authentication via client certificate is allowed with this option.
- If the back-end server requires actual client identity (from the browser), the CGI variables HTTP_IV_USER, HTTP_IV_GROUP, and HTTP_IV_CREDS can be used. For scripts and servlets, use the corresponding Tivoli Access Manager-specific HTTP headers: iv-user, iv-groups, iv-creds.
Creating TCP and SSL proxy junctions

You can create WebSEAL junctions that allow communication to traverse network topologies that use HTTP or HTTPS proxy servers. You can configure the junction to handle requests as standard TCP communication or protected SSL communication.

The create command requires one of the following arguments to the type option to establish either a TCP-based or SSL-based junction through a proxy server:

- `–t tcpproxy`
- `–t sslproxy`

Both create and add commands require the following options and arguments to identify the proxy server and the target Web server:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–H host-name</td>
<td>The DNS host name or IP address of the proxy server.</td>
</tr>
<tr>
<td>–P port</td>
<td>The TCP port of the proxy server.</td>
</tr>
<tr>
<td>–h host-name</td>
<td>The DNS host name or IP address of the target Web server.</td>
</tr>
<tr>
<td>–p port</td>
<td>The TCP port of target Web server. Default is 80 for TCP junctions; 443 for SSL junctions.</td>
</tr>
</tbody>
</table>

Example TCP proxy junction (entered as one line):
```
padmin> server task web1-webseald-cruz create -t tcpproxy \ 
-H clipper -P 8081 -h www.ibm.com -p 80 /ibm
```

Example SSL proxy junction (entered as one line):
```
padmin> server task web1-webseald-cruz create -t sslproxy \ 
-H clipper -P 8081 -h www.ibm.com -p 443 /ibm
```

![Figure 11. Example proxy junction](image-url)
WebSEAL-to-WebSEAL junctions over SSL

Tivoli Access Manager supports SSL junctions between a front-end WebSEAL server and a back-end WebSEAL server. Use the –C option with the create command to junction the two WebSEAL servers over SSL and provide mutual authentication.

Example:
```
padmin> server task webl-webseald-cruz create -t ssl -C -h serverA /jctA
```

Mutual authentication occurs in the following two stages:

- The SSL protocol allows the back-end WebSEAL server to authenticate to the front-end WebSEAL server through its server certificate.
- The –C option enables the front-end WebSEAL server to pass its identity information to the back-end WebSEAL server in a Basic Authentication (BA) header.

Additionally, the –C option enables single sign-on functionality provided by the –c option. The –c option allows you to place Tivoli Access Manager-specific client identity and group membership information into the HTTP header of the request destined for the back-end WebSEAL server. The header parameters include iv-user, iv-groups, and iv-creds. See “Supplying client identity in HTTP headers (–c)” on page 298.

The following conditions apply to WebSEAL-to-WebSEAL junctions:

- The junction is appropriate only with the –t ssl or –t sslproxy junction type.
- Both WebSEAL servers must share a common LDAP registry. This allows the back-end WebSEAL server to authenticate the front-end WebSEAL server identity information.
- If the WebSEAL-to-WebSEAL junction and the back-end application server junction both use the –j junction option (for junction cookies), a naming conflict can occur between the two junction cookies created by each of the two WebSEAL servers. (Refer to the diagram at the beginning of this section.) To prevent this conflict, you must configure the intermediary WebSEAL server (WebSEAL 2 in the diagram) to uniquely identify its junction cookie. On the intermediary WebSEAL server only, set the hostname-junction-cookie parameter in the [script-filtering] stanza of the WebSEAL configuration file to “yes” (default is “no”):
```
[script-filtering]
hostname-junction-cookie = yes
```

Junction cookies allow WebSEAL to handle server-relative URLs generated on the client-side. These URLs lack knowledge of the junction point of the destination application. The junction cookie provides this information. For complete information on junction cookies, see “Handling server-relative URLs with junction cookies (–j)” on page 292.
Modifying URLs to back-end resources

Pages returned to the client from back-end applications most often contain URL links to resources located on those application servers. It is important that these links are constructed to direct any requests back to the correct locations of these resources.

For example (non-WebSEAL environment), the URL entered by a client for a resource on an application server might appear as follows:
http://www.abc.com/file.html

WebSEAL, as a front-end reverse proxy, provides security services to back-end application servers via the WebSEAL junctioning feature. This feature results in resources being accessed via different URL expressions.

For example (WebSEAL environment), the URL entered by a client for the same resource on a junctioned back-end application server must appear as follows:
http://webseal.abc.com/jct/file.html

The junction feature of WebSEAL changes the server and path information that must be used to access resources on junctioned back-end systems. A link to a resource on a back-end junctioned server only succeeds if the URL contains the identity of the junction.

To support the junction feature and maintain the integrity of URLs, WebSEAL must, where possible:
1. Modify the URLs (links) found in responses sent to clients
2. Modify requests for resources resulting from URLs (links) that WebSEAL could not change

Note that WebSEAL’s rules and mechanisms for filtering and processing URLs do not apply to links that point to resources external to the Tivoli Access Manager junctioned environment.

The following diagram summarizes the solutions available to WebSEAL for modifying URLs to junctioned back-end resources:

![Diagram summarizing URL modifications]

Figure 12. Summary: Modifying URLs to back-end resources

This section contains the following topics:

- “Understanding path types used in URLs” on page 288
- “Filtering URLs in responses” on page 288
Understanding path types used in URLs

Any HTML page is likely to contain URLs (links) to other resources on that back-end server or elsewhere. URL expressions can appear in the following formats:

- relative
- server-relative
- absolute

Links containing URLs expressed in **relative** format never require any manipulation by WebSEAL. By default, the browser handles relative URLs in links by prepending the correct scheme, server, and directory information (including the junction) to the relative URL. The prepended information is derived from the request URL for the page on which the link is located.

Example **relative** URL expressions:

```
abc.html ../abc.html ./abc.html sales/abc.html
```

However, difficulties arise with **server-relative** and **absolute** path formats. Links to back-end resources expressed in absolute or server-relative formats succeed only if WebSEAL was able to modify the URL path expression and include junction information.

Example **server-relative** URL expressions:

```
/abc.html /accounts/abc.html
```

Example **absolute** URL expression:

```
http://www.tivoli.com/abc.html
```

**Note:** All programmers of Web scripts are encouraged to use **relative** links (not absolute or server-relative) for dynamically generated URLs.

Filtering URLs in responses

This section describes how WebSEAL filters URLs in responses from junctioned back-end application servers.

- “Standard URL filtering rules for WebSEAL” on page 288
- “Modifying absolute URLs with script filtering” on page 290
- “Filtering changes the Content-Length header” on page 290
- “Limitation with unfiltered server-relative links” on page 291

**Standard URL filtering rules for WebSEAL**

WebSEAL uses a set of standard rules to filter URLs contained in pages that are responses to client requests. To apply standard URL filtering, WebSEAL must be able to “see” the URLs on a page sent from the back-end server. WebSEAL cannot use standard filtering rules for URLs embedded in scripts.

By default, WebSEAL filters only documents of MIME type "text/html" and "text/vnd.wap.wml" that are received from junctioned servers. Additional MIME types can be configured using the [filter-content-types] stanza of the WebSEAL configuration file.
Relative URLs are always handled appropriately by the browser. By default, the browser handles relative URLs in links by prepending the correct scheme, server, and directory information (including the junction) to the relative URL. The prepended information is derived from the request URL for the page on which the link is located.

However, WebSEAL must add the junction name to the path of absolute and server-relative URLs that refer to resources located on junctioned servers.

- **Server-relative URLs** indicate a URL position in relation to the document root of the junctioned server, for example:
  
  `/dir/file.html`

  Server-relative URLs are modified by adding the junction point of the junctioned server to the path name. For example:

  `/jct/dir/file.html`

- **Absolute URLs** indicate a URL position in relation to a host name or IP address (and, optionally, a network port). For example:

  `http://host-name[:port]/file.html`, or

  `https://host-name[:port]/file.html`

  Absolute URLs are modified according to the following set of rules:

  - If the URL is HTTP and the host/port matches a TCP junctioned server, the URL is modified to be server-relative to WebSEAL and reflect the junction point. For example:

    `http://host-name[:port]/file.html`

    becomes:

    `/tcpjct/file.html`

  - If the URL is HTTPS and the host/port matches an SSL junctioned server, the URL is modified to be server-relative to WebSEAL and reflect the junction point. For example:

    `https://host-name[:port]/file.html`

    becomes:

    `/ssljct/file.html`

In addition:

- Only URLs in content types defined in the `[filter-content-types]` stanza of the WebSEAL configuration file are filtered.
- META tags are always filtered for refresh requests, for example:

  `<META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://server/url">`

- If a BASE tag contains an HREF attribute, the tag is removed from the response to the client.

Parameters for filtering URLs through junctioned servers are located in the `[filter-url]` stanza of the WebSEAL configuration file. The `[filter-url]` stanza contains a list of HTML tags that the WebSEAL server filters or modifies to adjust absolute URLs obtained through a junctioned server.

All commonly used HTML tags are configured by default. The administrator may need to add additional HTML tags that contain URLs.

---

Chapter 10. WebSEAL junctions 289
Modifying absolute URLs with script filtering

WebSEAL requires additional configuration to handle the processing of absolute URLs embedded in scripts. Web scripting languages include Javascripts, VBscripts, ASP, JSP, ActiveX, and others. The WebSEAL configuration file contains a parameter that enables or disables filtering of embedded absolute URLs:

```
[script-filtering]
script-filter = no
```

Script-filtering is disabled by default. To enable script filtering, set:

```
script-filter = yes
```

The `script-filter` mechanism expects absolute URLs with a standard scheme, server, resource format:

```
http://server/resource
```

The `script-filter` mechanism replaces the scheme and server portions of the link with the correct junction information (as a relative pathname):

```
/junction-name/resource
```

This filtering solution parses a script embedded in HTML code and therefore requires additional processing overhead that can negatively impact performance. Limit your use of the `script-filter` parameter only to junctions that require support for filtering of embedded absolute URLs.

The following diagram illustrates this absolute URL filtering solution:

![Diagram illustrating absolute URL filtering solution]

**Figure 13. Filtering absolute URLs**

You can optionally configure WebSEAL to rewrite the original absolute URL as an absolute URL, instead of a relative URL (default). To enable this functionality, set the `rewrite-absolute-with-absolute` parameter in the `[script-filtering]` stanza of the WebSEAL configuration file to equal "yes":

```
[script-filtering]
rewrite-absolute-with-absolute = yes
```

With this functionality enabled, the example URL in the diagram above would appear as follows:

```
http://webseal-hostname/jctA/abc.html
```

**Filtering changes the Content-Length header**

Normally, the Content-Length header in a response from a back-end server indicates the size of the content being returned. When WebSEAL filters URLs and
adds junction information to the path of URLs contained in the page, the actual size of the page becomes larger than indicated in the Content-Header.

WebSEAL has no way of knowing what the new content length is until it actually writes the stream to the client. At this point, it is too late to insert a new Content-Length header. WebSEAL responds to this situation in the following manner:

1. WebSEAL places the value of the original Content-Length header in a new header called X-Old-Content-Length.
   Any applets or applications written to look for this header can have access to the original (pre-filtered) Content-Length value.
2. WebSEAL logs the modified (post-filtered) Content-Length value in the request.log file.
3. The Content-Length header no longer appears.

**Limitation with unfiltered server-relative links**

WebSEAL provides solutions for processing client-side, script-generated, server-relative URLs to resources on back-end junctioned application servers. These server-relative URLs, generated on the client-side by applets and scripts, initially lack knowledge of the junction point in the path expression. During a client request for a resource, WebSEAL can attempt to reprocess a server-relative URL using junction cookies or a junction mapping table.

However, before the processing takes place, the request actually specifies a resource located on the local Web space of the WebSEAL server itself. The corrective reprocessing of the URL occurs only after WebSEAL receives the request and performs an ACL check. An ACL check on this unprocessed request (specifying an incorrect or nonexistent local resource) could result in an error that stops the fulfillment of the intended request.

For example, the following sequence takes place during processing:

1. The client makes a request for a resource using a client-side, script-generated, server-relative URL.
2. The server-relative URL is received by WebSEAL as a request.
   The unprocessed URL specifies a resource located in the Web space of the WebSEAL server itself (obviously, this is not the intended resource).
3. WebSEAL performs an ACL check on this local resource specified in the request URL.
   - If the ACL check fails, all processing of the request stops and the client receives a 403 error (Forbidden). This error occurs because the ACL check was performed for the incorrect (and probably nonexistent) resource.
   - If the ACL check succeeds and the resource exists in the local Web space, it is returned. This error results in the client receiving the incorrect resource.
   - If the ACL check succeeds and the resource does not exist in the local Web space, WebSEAL modifies the request URL (using the junction cookie or junction mapping table method) and performs an internal reprocessing of the request. This behavior is correct.
4. WebSEAL performs another ACL check on the modified URL that contains the corrected path that includes the junction point. This modified URL now allows an ACL check for the correct resource.

**Workaround**
To solve this problem:

1. Always write scripts that generate relative URL links. Avoid absolute and server-relative URL links.
2. If you must use server-relative links, do not duplicate resource names and paths on both the WebSEAL server and the junctioned application server.
3. If you must use server-relative links, design your ACL model so that more prohibitive ACLs do not affect false resources specified by unfiltered URLs.

### Processing URLs in requests

A difficulty arises when URLs are dynamically generated by client-side applications (applets) or embedded in scripts in the HTML code. Web scripting languages include Javascripts, VBscripts, ASP, JSP, ActiveX, and others. These applets and scripts execute as soon as the page has arrived at the client browser. WebSEAL never has a chance to apply its standard filtering rules to these dynamically generated URLs.

This section describes how WebSEAL processes client-side dynamically generated server-relative links found in requests for resources on junctioned back-end servers.

- “Handling server-relative URLs with junction cookies (-j)” on page 292
- “Handling server-relative URLs with junction mapping” on page 293
- “Processing root junction requests” on page 294

**Note:** There are no solutions available for handling absolute URLs generated on the client-side.

#### Handling server-relative URLs with junction cookies (-j)

Server-relative URLs generated on the client-side by applets and scripts initially lack knowledge of the junction point. WebSEAL cannot filter the URL because it is generated on the client-side. During a client request for a resource using this URL, WebSEAL can attempt to reprocess the server-relative URL using junction cookies.

In the following scenario, a script located on the requested page dynamically generates a server-relative URL expression upon arrival to the browser. If the client requests the resource specified by this link, WebSEAL receives a request for a local page. After failing to find the page, it returns a “Not Found” error to the client.

The -j option provides a cookie-based solution for handling server-relative URLs that are dynamically generated by a script that runs on the client machine.

**General syntax:**

```
pdadmin> server task server-name create ... -j ...
```

For each requested page, a "junction-identifier" cookie is sent to the client (as embedded Java script on the HTML page). The cookie contains the following header name and value:

```
IV_JCT = /junction-name
```

When the client makes a request from this page using a dynamically generated server-relative URL, WebSEAL (as before) receives a request for a local resource. When it fails to locate the resource, WebSEAL immediately retries the request using the junction information supplied by the cookie. With the correct junction information in the URL expression, the resource is successfully located.
The following diagram illustrates this solution for filtering server-relative URLs.

![Diagram](image-url)

**Figure 14. Processing server-relative URLs**

See also “Handling cookies from servers across multiple -j junctions” on page 295 and “Processing root junction requests” on page 294.

WebSEAL provides an alternative, non-cookie-based solution for handling dynamically generated server-relative URLs. See “Handling server-relative URLs with junction mapping” on page 293.

**Handling server-relative URLs with junction mapping**

Server-relative URLs generated on the client-side by applets and scripts initially lack knowledge of the junction point. WebSEAL cannot filter the URL because it is generated on the client-side. During a client request for a resource using this URL, WebSEAL can attempt to reprocess the server-relative URL using junction mapping.

Tivoli Access Manager provides an alternative to the cookie-based solution for filtering dynamically generated server-relative URLs. You can create and activate a junction mapping table that maps specific target resources to junction names.

WebSEAL checks the location information in the server-relative URL with the data contained in the junction mapping table. If the path information in the URL matches an entry in the table, WebSEAL directs the request to the junction associated with that location.

The table is an ASCII text file called `jmt.conf`. The location of this file is specified in the [junction] stanza of the WebSEAL configuration file:

```
jmt-map = lib/jmt.conf
```

The format for data entry in the table consists of the junction name, a space, and the resource location pattern. You can also use wildcard characters to express the resource location pattern.

In the following example of the junction mapping configuration file, two back-end servers are junctioned to WebSEAL at /jctA and /jctB:

```
# jmt.conf
#junction-name resource-location-pattern
/jctA /documents/release-notes.html
/jctA /travel/index.html
/jctA /accounts/*
/jctB /images/weather/*.*.jpg
```

Chapter 10. WebSEAL junctions 293
You must create the jmt.conf mapping table. This file does not exist by default. After you create the file and add data, use the jmt load command to “load” the data so that WebSEAL has knowledge of the new information.

```
pdadmin> server task server-name jmt load
JMT table successfully loaded.
```

The following conditions apply to the junction mapping table solution:

- This solution does not require the -j option or junction cookie
- The mapping table requires setup and activation by a security administrator
- This solution does not handle links created with absolute URLs
- Resource location pattern matching must be unique across the local Web space and across junctioned Web application servers
- If there is a duplicate pattern entry in the file, the mapping table does not load. However, WebSEAL continues to run.
- If there is an error loading the mapping table, the mapping table is not available. However, WebSEAL continues to run.
- If the mapping table is empty or there is an error in the table entries, the mapping table does not load. However, WebSEAL continues to run.
- Any errors that occur while loading the mapping table result in serviceability entries in the WebSEAL server log file (webseald.log).

See also “Processing root junction requests.”

### Processing root junction requests

You can specify how WebSEAL responds to requests for resources located at the root (“/”) junction. WebSEAL can either process the request immediately, or can attempt to identify a junction point to which to send the request. Requests are sent by using a junction mapping mechanisms such as the JMT or IV_JCT cookie.

Avoiding root junction processing prevents processing being performed for incorrect resources before the intended resource is identified. This has performance benefits and prevents false authorization or file type check failures.

To configure root junction processing, set the process-root-requests entry in the [server] stanza in the WebSEAL configuration stanza.

```
[server]
process-root-requests = always
```

Valid values are:

- never
  Root junction requests are never processed at the root junction. When a junction mapping mechanism is configured, such requests are immediately mapped, then processed at a mapped junction point if one is identified.
- always
  Always attempt to process requests for the root junction at the root junction first before attempting to use a junction mapping mechanism. This is not recommended unless the root junction serves a large set of resources or no junction mapping mechanisms are configured for the set of junctions served by this WebSEAL server.
- filter
  All root junction requests are examined to determine whether they start with the patterns specified in the [process-root-filter] stanza. If they do, they are
processed at the root junction first. If they do not start with patterns specified in the [process-root-filter] stanza, they are remapped immediately.

When process-root-requests = filter, you must specify the patterns for which you want root junction requests processed at the root junction. Use the [process-local-filter stanza]. The syntax for specifying a pattern is:

\[ \text{root = pattern} \]

Pattern must be a standard WebSEAL wildcard pattern. For example:

\[
\text{[process-local-filter]}
\text{root = /index.html}
\text{root = /cgi-bin*}
\]

For information on junction mapping mechanisms, see the following sections:

- “Handling server-relative URLs with junction cookies (-j)” on page 292
- “Handling server-relative URLs with junction mapping” on page 293

**Handling cookies from servers across multiple -j junctions**

This section describes WebSEAL’s default handling of cookies generated by back-end applications and returned to clients across -j junctions.

**Part 1: -j junctions modify Set-Cookie Path attributes**

In addition to providing a junction identifier cookie to the browser, a junction configured with the -j option also supports the handling of cookies sent with responses from the back-end application.

**Browser rule:**

If a Set-Cookie header in a response from the server contains a Path attribute (such as \[Path=/xyz\]), the browser returns the cookie only when a request URL (activated from the returned page) begins with this path (such as \[/xyz/memo.html\]).

**Problem:**

When the junction environment contains mixed solutions for handling visible and embedded URLs in responses, the ability of the browser to return cookies is compromised. For example, standard WebSEAL filtering of visible server-relative URLs normally adds the junction name to the Path attribute of a server cookie (for example, \[Path=/jct/xyz\]) in addition to modifying the URL itself. This match between URL path name and the cookie Path attribute allows the browser to return the cookie when the link is activated by the user.

However, the -j junction-cookie-based solution adds the junction name to a URL only after the link (URL) has been activated by the user. When the pre-modified link is activated, the URL path name (/xyz/memo.html) does not match the Path attribute (Path=/jct/xyz). The server cookie is not returned.

**Solution:**

The -j option converts the Path attribute for any server cookie (Set-Cookie) to "/" (for example, \[Path=/\]). Because all server-relative path names begin with a "/", all server cookies are returned regardless of the requirements of the original Path attribute specifications.
Part 2: -j junctions modify Set-Cookie Name attributes
The -j option also supports cookies returned from servers across multiple junctions.

Browser rule:

Browsers always replace any stored cookie with a newly arrived cookie containing the same Name attribute (Set-Cookie), unless the Path or Domain attributes, or both are unique.

Problem:

The previous section describes how the -j junction option modifies the Path attribute of a Set-Cookie header to allow the browser to return cookies in an environment where WebSEAL is applying different filtering rules for visible and embedded URLs contained in the response page.

In a scenario where multiple back-end servers are connected to WebSEAL across different junctions (such as in a WebSphere environment), it is possible for each server to send cookies (Set-Cookie) with the same Name attribute. If the junctions use the -j option, the Path attributes for each cookie become identical (Path=/). Because the same WebSEAL server is the point of contact for the browser, the Domain attribute likewise becomes identical. Although these identical cookies arrive from unique back-end applications, the browser overwrites the identically named cookies.

Solution:

The -j junction option provides an additional feature that uniquely renames any cookie returned with a response from a back-end application server. The Name attribute of a Set-cookie header is prepended with a special string. The string contains the name of the specific junction responsible for delivering the response (with cookie).

AMWEBJCT_jct-name_

For example, if a cookie with the Name "JSESSIONID" arrives across a junction named /jctA, the cookie Name is changed to:

AMWEBJCT_jctA_JSESSIONID

Refer also to "Preserving cookie names" on page 296

Preserving cookie names
By default, WebSEAL modifies the names of cookies (returned in responses from back-end applications) across junctions created with the -j option or listed in the junction mapping table. This functionality is described in the preceding section. In the scenario just described, WebSEAL creates unique cookie names to prevent possible naming conflicts with cookies returned across other -j junctions.

However, if front-end browsers and applications depend on the specific cookie name generated by the application, you can disable this default cookie renaming functionality for specific cookies. The name parameter in the [preserve-cookie-names] stanza of the WebSEAL configuration file allows you to list the specific cookie names that are not to be renamed by WebSEAL:

[preserve-cookie-names]
name = cookie-name1
name = cookie-name2
Additional junction options

You can provide the following additional WebSEAL junction functionality with additional options:

- “Forcing a new junction (–f)” on page 297
- “Supplying client identity in HTTP headers (–c)” on page 298
- “Supplying client IP addresses in HTTP headers (–r)” on page 299
- “Limiting the size of WebSEAL-generated HTTP headers” on page 299
- “Passing session cookies to junctioned portal servers (–k)” on page 300
- “Supporting case-insensitive URLs (–i)” on page 300
- “Stateful junction support (–s, –u)” on page 301
- “Specifying back-end server UUIDs for stateful junctions (–u)” on page 302
- “Junctioning to Windows file systems (–w)” on page 304

Forcing a new junction (–f)

You must use the –f option when you want to force a new junction to overwrite an existing junction.

The following example (server instance name = cruz) illustrates this procedure:

1. Login to pdadmin:
   # pdadmin
   pdadmin> login
   Enter User ID: sec_master
   Enter Password:
   pdadmin>

2. Use the server task list command to display all current junction points:
   pdadmin> server task web1-webseald-cruz list
   /

3. Use the server task show command to display details of the junction:
   pdadmin> server task web1-webseald-cruz show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /opt/pdweb/www/docs

4. Create a new local junction to replace the current junction point (the -f option is required to force a new junction that overwrites an existing junction):
   pdadmin> server task web1-webseald-cruz create -t local -f -d /tmp/docs /
   Created junction at /

5. List the new junction point:
   pdadmin> server task webseald-cruz list
   /

6. Display the details of this junction:
   pdadmin> server task webseald-cruz show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /tmp/docs

Chapter 10. WebSEAL junctions 297
Supplying client identity in HTTP headers (–c)

The –c option allows you to insert Tivoli Access Manager-specific client identity and group membership information into the HTTP headers of requests destined for junctioned third-party servers. The HTTP header information enables applications on junctioned third-party servers to perform user-specific actions (such as single sign-on) based on the client’s Tivoli Access Manager identity.

HTTP header information must be transformed by the back-end server to environment variable format for use by a service on the back-end server. Header information is transformed into a CGI environment variable format by replacing all dashes (-) with under bars (_) and prepending "HTTP" to the beginning of the string. The value of the HTTP header becomes the value of the new environment variable.

<table>
<thead>
<tr>
<th>PD-specific HTTP Header Fields</th>
<th>CGI Environment Variable Equivalents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv-user =</td>
<td>HTTP_IV_USER =</td>
<td>The short or long name of the client. Defaults to &quot;Unauthenticated&quot; if client is unauthenticated (unknown).</td>
</tr>
<tr>
<td>iv-groups =</td>
<td>HTTP_IV_GROUPS =</td>
<td>A list of groups to which the client belongs. Consists of comma separated quoted entries.</td>
</tr>
<tr>
<td>iv-creds =</td>
<td>HTTP_IV_CREDS =</td>
<td>Encoded opaque data structure representing an Tivoli Access Manager credential. Supplies credentials to remote servers so mid-tier applications can use the authorization API to call the authorization service. Refer to the IBM Tivoli Access Manager Authorization C API Developer’s Reference.</td>
</tr>
</tbody>
</table>

The Tivoli Access Manager-specific HTTP header entries are available to CGI programs as the environment variables HTTP_IV_USER, HTTP_IV_GROUPS and HTTP_IV_CREDS. For other application framework products, see the product’s documentation for instructions on extracting headers from HTTP requests.

–c syntax

The –c option specifies what Tivoli Access Manager-specific HTTP header data is sent to the back-end application server.

–c header-types

The header-types arguments include: all, iv_user, iv_user_l, iv_groups, and iv_creds.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv_user</td>
<td>Provides the user name (short form) as the iv-user field in the HTTP header of the request.</td>
</tr>
<tr>
<td>iv_user_l</td>
<td>Provides the full DN of the user (long form) as the iv-user field in the HTTP header of the request.</td>
</tr>
<tr>
<td>iv_groups</td>
<td>Provides the user’s list of groups as the iv-groups field in the HTTP header of the request.</td>
</tr>
<tr>
<td>iv_creds</td>
<td>Provides the user’s credential information as the iv-creds field in the HTTP header of the request.</td>
</tr>
</tbody>
</table>

Note: Use either iv_user or iv_user_l, but not both.
The -c all option inserts all three types of identity information into the HTTP header (the short name format (iv_user) is used in this case).

Note: Separate multiple arguments with commas only. Do not enter any spaces.

Examples:
- c all
- c iv_creds
- c iv_user,iv_groups
- c iv_user_l,iv_groups,iv_creds

Note: To ensure security of the iv_creds value, use SSL junctions.

Note: Refer also to the WebSEAL section of the IBM Tivoli Access Manager Performance Tuning Guide for a description of how to configure environment variables that cache –c junction information. It is possible to improve WebSEAL performance under –c junction conditions by applying this caching configuration.

Supplying client IP addresses in HTTP headers (–r)
The –r option allows you to insert client IP address information into the HTTP headers of requests destined for junctioned application servers. The HTTP header information enables applications on junctioned third-party servers to perform actions based on this IP address information.

HTTP header information must be transformed by the back-end server to environment variable format for use by a service on the back-end server. Header information is transformed into a CGI environment variable format by replacing all dashes (-) with under bars (_) and prepending "HTTP" to the beginning of the string. The value of the HTTP header becomes the value of the new environment variable.

Note: The value of the IP address does not always represent the address of the originating client machine. The IP address value could represent the address of a proxy server or a network address translator (NAT).

<table>
<thead>
<tr>
<th>PD-specific HTTP Header Field</th>
<th>CGI Environment Variable Equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv-remote-address</td>
<td>HTTP_IV_REMOTE_ADDRESS</td>
<td>The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT).</td>
</tr>
</tbody>
</table>

The –r option specifies that the IP address of the incoming request be sent to the back-end application server. The option is expressed without any arguments.

Limiting the size of WebSEAL-generated HTTP headers
You can limit the size of WebSEAL-generated HTTP headers that are inserted in requests to junctioned back-end servers. The max-webseal-header-size parameter in the [junction] stanza of the WebSEAL configuration file specifies the maximum size, in bytes, of WebSEAL-generated HTTP headers. A value of "0" disables this function.
This parameter can be useful if a back-end application server rejects WebSEAL-generated HTTP headers because they are too large. For example, an iv-creds header for a user belonging to many groups could be too large.

When configured, this parameter causes WebSEAL-generated headers exceeding the maximum value to split across multiple headers. The following example output from a CGI application illustrates the effect of split headers:

HTTP_IV_CREDS_1=Version=1,BAKs3DCCBnMMADCCBmOwgg2pAgIDkDCCAYuWwKzA
HTTP_IV_CREDS_2=+0+8eAgI8iAICEdYCAgCkAgFUBAaSVNCJqncMOWNuPXN1Y2I==
HTTP_IV_CREDS_SEGMENTS=2

If you enable this function, you must modify the back-end application to recognize split headers, instead of standard WebSEAL-specific HTTP headers.

**Passing session cookies to junctioned portal servers (–k)**

A Web portal is a server that offers a broad array of personalized resources and services. The –k option allows you to send the Tivoli Access Manager session cookie (originally established between the client and WebSEAL) to a back-end portal server. This option currently exists to directly support the integration of WebSEAL with the Plumtree Corporate Portal solution.

When a client requests a personal resource list from the portal server, the portal server builds this list by accessing resources located on other supporting application servers, also protected by WebSEAL. The session cookie allows the portal server to perform seamless single sign-on to these application servers, on behalf of the client.

You include the –k option, without arguments, when you create the junction between WebSEAL and the back-end portal server.

Conditions to consider for a portal server configuration:

- For access via user name and password, Forms authentication is required. Do not use Basic Authentication (BA).
- The ssl-id-sessions parameter in the [session] stanza of the WebSEAL configuration files must be set to "no". For HTTPS communication, this setting forces the use of a session cookie, instead of the SSL session ID, to maintain session state.
- If the portal server is front-ended by a WebSEAL cluster, enable the failover type cookie. The failover cookie contains encrypted credential information that allows authentication to succeed with any replicated WebSEAL server that processes the request.

**Supporting case-insensitive URLs (–i)**

By default, Tivoli Access Manager treats URLs as case-sensitive when performing checks on access controls. The –i option is used to specify that WebSEAL treat URLs as case-insensitive when performing authorization checks on a request to a junctioned back-end server.

When you set this option on the junction, WebSEAL does not distinguish between uppercase and lowercase characters when parsing URLs. By default, Web servers are expected to be case-sensitive.
Although most HTTP servers support the HTTP specification that defines URLs as case-sensitive, some HTTP servers treat URLs as case-insensitive.

For example, on case-insensitive servers, the following two URLs:
http://server/sales/index.htm
http://server/SALES/index.HTM

are viewed as the same URL. This behavior requires an administrator to place the same access controls (ACLs) on both URLs.

By junctioning a third-party server with the \(-i\) option, WebSEAL treats the URLs directed to that server as case-insensitive.

**Attention:** When using the \(-i\) option, object names must be lower case in order for WebSEAL to be able to find any ACLs or POPs attached to those objects. For more information, see “ACLs and POPs must attach to lower-case object names” on page 305

### Stateful junction support (\(-s\), \(-u\))

Most Web-enabled applications maintain a “state” for a sequence of HTTP requests from a client. This state is used, for example, to:

- Track a user’s progress through the fields in a data entry form generated by a CGI program
- Maintain a user’s context when performing a series of database inquiries
- Maintain a list of items in an online shopping cart application where a user randomly browses and selects items to purchase

Servers that run Web-enabled applications can be replicated in order to improve performance through load sharing. When the WebSEAL server provides a junction to these replicated back-end servers, it must ensure that all the requests contained within a client session are forwarded to the correct server, and not distributed among the replicated back-end servers according to the load balancing rules.

By default, Tivoli Access Manager balances back-end server load by distributing requests across all available replicated servers. Tivoli Access Manager uses a “least-busy” algorithm. This algorithm directs each new request to the server with the fewest connections already in progress.

The **create** command \(-s\) flag overrides this load balancing rule and creates a “stateful junction” that ensures a client’s requests are forwarded to the same server throughout an entire session. When the initial client request occurs, WebSEAL places a cookie on the client system that contains the UUID of the designated back-end server. When the client makes future requests to the same resource, the cookie’s UUID information ensures that the requests are consistently routed to the same back-end server.

The \(-s\) option is appropriate for a single front-end WebSEAL server with multiple back-end servers junctioned at the same junction point. Note that as soon as the initial junction is created as stateful, the **add** command is used without the \(-s\) option to junction the remaining replicated back-end servers to the same junction point.

If the scenario involves multiple front-end WebSEAL servers, all junctioned to the same back-end servers, you must use the \(-u\) option to correctly specify each
specifying back-end server UUIDs for stateful junctions (–u)

When a new junction is created to a back-end Web application server, WebSEAL normally generates a Unique Universal Identifier (UUID) to identify that back-end server. This UUID is used internally and also to maintain stateful junctions (create –s).

When the initial client request occurs, WebSEAL places a cookie on the client system that contains the UUID of the designated back-end server. When the client makes future requests to the same resource, the cookie’s UUID information ensures that the requests are consistently routed to the same back-end server.

Multiple front-end servers require a load balancing mechanism to distribute the load between the two servers. For example, an initial “state” could be established to a back-end server through WebSEAL server 1 using a specific UUID.

However, if a future request from the same client is routed through WebSEAL server 2 by the load balancing mechanism, the “state” will no longer exist, unless WebSEAL server 2 uses the same UUID to identify the same back-end server. Normally, this will not be the case.

The –u option allows you to supply the same UUID for a specific back-end server to each front-end WebSEAL server.

As an example, consider two replicated front-end WebSEAL servers, each with a stateful junction to two back-end servers. When you create the stateful junction between WebSEAL server 1 and back-end server 2, a unique UUID (UUID A) is generated to identify back-end server 2. However, when a stateful junction is created between WebSEAL server 2 and back-end server 2, a new and different UUID (UUID B) is generated to identify back-end server 2.
A "state" established between a client and back-end server 2, via WebSEAL server 1 will fail if a subsequent request from the client is routed through WebSEAL server 2.

Apply the following process for specifying a UUID during the creation of a junction:

1. Create a junction from WebSEAL server 1 to each back-end server. Use `create -s` and `add`.
2. List the UUID generated for each back-end server during Step 1. Use `show`.
3. Create a junction from WebSEAL server 2 to each back-end server and specify the UUIDs identified in Step 2. Use `create -s --u` and `add --u`.

In the following figure, back-end server 1 is known by both WebSEAL-1 and WebSEAL-2 as UUID 1. Back-end server 2 is known by both WebSEAL-1 and WebSEAL-2 as UUID 2.

**Example:**
In the following example,
- WebSEAL-1 instance is called WS1. Host name is cruz.
- WebSEAL-2 instance is called WS2. Host name is meow.
- Back-end server 1 is called APP1
• Back-end server 2 is called APP2

```
pdadmin> server task WS1-webseald-cruz create -t tcp -h APP1 -s /mnt
pdadmin> server task WS1-webseald-cruz add -h APP2 /mnt
pdadmin> server task WS1-webseald-cruz show /mnt
```

(This reveals UUID1 and UUID2)

```
pdadmin> server task WS2-webseald-meow create -t tcp -h APP1 -u UUID1 -s /mnt
pdadmin> server task WS2-webseald-meow add -h APP2 -u UUID2 /mnt
```

When a client establishes a stateful connection with back-end server 2, it receives a cookie containing UUID2. The above example now ensures that the client will always connect to back-end server 2, regardless of whether future requests are routed through WebSEAL-1 or WebSEAL-2.

### Junctioning to Windows file systems (–w)

WebSEAL performs security checks on client requests to junctioned back-end servers based on the file paths specified in the URL. A compromise in this security check can occur because Win32 file systems allow two different methods for accessing long file names.

The first method acknowledges the entire file name. For example:

```
abcdefg1jk1.txt
```

The second method recognizes the old 8.3 file name format for backward compatibility. For example:

```
abcdef“1.txt
```

When you create junctions in a Windows environment, it is important to restrict access control to one object representation only and not allow the possibility of “back doors” that bypass the security mechanism.

The –w option on a junction provides the following measures of protection:

• Prevents the use of the 8.3 file name format
  When the junction is configured with the –w option, a user cannot avoid an explicit ACL on a long file name by using the short (8.3) form of the file name. The server returns a “403 Forbidden” error on any short form file name entered.

• Disallows trailing dots in directory and file names
  If a file or directory contains trailing dots, a 403 ”Forbidden” error is returned.

• Enforces case-insensitivity by setting the –i option
  The –w option automatically invokes the –i option. This option specifies that WebSEAL treat URLs as case-insensitive when performing authorization checks on a request to a junctioned back-end server. After a successful ACL check, the original case of the URL is restored when the request is sent to the back-end server.

**Note:** If you require control over case-insensitivity only for file names, use only the –i option on the junction instead of the –w option.

### Example:

In a Windows environment, the file:

```
\Program Files\Company Inc\Release.Notes
```

can also be accessed via the following paths:

1. `\progra~1\compan~2\releas~3.not`
Example 1 illustrates how Windows can create an alias (for DOS compatibility) that contains no spaces in the file names and conforms to the 8.3 format. The -w option causes WebSEAL to reject this format for ACL checks.

Example 2 illustrates how Windows can include trailing extension dots. The -w option causes WebSEAL to reject this format for ACL checks.

Example 3 illustrates how Windows allows case-insensitivity on the file name. The -w option invokes the -i option to ensure a case-insensitive ACL check.

**ACLs and POPs must attach to lower-case object names**

When a junction is created with the -w or -i option, WebSEAL performs ACL and POP comparisons as case-insensitive. This means that the name of any object being evaluated for an ACL is placed into lower-case before WebSEAL checks it against the object list to which ACLs are attached.

As a result, protected objects with names that contain upper case letters are not found during the ACL or POP checks. If this occurs, the ACL or POP is not applied to the protected object, and the parent policy is applied instead.

To avoid the possible misapplication of policy in this configuration, you must create lower-case versions of the same names of the real protected objects to which you want to attach explicit ACLs or POPs.

**Specifying UTF-8 encoding for HTTP header data**

WebSEAL inserts information into HTTP headers for requests to the backend server. This information can include extended attributes or user data. In WebSEAL versions prior to 5.1, the headers were added to the request using raw local code page. In Version 5.1, the header data is transmitted in a configurable format.

By default, WebSEAL now adds information to HTTP headers using UTF-8 encoding. This prevents any potential data loss that could occur when converting to a non-UTF-8 code page. Also by default, this data is sent URI encoded. For backwards compatibility, the format of the header data can be configured to raw local code page. In addition, two other formats are supported: Raw UTF-8 and URI encoded local code page.

The -e option for creating junctions specifies the encoding of user name, groups, and other extended attributes which are sent within the HTTP header to the backend server. The encode option can take one of the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>utf8_uri</td>
<td>URI encoded UTF-8 data. All white space and non-ascii bytes are encoded %XY, where X and Y are hex values (0–F).</td>
</tr>
<tr>
<td>utf8_bin</td>
<td>Un-encoded UTF-8 data. This setting allows data to be transmitted without data loss, and the customer does not need to URI-decode the data. This setting should be used with caution, because it is not part of the HTTP specification.</td>
</tr>
<tr>
<td>lcp_uri</td>
<td>URI encoded local code page data</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td></td>
<td>Any UTF-8 characters which can not be converted to local code page will be converted to question marks (?). Use this option with caution and only in environments where local code page produces the desired strings.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lcp_bin</th>
<th>Un-encoded local code page data.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This mode was used by versions of WebSEAL prior to Version 5.1. Use of this mode enables migration from previous versions, and is used in upgrade environments. Use with caution, because data loss can potentially occur with this mode.</td>
</tr>
</tbody>
</table>

For more information on WebSEAL support for UTF-8 encoding, see "Multi-locale support with UTF-8" on page 44.
Technical notes for using WebSEAL junctions

- “Mounting multiple servers at the same junction” on page 307
- “Exceptions to enforcing permissions across junctions” on page 307
- “Certificate authentication across junctions” on page 307
- “Handling domain cookies” on page 308

Mounting multiple servers at the same junction
You can mount multiple replicated servers at the same junction point. There can be any number of servers mounted at the same point.

All servers mounted at one junction point must be replicas (mirrored Web spaces), and must use the same protocol—HTTP or HTTPS. Do not mount dissimilar servers on the same junction point.

From the primary Tivoli Access Manager server Web space, access pages belonging to the junctioned servers. You should be able to access these pages (subject to permissions, of course) and the pages should appear consistent. If a page cannot be found occasionally, or if it changes occasionally, it means that page was not replicated properly.

Check that the document exists and is identical in the document tree of both replicated servers.

Exceptions to enforcing permissions across junctions
Certain Tivoli Access Manager permissions are not enforceable across a junction. You cannot control, for example, the execution of a CGI script with the x permission, or a directory listing with the I permission. WebSEAL has no means of accurately determining whether or not a requested object on a back-end server is, for example, a CGI program file, a dynamic directory listing, or a regular HTTP object.

Access to objects across junctions, including CGI programs and directory listings, is controlled only through the r permission.

Certificate authentication across junctions
At installation, WebSEAL is configured with a non-default test certificate. The test certificate is designated as the active server-side certificate by the webseal-cert-keyfile-label parameter in the [ssl] stanza of the WebSEAL configuration file.

If a junctioned back-end application server requires WebSEAL to identify itself with a client-side certificate, you must first create, install, and label this certificate using the iKeyman utility. Then, configure the junction using the -K key-label option. See “Mutually authenticated SSL junctions” on page 282.

If the junction is not configured with -K, GSKit handles a request for mutual authentication by automatically sending the “default” certificate contained in the keyfile database. If this is not the required response, you must ensure that there are no certificates marked as ”default” (an asterisk mark) in the keyfile database (pdsrv.kdb).

In summary:
- Identify all required certificates by label name.
• Do not mark any certificate in the keyfile database as "default".
• Control the WebSEAL server-side certificate response with the
  webseal-cert-keyfile-label parameter.
• Control the WebSEAL client-side certificate response through the –K junction
  option.

Handling domain cookies

The allow-backend-domain-cookies parameter in the [session] stanza of the
WebSEAL configuration file allows you to control how WebSEAL handles domain
attributes in cookie headers.

When this parameter is set to "no" (default), WebSEAL performs "tail matching" to
determine if the domain (contained as an attribute in the cookie header) is valid. If
the domain in the cookie header is valid, the cookie is sent to the browser with the
domain attribute removed from the cookie header. When a browser receives a
cookie with no domain attribute, it can return the cookie only to the originating
server. If "tail matching" determines that the domain in the cookie header is not
valid, the cookie is not sent to the browser. The browser has no cookies to return.

[session]
allow-backend-domain-cookies = no

When this parameter is set to "yes", WebSEAL does not perform "tail matching"
and allows all cookies, regardless of the domain attribute value, to be sent to the
browser. The browser can return the cookies to the appropriate server or servers.

[session]
allow-backend-domain-cookies = yes

WebSEAL returns HTTP/1.1

HTTP/1.0 requests are sent to junctioned back-end servers only if those servers
return a status of 400 (Bad Request), return a status of 504 (HTTP version not
supported), or if the client browser specifies HTTP/1.0 in the request.

Otherwise, if the back-end server accepts HTTP/1.1, WebSEAL sends HTTP/1.1
requests.

However, even when WebSEAL sends an HTTP/1.0 request to a junctioned
back-end server (and the back-end server returns an HTTP/1.0 response),
WebSEAL always returns an HTTP/1.1 response to the client browser.

Junctioned application with Web Portal Manager

Problem: Web Portal Manager sends absolute or server-relative URLs in its
Javascript. These addresses are not resolved successfully by the browser and
require junction cookie information to complete the path name.

Solution: If an application server with Web Portal Manager is junctioned to
WebSEAL, you must use the –j option when creating this junction. The junction
cookie provided by the –j option allows the browser (client) to successfully issue
commands to Web Portal Manager.

In addition to using the –j option, you must also use the –c iv_user,iv_creds
option.
Using `query_contents` with third-party servers

If you want to protect the resources of the third-party application Web space using the Tivoli Access Manager security service, you must provide WebSEAL with information about the contents of the third-party Web space.

A CGI program called `query_contents` provides this information. The `query_contents` program searches the third-party Web space contents and provides this inventory information to the Web Portal Manager on WebSEAL. The program comes with the WebSEAL installation, but must be manually installed on the third-party server. There are different program file types available, depending on whether the third-party server is running UNIX or Windows.

The Object Space manager of the Web Portal Manager automatically runs `query_contents` any time the portion of the Protected Object Space representing the junction is expanded in the Object Space management panel. Now that the Web Portal Manager knows about the contents of the third-party application space, you can display this information and apply policy templates to appropriate objects.

Installing `query_contents` components

Installing `query_contents` is typically very easy. Installation involves copying one or two files from the Tivoli Access Manager server to the third-party server and editing a configuration file.

The following Tivoli Access Manager directory contains a template of the program:

UNIX:

```
install-path/www/lib/query_contents
```

Windows:

```
install-path\www\lib\query_contents
```

The contents of the directory includes:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>query_contents.exe</code></td>
<td>Main executable program for Win32 systems. Should be installed in the <code>cgi-bin</code> directory of the third-party Web server.</td>
</tr>
<tr>
<td><code>query_contents.sh</code></td>
<td>Main executable program for UNIX systems. Should be installed in the <code>cgi-bin</code> directory of the third-party Web server.</td>
</tr>
<tr>
<td><code>query_contents.c</code></td>
<td>Source code. The source is provided in case you need to modify the behavior of <code>query_contents</code>. In most cases, this will not be necessary.</td>
</tr>
<tr>
<td><code>query_contents.html</code></td>
<td>Help file in HTML format.</td>
</tr>
<tr>
<td><code>query_contents.cfg</code></td>
<td>A sample configuration file that identifies the document root for the Web server.</td>
</tr>
</tbody>
</table>

Installing `query_contents` on third-party UNIX servers

Locate the shell script named `query_contents.sh` in the following directory:

```
install-path/www/lib/query_contents
```
1. Copy query_contents.sh into a functioning /cgi-bin directory on the third-party Web server.
2. Remove the .sh extension from the file name.
3. Manually edit the script file to correctly specify the doc root directory.
4. Set the UNIX execute bit for the administration account of the Web server.

**Installing query_contents on third-party Win32 servers**

**Special junction option for Windows:**

When you require, and install, the query_contents program on a back-end junctioned Windows Web application server, you must use the --q option when creating the junction to that server.

The reason for this requirement is that by default, WebSEAL looks for the program, query_contents, in the cgi-bin directory:

```
/cgi-bin/query_contents
```

If you change the name of the query_contents program or change its directory location, you must use the --q location option when creating the junction. The location argument specifies the new location and name of the program.

The name of the query_contents program used on the Windows platform is query_contents.exe. The presence of the .exe extension makes the program name different. Therefore, WebSEAL cannot find the default program name. You must use the --q location option and argument to tell WebSEAL where to find the file. For example:

```
create -t tcp -h host-name ... -q /cgi-bin/query_contents.exe /junction-name
```

The --q option is not required for a UNIX server because the query_contents program name and location matches the default condition.

**Procedure:**

Locate the executable program named query_contents.exe and the configuration file named query_contents.cfg in the following directory:

Windows: `install-path\www\lib\query_contents`

1. Ensure that the third-party Web server has a CGI directory correctly configured.
2. For testing purposes, ensure that a valid document exists in the document root of the third-party Web server.
3. Copy query_contents.exe into the CGI directory of the third-party Web server.

Default values for this directory are shown in the table below:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Windows Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 95 and 98</td>
<td>c:\windows</td>
</tr>
<tr>
<td>Windows NT 4.x and 2000</td>
<td>c:\winnt</td>
</tr>
</tbody>
</table>

5. Edit the query_contents.cfg file to correctly specify the document root directory for the third-party Web server.
The file currently contains sample entries for the Microsoft Internet Information Server and Netscape FastTrack servers. The lines in this file that start with a semi-colon (;) are comments and are ignored by the \texttt{query\_contents} program.

6. Create a junction to the back-end Windows server and use the \texttt{–q} option to specify \texttt{query\_contents.exe} as the correct file. For example:

\texttt{padmin> server task web1-webseald-cruz create -t tcp -h host-name ... \-q /cgi-bin/query\_contents.exe /junction-name}

\textbf{Testing the configuration}

1. From an MS-DOS prompt on the Win32 machine, execute the \texttt{query\_contents} program from the CGI directory as follows:

\texttt{MSDOS> query\_contents dirlist=/}

You should see something similar to the following output:

\begin{verbatim}
100 index.html
cgi-bin//
pics//
\end{verbatim}

The number 100 is a return status that indicates success. It is most important to see at least the number 100 as the first (and perhaps only) value.

If you see an error code instead, then the configuration file is not in the correct place, or does not contain a valid document root entry. Check the configuration of the \texttt{query\_contents.cfg} file and make sure that the document root exists.

2. From a browser, enter the following URL

\texttt{http://win32-machine-name/cgi-bin/query\_contents.exe?dirlist=/}

This should return the same result as the preceding step. If it does not return this result, the CGI configuration of your Web server is not correct. See the server’s documentation to correct the problem.

\textbf{Customizing query\_contents}

The job of \texttt{query\_contents} is to return the contents of directories included in a URL request.

For example, to get the contents of the root directory of a server’s Web space, the browser runs \texttt{query\_contents} on a URL such as:

\texttt{http://third-party-server/cgi-bin/query\_contents?dirlist=/}

The \texttt{query\_contents} script performs the following actions:

1. Reads \texttt{SERVER\_SOFTWARE}, a standard CGI environment variable, to determine the server type.

Based on the Web server type, the variable \texttt{DOCROOTDIR} is set to a typical document root location.

2. Reads the environment variable \texttt{QUERY\_STRING} from the requested URL to obtain the requested \texttt{operation} and get the \texttt{object path}.

The operation value is stored in the variable \texttt{OPERATION}, and the object path in \texttt{OBJPATH}. In the above example, the \texttt{OPERATION} is \texttt{dirlist} and \texttt{OBJPATH} is \\

3. Performs a directory listing (\texttt{ls}) on the object path and places the results on standard output, for use by the Tivoli Access Manager server. Entries that indicate subdirectories have a double slash (\texttt{//}) appended to them.
Typical output looks like:

```
100
index.html
cgi-bin//
pics//
```

The number 100 is a return status that indicates success.

**Customizing the doc root directory**

**UNIX:**

To customize `query_contents.sh` for your UNIX server, you might need to modify the document root directory setting.

If `query_contents` returns an error status (a number other than 100) and lists no files, examine the script and modify the `$DOCROOTDIR` variable, if needed, to match your server’s configuration.

If the document root directory is specified correctly and the script still fails, the `cgi-bin` location specification might be incorrect. Examine the `$FULLOBJPATH` variable and modify the value assigned to it to reflect the correct `cgi-bin` location.

**Windows:**

To customize `query_contents.exe` for your Windows server, modify the `query_contents.cfg` file.

**Additional functionality**

The source code for the `query_contents` program (`query_contents.c`) is distributed royalty-free with Tivoli Access Manager.

Additional functionality can be added to this program to support special features of some third-party Web servers. These features include:

- Directory mapping — where a sub-directory not below the document root is mapped into the Web space.
- Generation of a Web space that is not file system based.

This might be the case for a database-hosted Web server.

**Securing query_contents**

The `query_contents` CGI program is used by Tivoli Access Manager to display junctioned Web server object spaces in the Web Portal Manager. It is very important to secure this file to prevent unauthorized users from running it.

You must set a security policy that allows only the policy server (pdmgrd) identity to have access to the `query_contents` program. The following example ACL (`query_contents_acl`) meets this criteria:

```
group ivmgrd-servers T1
user sec_master dbxTrlcam
```

Use the `pdadmin` utility to attach this ACL to the `query_contents.sh` (UNIX) or `query_contents.exe` (Windows) object on the junctioned servers. For example (UNIX):

```
pdadmin> acl attach /WebSEAL/host/junction-name/query_contents.sh \     query_contents_acl
```
Troubleshooting

- Problem:
  query_contents does not work with -b ignore option on junctions.

Explanation:
If WebSEAL and the Microsoft IIS Web plug-in are both configured and both use basic authentication, then a junction created in WebSEAL to IIS with the -b ignore option specified might not be treated correctly in Web Portal Manager or by the pdadmin command. This problem manifests itself in Web Portal Manager by the object space branch not being expanded for the WebSEAL junction because the query_contents.exe command does not return any information.

Similarly, the pdadmin object list /WebSEAL/host_name/junction_name command might not display information on the junction.

Workaround:
To avoid this problem, temporarily turn off basic authentication on the backend server when the administrators need to apply URI-level access control. Alternately, ensure that the query_contents.exe command is run from a directory that does not require basic authentication.
Chapter 11. Single sign-on solutions across junctions

When WebSEAL is implemented as a proxy server to provide protection to a secure domain, there is often a requirement to provide solutions for single sign-on to Web resources located on junctioned back-end servers. This chapter discusses single sign-on solutions for the junctioned Web space of a WebSEAL proxy configuration. Examples include specially configured junctions, Global Sign-on, and LTPA.

Topic Index:
- “Configuring BA headers for single sign-on solutions” on page 316
- “Using global sign-on (GSO)” on page 320
- “Configuring single sign-on to IBM WebSphere (LTPA)” on page 324
- “Configuring single sign-on forms authentication” on page 326
Configuring BA headers for single sign-on solutions

This section discusses the possible solutions for creating single sign-on configurations across WebSEAL junctions using the –b options.

- “Single sign-on (SSO) concepts” on page 316
- “Supplying client identity in BA headers” on page 316
- “Supplying client identity and generic password” on page 316
- “Forwarding original client BA header information” on page 318
- “Removing client BA header information” on page 318
- “Supplying user names and passwords from GSO” on page 318

Single sign-on (SSO) concepts

When a protected resource is located on a back-end Web application server, a client requesting that resource can be required to perform multiple logins — one for the WebSEAL server and one for the back-end server. Each login likely requires different login identities.

The problem of administering and maintaining multiple login identities can often be solved with a single sign-on (SSO) mechanism. A single sign-on solution allows the user to access a resource, regardless of the resource’s location, using only one initial login. Any further login requirements from back-end servers are handled transparently to the user.

Supplying client identity in BA headers

You can configure WebSEAL junctions to supply the back-end server with original or modified client identity information. The set of –b options allows you to supply specific client identity information in HTTP Basic Authentication (BA) headers.

As the administrator, you must analyze your network architecture and security requirements, and determine answers to the following questions:

1. Is authentication information required by the back-end server?
   (WebSEAL uses the HTTP Basic Authentication header to convey authentication information.)

2. If authentication information is required by the back-end server, where does this information come from?
   (What information does WebSEAL place in the HTTP header?)

3. Does the connection between WebSEAL and the back-end server need to be secure?
   (TCP or SSL junction?)

After the initial authentication between the client and WebSEAL, WebSEAL can build a new Basic Authentication header. The request uses this new header as it continues across the junction to the back-end server. You use the –b options to dictate what specific authentication information is supplied in this new header.

Supplying client identity and generic password

–b supply

The –b supply option instructs WebSEAL to supply the authenticated Tivoli Access Manager user name (client’s original identity) with a static, generic (“dummy”) password. The original client password is not used in this scenario.
A generic password eliminates password administration and supports the application on a per-user basis. The “dummy” password is set in `basicauth-dummy-passwd` parameter of the WebSEAL configuration file:

```
[junction]
basicauth-dummy-passwd = password
```

This scenario assumes the back-end server requires authentication from a Tivoli Access Manager identity. By mapping a client user to a known Tivoli Access Manager user, WebSEAL manages authentication for the back-end server and provides a simple domain-wide single sign-on solution.

The following conditions exist for this solution:

- WebSEAL is configured to supply the back-end server with the user name contained in the original client request plus a generic (“dummy”) password.
- The “dummy” password is configured in the WebSEAL configuration file.
- The back-end server registry must recognize the Tivoli Access Manager identity supplied in the HTTP BA header.
- Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. An SSL junction is highly recommended.

![Diagram](image)

*Figure 18. BA Header contains identity and “dummy” password*

**Limitations**

The same Tivoli Access Manager “dummy” password is used for all requests; all users have the same password in the back-end server registry. The use of the common “dummy” password offers no basis for the application server to prove the legitimacy of the client logging in with that user name.

If clients always go through WebSEAL to access the back-end server, this solution does not present any security problems. However, it is important to physically secure the back-end server from other possible means of access.

Because this scenario has no password-level security, the back-end server must implicitly trust WebSEAL to verify the legitimacy of the client.

The back-end server registry must also recognize the Tivoli Access Manager identity in order to accept it.
Forwarding original client BA header information

–b ignore

The –b ignore option instructs WebSEAL to pass the original client Basic Authentication (BA) header straight to the back-end server without interference. WebSEAL can be configured to authenticate this BA client information or ignore the BA header supplied by the client and forward the header, without modification, to the back-end server.

**Note:** This is not a true single sign-on mechanism, but rather a direct login to the third-party server, transparent to WebSEAL.

The following conditions exist for this solution:

• The back-end server requires client identity information via BA
  The back-end server will send a Basic Authentication challenge back to the client. The client responds with user name and password information which the WebSEAL server passes through without modification.

• The back-end server maintains its own client-supplied passwords

• WebSEAL is configured to supply the back-end server with the user name and password contained in the original client request.

• Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. An SSL junction is highly recommended.

Removing client BA header information

–b filter

The –b filter option instructs WebSEAL to remove all Basic Authentication header information from any client requests before forwarding the requests to the back-end server. In this scenario, WebSEAL becomes the single security provider.

The following conditions exist for this solution:

• Basic Authentication is configured between the client and WebSEAL

• The back-end server does not require Basic Authentication

• The back-end server can be accessed only through WebSEAL

• WebSEAL handles authentication on behalf of the back-end server

If you need to supply the back-end server with some client information, you can combine this option with the –c option to insert Tivoli Access Manager client identity information into HTTP header fields. See “Supplying client identity in HTTP headers (–c)” on page 298.

Supplying user names and passwords from GSO

–b gso

The –b gso option instructs WebSEAL to supply the back-end server with authentication information (user name and password) obtained from a server that is set up to handle global sign-on (GSO).

The following conditions exist for this solution:

• The back-end server applications require different user names and passwords that are not contained in the WebSEAL registry.
• Security is important for both WebSEAL and the back-end server.

Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. An SSL junction is highly recommended.

This mechanism is fully described in "Using global sign-on (GSO)" on page 320.
Using global sign-on (GSO)

Tivoli Access Manager supports a flexible single sign-on solution that features the ability to provide alternative user names and passwords to the back-end Web application server.

Global Sign-on grants users access to the computing resources they are authorized to use — through a single login. Designed for large enterprises consisting of multiple systems and applications within heterogeneous, distributed computing environments, GSO eliminates the need for end users to manage multiple user names and passwords.

The integration is achieved by creating "aware" junctions between WebSEAL and back-end Web servers. GSO resources and GSO resource groups must first be created using the Web Portal Manager or the pdadmin utility.

When WebSEAL receives a request for a resource located on the junctioned server, WebSEAL asks the user registry server for the appropriate authentication information. The user registry server contains a database of mappings—for each registered user—that provides alternative user names and passwords for specific resources and applications.

The following figure illustrates how the GSO mechanism is used to retrieve user names and passwords for back-end application resources.

1. The client authenticates to WebSEAL with a request for access to an application resource on an back-end server. A Tivoli Access Manager identity is obtained.

   **Note:** The single sign-on process is independent of the initial authentication method.

2. WebSEAL passes the Tivoli Access Manager identity to the user registry server.
3. The registry returns a user name and password appropriate for the user and the requested application resource.
4. WebSEAL inserts the user name and password information in the HTTP Basic Authentication header of the request that is sent across the junction to the back-end server.
Mapping the authentication information

The following example illustrates how the user registry provides authentication information to WebSEAL. If user Michael wants to run the travel-app application resource (refer to Figure 19), WebSEAL asks the user registry server for Michael’s authentication information.

The user registry server maintains a complete database of authentication information in the form of mappings of resources to specific authentication information. The authentication information is a user name / password combination known as a resource credential. Resource credentials can be created only for registered users.

The registry contains a database for Michael that maps the resource travel-app to a specific resource credential.

The following table illustrates the structure of the GSO resource credential database:

<table>
<thead>
<tr>
<th>Michael</th>
<th>Paul</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource: travel-app</td>
<td></td>
</tr>
<tr>
<td>username=mike</td>
<td></td>
</tr>
<tr>
<td>password=123</td>
<td></td>
</tr>
<tr>
<td>resource: payroll-app</td>
<td></td>
</tr>
<tr>
<td>username=powell</td>
<td></td>
</tr>
<tr>
<td>password=456</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resource: travel-app</td>
</tr>
<tr>
<td>username=bundy</td>
<td></td>
</tr>
<tr>
<td>password=abc</td>
<td></td>
</tr>
<tr>
<td>resource: payroll-app</td>
<td></td>
</tr>
<tr>
<td>username=jensen</td>
<td></td>
</tr>
<tr>
<td>password=xyz</td>
<td></td>
</tr>
</tbody>
</table>

In this example, the registry returns user name "mike" and password "123" to WebSEAL. WebSEAL uses this information when it constructs the Basic Authentication header in the request sent across the junction to the back-end server.
Configuring a GSO-enabled WebSEAL junction

Support for GSO is configured at the junction between WebSEAL and a back-end server.

To create a junction that enables GSO, use the `create` command with the `-b gso` option. The following example illustrates the syntax for the `create` command:

```bash
create -t tcp -h host-name -b gso -T resource jct-point
```

Options for setting up GSO junctions are listed below:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b gso</code></td>
<td>Specifies that GSO should provide authentication information for all requests crossing this junction.</td>
</tr>
<tr>
<td><code>-T resource/resource-group</code></td>
<td>Specifies the GSO resource or resource group. The resource name used as the argument to this option must exactly match the resource name as listed in the GSO database. Required for gso junctions.</td>
</tr>
</tbody>
</table>

A junction used in a WebSEAL/GSO solution can be made secure through SSL by additionally applying the `-t ssl` option when creating the junction.

Always use SSL junctions with GSO to ensure encryption of credentials and all data.

**Examples of GSO-enabled WebSEAL junctions**

Junction the application resource `travel-app` on host `sales_svr` to junction point `/sales`:

```bash
create -t tcp -b gso -T travel-app -h sales_svr /sales
```

Junction the application resource `payroll-app` on host `adm_svr` to junction point `/admin` and make the junction secure with SSL:

```bash
create -t ssl -b gso -T payroll-app -h adm_svr /admin
```

**Note:** In the above example, the `-t ssl` option dictates a default port of 443.

Configuring the GSO cache

The Global Sign-on (GSO) cache functionality allows you to improve the performance of GSO junctions in a high load environment. By default, the GSO cache is disabled. Without the enhancement of the cache, a call to the user registry server is required for each retrieval of GSO target information (GSO user name and GSO password).

Parameters for configuring the GSO cache are located in the `[gso-cache]` stanza of the WebSEAL configuration file. You must first enable the cache. The remaining parameters configure the cache size and the timeout values for cache entries. Larger lifetime and inactivity timeout values improve performance, but increase the risk of information being exposed in the WebSEAL memory. Do not enable the GSO cache if GSO junctions are not used in your network solution.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gso-cache-enabled</td>
<td>Enable and disable the GSO cache functionality. Values are yes or no. Default is no.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gso-cache-size</td>
<td>Sets the maximum number of entries allowed in the cache hash table. Set this value to approximate the peak number of concurrent user sessions that access an application across a GSO junction. A high value uses more memory but results in faster information access. Each cache entry consumes approximately 50 bytes.</td>
</tr>
<tr>
<td>gso-cache-entry-lifetime</td>
<td>Maximum time (in seconds) any cache entry can remain in the cache, regardless of activity. After a cache entry expires, the next request by that same user requires a new call to the user registry server. Default value is 900 seconds.</td>
</tr>
<tr>
<td>gso-cache-entry-idle-timeout</td>
<td>Maximum time (in seconds) an inactive cache entry can remain in the cache. Default value is 120 seconds.</td>
</tr>
</tbody>
</table>

For more information, see “Global Sign-On cache” on page 467.
Configuring single sign-on to IBM WebSphere (LTPA)

WebSEAL can provide authentication and authorization services and protection to an IBM WebSphere environment. When WebSEAL is positioned as a protective front-end to WebSphere, accessing clients are faced with two potential login points. Therefore, WebSEAL supports a single sign-on solution to one or more IBM WebSphere servers across WebSEAL junctions.

WebSphere provides the cookie-based lightweight third-party authentication mechanism (LTPA). You can configure WebSEAL junctions to support LTPA and provide a single sign-on solution for clients.

When a user makes a request for a WebSphere resource, the user must first authenticate to WebSEAL. After successful authentication, WebSEAL generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token for WebSphere, contains the user identity, key and token data, buffer length, and expiration information. This information is encrypted using a password-protected secret key shared between WebSEAL and the WebSphere server.

WebSEAL inserts the cookie in the HTTP header of the request that is sent across the junction to WebSphere. The back-end WebSphere server receives the request, decrypts the cookie, and authenticates the user based on the identity information supplied in the cookie.

To improve performance, WebSEAL can store the LTPA cookie in a cache and use the cached LTPA cookie for subsequent requests during the same user session. You can configure lifetime timeout and idle (inactivity) timeout values for the cached cookie.

**Configuring an LTPA junction**

Single sign-on to WebSphere via an LTPA cookie requires the following configuration tasks:

1. Enable the LTPA mechanism.
2. Provide the location of the key file used to encrypt the identity information.
3. Provide the password to this key file.

These three configuration requirements are specified in three additional options to the junction `create` command:

- The `–A` option enables LTPA cookies.
- The `–F "keyfile"` option and argument specifies the full path name location (on the WebSEAL server) of the key file used to encrypt the identity information contained in the cookie. The shared key is originally created on the WebSphere server and copied securely to the WebSEAL server. Refer to the appropriate WebSphere documentation for specific details regarding this task.
- The `–Z "keyfile-password"` specifies the password required to open the key file.
  The password appears as encrypted text in the junction XML file.

Use these options in addition to other required junction options when you create the junction between WebSEAL and the back-end WebSphere server. For example:

```bash
create ... -A -F "/abc/xyz/key.file" -Z "abcdefg" ...
```
Configuring the LTPA cache

The creation, encryption, and decryption of LTPA cookies introduces processing overhead. The LTPA cache functionality allows you to improve the performance of LTPA junctions in a high load environment. Without the enhancement of the cache, a new LTPA cookie is created and encrypted for each subsequent user request. By default, the LTPA cache is enabled.

Parameters for configuring the LTPA cache are located in the [ltpa-cache] stanza of the WebSEAL configuration file. Parameters specify the cache size and the timeout values for cache entries. Larger lifetime and inactivity timeout values improve performance, but increase the risk of information being exposed in the WebSEAL memory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltpa-cache-enabled</td>
<td>Enable and disable the LTPA cache functionality. Values include &quot;yes&quot; and &quot;no&quot;. Default value is &quot;yes&quot;.</td>
</tr>
<tr>
<td>ltpa-cache-size</td>
<td>Sets the maximum number of entries allowed in the cache hash table. Set this value to approximate the peak number of concurrent user sessions that access an application across an LTPA junction. A high value uses more memory but results in faster information access. Each cache entry consumes approximately 50 bytes. Default value is 4096 entries.</td>
</tr>
<tr>
<td>ltpa-cache-entry-lifetime</td>
<td>Maximum time (in seconds) any cache entry can remain in the cache, regardless of activity. After a cache entry expires, the next request by that same user requires the creation of a new LTPA cookie. Default value is 3600 seconds</td>
</tr>
<tr>
<td>ltpa-cache-entry-idle-timeout</td>
<td>Maximum time (in seconds) an inactive cache entry can remain in the cache. Default value is 600 seconds.</td>
</tr>
</tbody>
</table>

For more information, see “Lightweight Third Party Authentication cache” on page 469.

Technical notes for LTPA single sign-on

- The key file contains information about a specific WebSphere server. An LTPA junction is specific to one WebSphere server. If you add more than one server to the same junction point, all servers will share the same key file.
- For single sign-on to succeed, WebSEAL and the WebSphere server must share the same registry information.
- The WebSphere server is responsible for setting up LTPA and the creation of the shared secret key. The WebSEAL participation involves the junction and cache configurations.
Configuring single sign-on forms authentication

Single sign-on forms authentication allows WebSEAL to transparently log an authenticated Tivoli Access Manager user in to a back-end junctioned application server that requires authentication via an HTML form.

Background and goals

Single sign-on forms authentication supports existing applications that use HTML forms for authentication and cannot be modified to directly trust the authentication performed by WebSEAL. Enabling single sign-on forms authentication produces the following results:

- WebSEAL interrupts the authentication process initiated by the back-end application
- WebSEAL supplies data required by the login form and submits the login form on behalf of the user.
- WebSEAL saves and restores all cookies and headers
- The user is unaware that a second login is taking place.
- The back-end application is unaware that the login form is not coming directly from the user.

WebSEAL must be configured to:

- Recognize and intercept the login form
- Fill in the appropriate authentication data

The administrator enables forms single sign-on by:

- Creating a configuration file to specify how the login form is to be recognized, completed, and processed
- Enable forms single sign-on by configuring the appropriate junction with the –S option (which specifies the location of the configuration file)

Forms single sign-on process flow

The following scenario assumes that WebSEAL has already authenticated the user.
1. Client browser requests the page:
   https://webseal/formsso/content.html
2. WebSEAL passes the request to the junction.
3. Because the back-end application requires the user to authenticate, a redirect to the application’s login page (login.html) is sent back across the junction.
4. WebSEAL passes the redirect to the browser.
5. The browser follows the redirect and requests:
   https://webseal/formsso/login.html

   **Note:** Everything to this point in the process flow is standard WebSEAL functionality.
6. WebSEAL has been configured for forms single sign-on (–S option on the junction). WebSEAL recognizes the request as a request for a login page, based on information contained in the forms SSO configuration file. The request is passed to the junction. WebSEAL saves all cookies sent by the browser for use in step 8.
7. The application returns the login page and perhaps application-specific cookies.

   WebSEAL parses the HTML returned to identify the login form. When WebSEAL finds an HTML form in the document, it compares the action URI in the form to the value of the `login-form-action` parameter in the custom configuration file. If there is a match, WebSEAL uses the form found. Otherwise, WebSEAL keeps searching for other forms. If no form in the page matches the action URI pattern from the configuration file, then WebSEAL terminates forms single sign-on processing and returns an error to the browser.
WebSEAL parses the HTML page to identify the request method, the action URI, and any other input fields in the form, and saves them for use in step 8.
8. WebSEAL generates the authentication request (completes the login form) and sends it to the back-end application.
9. The application authenticates the user using the authentication data supplied by WebSEAL in the form. The application returns a redirect to content.html.
10. WebSEAL combines any cookies saved from the responses at step 7 and step 9, and returns these cookies with the redirect to the browser.

**Note:** This completes the forms SSO-specific functionality.
11. The browser follows the redirect and requests:
    https://webseal/formsso/content.html
12. WebSEAL passes the request to the back-end application across the junction.

During this process, the browser makes three requests to WebSEAL. From the user’s perspective, only a single request for https://webseal/formsso/content.html is made. The other requests occur automatically through HTTP redirects.

**Requirements for application support**

Single sign-on for forms authentication is supported on applications that meet the following requirements:

- The login page or pages for the application must be uniquely identifiable via a single regular expression or several regular expressions.
- The login page can include more than one HTML form. However, the login form must be identified by applying a regular expression to the action URIs of each of the login forms, or the login form must be the first form in the login page. Note that when using the "action" attribute to identify the login form, the "action" attribute has not passed through WebSEAL's HTML filtering. The regular expression should match the action URI prior to being filtered.
- Client-side scripting may be used to validate input data, but it must not modify the input data (such as using Javascript to set cookies in the user's browser).
- Login data is submitted at only one point in the authentication process.
- The junction where the authentication request is directed must be the same junction where the login page is returned.

**Creating the configuration file for forms single sign-on**

The forms single sign-on configuration file is custom-created by the administrator and saved in any location. The -S option on the junction enables the forms single sign-on functionality and specifies the location of the configuration file. See "Enabling forms single sign-on" on page 332. A sample configuration file (containing commented instructions) is provided with the WebSEAL installation and is located in the following directory:

**UNIX:**

/opt/pdweb/etc/fsso.conf.template

**Windows:**

C:\Program Files\Tivoli\PDWeb\etc\fsso.conf.template

The configuration file must begin with the [forms-sso-login-pages] stanza and has the following format
The [forms-sso-login-pages] stanza
The forms single sign-on configuration file must always begin with the [forms-sso-login-pages] stanza. The stanza contains one or more login-page-stanza entries that point to other custom-named stanzas that contain configuration information for the login pages found on the back-end application server.

The ability to support multiple login pages on a single junction is important because a single back-end server might host several applications that each use a different authentication method.

For example:
[forms-sso-login-pages]
login-page-stanza = loginpage1
login-page-stanza = loginpage2

The custom login page stanza
Each custom login page stanza is used to intercept a particular URL pattern. The stanza can contain the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login-page</td>
<td>This parameter specifies a pattern, using a regular expression, that uniquely identifies requests for an application’s login page. WebSEAL intercepts these pages and begins the forms single sign-on process. The regular expression is compared against the request URI and is relative to (and not including) the junction point where the server is mounted.</td>
</tr>
<tr>
<td>login-form-action</td>
<td>This parameter specifies a pattern, using a regular expression, that identifies which form contained in the intercepted page is the application’s login form. If there is only a single form in the page, or if the login form is the first form in the document, then the expression can be &quot;*&quot;. Otherwise, the regular expression should match the &quot;action&quot; attribute of the login form.</td>
</tr>
<tr>
<td>gso-resource</td>
<td>This parameter specifies the GSO resource to use when retrieving the GSO user name and password from a GSO database. Leave this parameter blank if GSO is not used to store a GSO user name and password.</td>
</tr>
<tr>
<td>argument-stanza</td>
<td>This parameter points to another custom stanza that lists the fields and data required for completing the login form.</td>
</tr>
</tbody>
</table>

For example:
[loginpage1]
login-page = /cgi-bin/getloginpage*
login-form-action = *
gso-resource =
argument-stanza = form1-data

About the login-page parameter:

The value of the login-page parameter is a regular expression that WebSEAL uses to determine if an incoming request is actually a request for a login page. If this is the case, WebSEAL intercepts this request and begins the forms single sign-on processing.

Only one login-page parameter is allowed in each custom login page stanza. You must create an additional custom login page stanza for each additional login-page parameter.

The login-page regular expression is compared against the request URI, relative to the junction. In the following example, the URI of a request to a WebSEAL server called "websealA" for a resource on a junction called "junctionX" might appear as follows:

The part of this URL that is compared to the login-page regular expression is:
/auth/login.html

About the login-form-action parameter:

The login-form-action parameter is used to identify the login form on the intercepted page. Only one login-form-action parameter is allowed in each stanza.

The value of the login-form-action parameter is a regular expression that is compared against the contents of the "action" attribute of the HTML "form" tag. The "action" attribute is a URI expressed as a relative, server-relative, or absolute path. The login-form-action parameter must match this path as it comes from the back-end server - even if it would normally be modified by WebSEAL before being forwarded to the client.

If multiple "action" attributes on the page match the regular expression, only the first match is accepted as the login form.

If the regular expression does not match any form on the page, an error is returned to the browser reporting that the form could not be found.

You can set login-form-action = * as a simple way to match the login form when the page includes only one login form.

Using regular expressions

The following table lists the special characters allowed in regular expressions used in the forms single sign-on configuration file.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches zero or more characters</td>
</tr>
<tr>
<td>?</td>
<td>Matches any one character</td>
</tr>
<tr>
<td>\</td>
<td>Escape character (for example, ? matches ?)</td>
</tr>
<tr>
<td>[acd]</td>
<td>Matches character a, c, or d (case-sensitive)</td>
</tr>
<tr>
<td>[^acd]</td>
<td>Matches any character except a, c, or d (case-sensitive)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>[a-z]</td>
<td>Matches any character between a and z (lowercase letter)</td>
</tr>
<tr>
<td>[^0-9]</td>
<td>Matches any character not between 0 and 9 (not a number)</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>Matches any character between a and z (lowercase) or A and Z (uppercase)</td>
</tr>
</tbody>
</table>

In most cases, special characters are not required because the login page request is a single identifiable URI. In some cases, you can use the "*" at the end of the expression so that any query data at the end of the URI does not prevent the login page from being matched.

**The argument stanza**

The custom argument stanza contains one or more entries in the following form:

```
name> = method>:value>
```

**name**

The value of the name parameter is set to equal the value of the "name" attribute of the HTML "input" tag. For example:

```html
<input name=uid type=text>Username</input>
```

This parameter can also use the value of the "name" attribute of the HTML "select" or "textarea" tags.

**method: value**

This parameter combination retrieves the authentication data required by the form. The authentication data can include:

- Literal string data
  ```
  string:text
  ```
  The input used is the text string.

- GSO user name and password
  ```
  gso:username
  gso:password
  ```
  The input is the current user's GSO username and password (from the target specified in the custom login page stanza).

- Value of an attribute in the user's credential
  ```
  cred:cred-ext-attr-name
  ```
  By default, the credential includes information such as the user's Tivoli Access Manager user name and DN. To use the user's Tivoli Access Manager user name as the input value, specify the value as:
  ```
  cred:azn_cred_principal_name
  ```
  The user's DN may be accessed as:
  ```
  cred:azn_cred_authzn_id
  ```
  Custom credential attributes (added via the tag/value mechanism) can also be used.
It is not necessary to specify hidden input fields in this stanza. These fields are automatically retrieved from the HTML form and submitted with the authentication request.

For example:

[form1-data]
uid = string:brian

### Enabling forms single sign-on

After completing the custom forms single sign-on configuration file and locating the file in an appropriate directory, you must configure the appropriate junction to support forms single sign-on. Use the –S junction option with the `pdadmin create` command:

```
-S config-file-path
```

The `config-file-path` argument specifies the location of the custom forms single sign-on configuration file. The –S junction option enables the forms single sign-on functionality on the junction. For example (entered as one line):

**UNIX:**

```
pdadmin> server task web1-webseald-cruz -t tcp -h websvrA \
-S /opt/pdweb/fsso/fsso.conf /jctX
```

**Windows:**

```
pdadmin> server task web1-webseald-cruz -t tcp -h websvrA \
-S C:/Program Files/Tivoli/PDWeb/fsso/fsso.conf /jctX
```

**Note:** In a Windows environment, you must use forward slashes (/) in the `fsso.conf` path name, rather than the conventional back slashes (\).

The configuration file is read when the junction is created and each time WebSEAL is started. Errors in the configuration file can cause WebSEAL to fail at start-up.

### Example configuration file for IBM HelpNow

The IBM HelpNow site invokes its own forms-based login and is therefore an example of how a forms single sign-on solution can provide seamless access to the site for its enrolled users.

This section contains:

- A form section, similar to the form sent on the HTML login page returned by the HelpNow application
- The custom forms single sign-on configuration file used to process this form

**The form found in the intercepted HTML page:**

```html
<form name="confirm" method="post" action="../files/wcls_hnb_welcomePage2.cgi">
  <p>
    Employee Serial Number:
    <input name="data" size="10" maxlength="6">
  </p>
  <p>
    Country Name:
    <select name="Cntselect" size="1">
      <OPTION value="notselected" selected>Select Country</OPTION>
      <OPTION value=675>United Arab Emirates - IBM</OPTION>
      <OPTION value=866>United Kingdom</OPTION>
      <OPTION value=897>United States</OPTION>
      <OPTION value=869>Uruguay</OPTION>
    </select>
  </p>
</form>
```
<OPTION value=871>Venezuela</OPTION>
<OPTION value=852>Vietnam</OPTION>
<OPTION value=707>Yugoslavia</OPTION>
<OPTION value=825>Zimbabwe</OPTION>
</select>
<p>
<input type=submit value=Submit>
</form>

**The custom configuration file used to process this form:**

helpnow FSSO configuration:

```
[forms-sso-login-pages]
login-page-stanza = helpnow

[helpnow]
# The HelpNow site redirects you to this page
# you are required to log in.
login-page = /bluebase/bin/files/wcls_hnb_welcomePage1.cgi

# The login form is the first in the page, so we can just call it
# '*'.
login-form-action = *

# The GSO resource, helpnow, contains the employee serial number.
gso-resource = helpnow

# Authentication arguments follow.
argument-stanza = auth-data

[auth-data]
# The 'data' field contains the employee serial number.
data = gso:username

# The Cntselect field contains a number corresponding to the employee's
# country of origin. The string "897" corresponds to the USA.
Cntselect = string:897
```
Chapter 12. Application integration

WebSEAL supports third-party application integration through environment variables and dynamic URL capability. WebSEAL extends the range of environment variables and HTTP headers to enable third-party applications to perform operations based on a client’s identity. In addition, WebSEAL can provide access control on dynamic URLs, such as those that contain query text.

Topic Index:
- “Supporting CGI programming” on page 336
- “Supporting back-end server-side applications” on page 338
- “Junction "best practices" for application integration” on page 339
- “Building a custom personalization service” on page 341
- “Maintaining session state between client and back-end applications” on page 343
- “Providing access control to dynamic URLs” on page 347
- “Dynamic URL example: The Travel Kingdom” on page 353
Supporting CGI programming

To support CGI programming, WebSEAL adds three additional environment variables to the standard set of CGI variables. These environment variables can be used by CGI applications running on either the local WebSEAL server or a junctioned back-end server. The variables provide Tivoli Access Manager-specific user, group, and credential information to the CGI application.

On a local WebSEAL server, these environment variables are automatically available to CGI programs.

Environment variables used by a CGI application running on a junctioned third-party server are produced from the HTTP header information passed to the server from WebSEAL. You must use the -c option to create a junction that supports Tivoli Access Manager-specific header information in HTTP requests destined for a back-end server.

See also “Supplying client identity in HTTP headers (–c)” on page 298.

Additional Tivoli Access Manager-specific environment variables:

<table>
<thead>
<tr>
<th>CGI Environment Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_IV_USER</td>
<td>The Tivoli Access Manager user account name of the requester.</td>
</tr>
<tr>
<td>HTTP_IV_GROUPS</td>
<td>The Tivoli Access Manager groups to which the requester belongs. Specified as a comma-separated list of groups — each group is surrounded by double-quotation marks.</td>
</tr>
<tr>
<td>HTTP_IV_CREDS</td>
<td>Encoded opaque data structure representing a Tivoli Access Manager credential. Supplies credentials to remote servers so mid-tier applications can use the authorization API to call the authorization service. Refer to the IBM Tivoli Access Manager Authorization C API Developer’s Reference.</td>
</tr>
</tbody>
</table>

The REMOTE_USER variable on a local WebSEAL server:

On a local server environment controlled by WebSEAL, the value of the HTTP_IV_USER variable listed above is provided as the value for the standard REMOTE_USER variable. Note that the REMOTE_USER variable may also be present in the environment of a CGI application running on a junctioned back-end server. However, in this situation, its value is not controlled by WebSEAL.

<table>
<thead>
<tr>
<th>CGI Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE_USER</td>
<td>Contains the same value as the HTTP_IV_USER field.</td>
</tr>
</tbody>
</table>

UTF-8 environment variables for CGI programs

CGI scripts use environment variables to communicate with WebSEAL, and the environment variables must be in the local code page. Legacy CGI scripts expect raw (binary) local code page strings.

To enable CGI scripts to understand environment variable values that can consist of UTF-8 data, WebSEAL provides additional environment variables. These variables have the same names as current CGI variables, but with the characters
“_UTF8” appended to the end. The values of these variables are URI (Uniform Resource Indicator) encoded UTF-8 strings. URI encoding is used to prevent data loss on platforms which expect local code page environment variables in spawned processes.

The variables are:
• REMOTE_USER_UTF8
• IV_USER_UTF8
• HTTP_IV_USER_UTF8
• IV_GROUPS_UTF8
• HTTP_IV_GROUPS_UTF8

New CGI programs should use these variables because their values contain UTF-8 data. WebSEAL stores the data for these variables internally in UTF-8 format. The data must be converted to local code page in order for CGI programs to use it. When the old CGI variables (for example, REMOTE_USER) are used, and the local code page is not UTF-8 encoded, the conversion of the UTF-8 data to the local code page can, in some cases, result in data corruption.

**Windows: Supporting WIN32 environment variables**

This section applies to local junctions only.

Windows does not automatically make all of its system environment variables available to processes such as CGI applications. Typically, the system environment variables you require are present.

However, if any Windows system environment variables that you require are not present in the CGI environment, you can explicitly make them available to CGI programs through the WebSEAL configuration file. (Note that the Tivoli Access Manager environment variables mentioned in the previous section are automatically available on all platforms).

Add any of the required Windows system environment variables to the [cgi-environment-variables] stanza of the WebSEAL configuration file. Use the following format:

```
ENV = variable_name
```

For example:

```
[cgi-environment-variables]
#ENV = SystemDrive
ENV = SystemRoot
ENV = PATH
ENV = LANG
ENV = LC_ALL
ENV = LC_CTYPE
ENV = LC_MESSAGES
ENV = LOCPATH
ENV = NLSPATH
```

Any uncommented lines are inherited by a CGI environment.
Supporting back-end server-side applications

WebSEAL also provides support for executable code that runs as an embedded component of a back-end Web server. Examples of such server-side executable code include:

- Java servlets
- Cartridges for Oracle Web Listener
- Server-side plug-ins

When you create a junction to a back-end server using the -c option, WebSEAL inserts Tivoli Access Manager-specific client identity and group membership information into the HTTP headers of requests destined for that server.

The Tivoli Access Manager-specific HTTP header information enables applications on junctioned third-party servers to perform user-specific actions based on the client’s Tivoli Access Manager identity.

WebSEAL provides the following Tivoli Access Manager-specific HTTP headers:

<table>
<thead>
<tr>
<th>PD-specific HTTP Header Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iv-user</td>
<td>The short or long name of the client. Defaults to “Unauthenticated” if client is unauthenticated (unknown).</td>
</tr>
<tr>
<td>iv-groups</td>
<td>A list of groups to which the client belongs. Specified as a comma separated list of quoted groups.</td>
</tr>
<tr>
<td>iv-creds</td>
<td>Encoded opaque data structure representing a Tivoli Access Manager credential. Supplies credentials to remote servers so mid-tier applications can use the authorization API to call the authorization service. Refer to the IBM Tivoli Access Manager Authorization C API Developer’s Reference.</td>
</tr>
</tbody>
</table>

These HTTP headers are available to CGI applications as the environment variables HTTP_IV_USER, HTTP_IV_GROUPS and HTTP_IV_CREDS. For other non-CGI application frameworks, see their associated product documentation for instructions on extracting headers from HTTP requests.

See also “Supplying client identity in HTTP headers (–c)” on page 298.

Note: To ensure security of the iv_creds value, use SSL junctions.
**Junction "best practices" for application integration**

This section includes “best practices” recommendations when using WebSEAL junctions.

- “Supplying complete HOST header information with -v” on page 339
- “Supporting standard absolute URL filtering” on page 339

**Supplying complete HOST header information with -v**

Virtual host configurations and portal applications require correct IP address information for proper socket connections, and complete server name information for accurate routing.

These special back-end application services require complete server name and port designation information in any requests from browsers. The HOST header of a request contains this information and makes it available to the application. When using WebSEAL junctions, this information is supplied to the HOST header through the use of the -v junction option.

Insufficient or missing server name and port information degrades the performance of virtual hosting and portal applications. In addition, domain cookies set by these applications may not contain sufficient information.

To provide the most complete information to the HOST header, the “best practices” recommendation is to always use both the fully qualified domain name of the junctioned server and the connection port number in the -v option when creating or adding the junction.

The -v option uses the following syntax:

-v fully-qualified-host-name[:port]

For example:

-v xyz.ibm.com:7001

**Note:** The port designation should be supplied only if you are using a non-standard port number.

**Supporting standard absolute URL filtering**

WebSEAL, as a front-end reverse proxy, provides security services to back-end junctioned application servers. Pages returned to the client from back-end applications most often contain URL links to resources located on the back-end junctioned server.

It is important that these links include the junction name to successfully direct the requests back to the correct locations of the resources. WebSEAL uses a set of standard rules to filter static URLs and supply this junction information. Additional configuration is required to filter URLs in scripts and dynamically generated URLs. For detailed information on URL filtering, see “Modifying URLs to back-end resources” on page 287.

WebSEAL’s ability to properly filter absolute URLs from static HTML pages requires information about the server name provided in the -h junction option. This option provides WebSEAL with the name of the back-end junctioned server. Arguments to this option can include:

- Fully qualified domain name of the server
• Short name of the server
• IP address of the server

WebSEAL identifies absolute URLs to filter based on its knowledge of the back-end junctioned server name. Depending on your network environment, the –h short-name configuration might not provide WebSEAL with sufficient information.

In the following example, a junction is created using the following option and argument for a back-end server, located in the ibm.com network, with a short name of “xyz”:
-h xyz

A link on an HTML page from this server appears as follows:

When this page passes to the client during a request, WebSEAL might fail to filter this URL because, based on the information provided by –h, it could not recognize “xyz.ibm.com” as the server name. Without the junction name in the path, a request for the release notes document fails.

To support proper filtering of static absolute URLs, the “best practices” recommendation is to always use the fully qualified domain name of the junctioned server in the –h option when creating or adding the junction.
Building a custom personalization service

A Web portal, or launch page, is an integrated Web site service that dynamically produces a customized list of Web resources available to a specific user. Resources can include corporate content, support services, and learning tools. The portal output represents a personalized list of resources based on the access permissions for the particular user. The launch page displays only those resources that have the correct access permissions for that user.

You can use WebSEAL configuration options and the authorization API entitlements service to build a custom portal solution in a Tivoli Access Manager environment.

The process flow for building a custom WebSEAL portal service includes the following tasks:

1. Secure policies are formulated and attached at the appropriate points in the protected object resource.
2. Appropriate explicit ACLs are attached to each of these resource objects.
3. The WebSEAL configuration file is edited to include the URL to the portal service, the path of the object space containing the portal resources, and the permission bit required by the user for access to these resources.
4. For each user request to the portal URL, WebSEAL uses the Authorization Entitlement Service to search this object space and produce a list of resources that meet the authorization conditions for that user.
5. WebSEAL places this information in a PD_PORTAL HTTP header that is sent to the back-end (junctioned) portal server.
6. The custom portal service (such as a CGI or servlet) located on the back-end server reads the PD_PORTAL header contents and, for example, maps the contents to descriptions and URL links that are displayed to the user on a Web page. This information represents the personalized list of resources available to the user based on access control permissions.

Configuring WebSEAL for a personalization service

1. Create a new WebSEAL junction to the personalization service. For example:
   ```
padmin> server task server_name create -t tcp -h portalhost.abc.com \\ /portal-jct
   ```
2. Edit the WebSEAL configuration file to add a new [portal-map] stanza:
   ```
   [portal-map]
   ```
3. The entry in this stanza identifies the server-relative URL of the portal service program and the region of the object space that is searched for available protected portal resources, followed by the permission required for access. This is the list that is placed in the PD_PORTAL header.
   ```
   [portal-map]
   URL = object_space_region:permission
   ```
4. After adding the stanza and the appropriate mapping entries, WebSEAL (webseald) must be re-started.

Personalization service example

- Create a junction to the portal server:
  ```
padmin> server task web1-webseald-cruz -t ssl -h PORTALL1 /portal
  ```
- Define the region of the WebSEAL protected object space that contains resources available to the personalization service:
pdadmin> objectspace create /Resources "Portal Object Hierarchy" 10
pdadmin> object create /Resources/Content "" 10 ispolicyattachable yes
pdadmin> object create /Resources/Support "" 10 ispolicyattachable yes
pdadmin> object create /Resources/Content/CGI "" 11 ispolicyattachable yes
pdadmin> object create /Resources/Support/Servlet "" 11 ispolicyattachable yes

**Note:** The “ispolicyattachable” argument must be set to “yes” for each resource. The search mechanism selects only qualified resource objects with explicitly set ACLs.

- WebSEAL configuration file:
  
  ```
  [portal-map]
  /portal/servlet/PortalServlet = /Resources:r
  ```

- Portal URL used by the user:
  
  https://WS1/portal/servlet/PortalServlet
Maintaining session state between client and back-end applications

WebSEAL can maintain session state with clients over HTTP and HTTPS. Additionally, you can configure WebSEAL to provide user session information to back-end junctioned application servers. With this user session information, back-end applications can maintain session state with clients.

Background to user session management

A secure connection, or session, between a client and a server requires that the server have the ability to remember—over numerous requests—who it is talking to. The server must have some form of session state information that identifies the client associated with each request.

Without an established session state between client and server, the communication between the client and the server must be renegotiated for each subsequent request. Session state information improves performance by eliminating repeated closing and re-opening of client/server connections. The client can log in once and make numerous requests without performing a separate login for each request.

WebSEAL maintains session state information through the GSKit SSL session ID cache and the WebSEAL session/credentials cache. The GSKit session cache supports HTTPS (SSL) communication when the SSL session ID is used to maintain session state. The WebSEAL credentials cache stores a WebSEAL session ID for each client plus any credential information specific to each client.

You can configure WebSEAL to store a unique User Session ID for each authenticating client as an extended attribute in the credential of each client. Using extended attributes for Tivoli Access Manager objects, you can configure a junction to provide this User Session ID information to the back-end server. An application on this back-end server can take advantage of the user session information to manage the client-server interaction, such as tracking the activity of users.

Enabling user session id management

The user-session-ids parameter in the [session] stanza of the WebSEAL configuration file allows you to enable and disable the creation of a unique User Session ID in the credential of each client making a request. The default value is “no” (disabled):

[session]
user-session-ids = no

To enable the creation of unique User Session IDs, set user-session-ids = yes.

The unique User Session ID is stored in a user’s credential as an extended attribute with a name and value:

tagvalue_user_session_id = user_session_id

In the credential itself, the credential extended attribute name (user_session_id) appears with a “tagvalue_” prefix to prevent any conflicts with other existing information in the credential.

The value of the User Session ID is a string that uniquely identifies a specific session for an authenticated user. The User Session ID is a MIME-64 encoded string that includes the WebSEAL instance names and the standard WebSEAL session ID for the user.
A single user that logs in multiple times (for example, from different machines) has multiple WebSEAL session IDs. Because the User Session ID is based on the WebSEAL session ID, there exists a one-to-one mapping between them. The unique user session ID is stored as an attribute to the user’s credential. This allows the value to be passed across a junction as an HTTP header (using tag-value functionality) and made available to a back-end application.

Inserting credential data into the HTTP header

The goal of user session management is to provide the unique User Session ID to the back-end application server. This goal is accomplished by configuring the HTTP-Tag-Value extended attribute on the junction.

You use the `padmin object modify set attribute` command to set an extended attribute on a junction object in the WebSEAL protected object space.

```
padmin> object modify object_name set attribute attr_name attr_value
```

An attribute (“attr-name”) enables the junction to perform a specific type of functionality. The HTTP-Tag-Value attribute enables the junction to extract a value from a credential extended attribute and send the value to the back-end server in an HTTP header.

The value of the HTTP-Tag-Value extended attribute uses the following format:

```
credential_extended_attribute_name=http_header_name
```

For User Session ID data, the `credential_extended_attribute_name` entry is the same as the `user_session_id` extended attribute name specified in the configuration file but without the “tagvalue_” prefix. The entry is not case-sensitive. The value of this extended attribute contains the unique User Session ID.

The `http_header_name` entry specifies the name of the HTTP header used to deliver the data across the junction. In this example, a header called PD-USER-SESSION-ID is used:

```
padmin> object modify /WebSEAL/WS1/junctionA set attribute \ HTTP-Tag-Value user_session_id=PD-USER-SESSION-ID
```

When WebSEAL processes a user request to a back-end application server, it looks for any HTTP-Tag-Value extended attributes configured on the junction object.

In this example, the configured junction looks at the credential of the user making the request, extracts the User Session ID value from the `tagvalue_user_session_id` extended attribute in the credential, and places the value in an HTTP header as:

```
PD-USER-SESSION-ID:user_session_id
```

In summary:

<table>
<thead>
<tr>
<th>Value of HTTP-Tag-Value attribute set on the junction object:</th>
<th>user_session_id=PD-USER-SESSION-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute name and value as they appear in the user credential:</td>
<td>tagvalue_user_session_id:user_session_id_number</td>
</tr>
<tr>
<td>HTTP header name and value:</td>
<td>PD-USER-SESSION-ID:user_session_id_number</td>
</tr>
</tbody>
</table>

If the back-end application is a CGI application, the CGI specification dictates that HTTP headers are made available to CGI programs as environment variables in the form:
Terminating user sessions

A user can initiate the termination of the current session through the pkmslogout command. Additionally, the information in the User Session ID allows administrators and back-end applications to track and manage users. This section describes two methods of terminating the user session at an administration level:

- “Using Administration API to terminate single user sessions” on page 345
- “Using pdadmin to terminate all user sessions” on page 345

Using Administration API to terminate single user sessions

A back-end application can use the Tivoli Access Manager administration API to terminate a specific user session based on the User Session ID passed across the junction. The application invokes the ivadmin_server_performtask() function inside its terminate code. The WebSEAL server instance and the User Session ID are included as parameters to this function.

WebSEAL verifies that the back-end server initiating the terminate operation has appropriate permissions before terminating the user session.

It is important to consider the conditions under which this command might be used. If the intent is to make sure a user is removed from the secure domain entirely, the termination of a single user is only effective when, in addition, the account for that user is made invalid (removed). Certain authentication methods—such as Basic Authentication, client-side certificate, and failover cookies—return cached authentication information automatically with no user intervention. The termination action would not prevent return logins for a user using any of those authentication methods. You must additionally invalidate the appropriate user account in the registry.

Refer to the IBM Tivoli Access Manager for e-business Administration C API Developer Reference for further information.

Using pdadmin to terminate all user sessions

An administrator can use the pdadmin utility to terminate all sessions for a specified user based on the user ID.

```
padmin> server task instance_name=webseald-host_name terminate all_sessions user_id
```

The WebSEAL credentials cache is organized to cross-reference user ID, WebSEAL session ID, and cache entry information. A user always has the same user ID across multiple sessions. Each WebSEAL session ID, however, is unique. The terminate all_sessions command removes all cache entries belonging to the user_id.

WebSEAL checks for appropriate permissions for the administrator initiating the command before terminating user sessions.

It is important to consider the conditions under which this command might be used. If the intent is to make sure a certain group of users are removed from the secure domain entirely, the terminate all_sessions command is only effective when, in addition, the accounts for those users are made invalid (removed). Certain authentication methods—such as Basic Authentication, client-side certificate, and failover cookies—return cached authentication information
automatically with no user intervention. The **terminate all_sessions** command would not prevent return logins for users using any of those authentication methods. You must additionally invalidate the appropriate user accounts in the registry.
Providing access control to dynamic URLs

The current Web environment gives users immediate access to rapidly changing information. Many Web applications dynamically generate Uniform Resource Locators (URLs) in response to each user request. These dynamic URLs may exist only for a short time. Despite their temporary nature, dynamic URLs still need strong protection from unwanted use or access.

Dynamic URL components

Some sophisticated Web application tools use standard Web browsers to communicate with application servers through the CGI interface of a Web server.

All these tools use dynamic URLs and hidden form elements to communicate the requested operation (with its parameter value) to the application server. A dynamic URL augments the standard URL address with information about the specific operation and its parameter values. The query string portion of the URL provides operations, parameters, and values to the Web application interface.

![Figure 21. Passing data to a CGI gateway using a URL](image)

Mapping ACL and POP objects to dynamic URLs

WebSEAL uses the protected object space model, access control lists (ACL), and protected object policies (POP) to secure dynamically generated URLs, such as those generated by database requests. Each request to WebSEAL is resolved to a specific object as the first step in the authorization process. An ACL/POP applied to the object dictates the required protection on any dynamic URL mapped to that object.

Because dynamic URLs exist only temporarily, it is not possible to have entries for them in a pre-configured authorization policy database. Tivoli Access Manager solves this problem by providing a mechanism where many dynamic URLs can be mapped to a single static protected object.

Mappings from objects to patterns are kept in a plain text configuration file:

```
/opt/pdweb/www/lib/dynurl.conf
```

The location of this file (relative to the server-root) is defined by the `dynurl-map` parameter in the `[server]` stanza of the WebSEAL configuration file:

```
[server]
dynurl-map = lib/dynurl.conf
```

You must create this file; the file does not exist by default. The existence of this file (with entries) enables the dynamic URL capability.
Note: If you use Web Portal Manager to view the file and the file does not exist, the following error message is displayed:

The dynurl configuration file /opt/pdweb/www-instance/lib/dynurl.conf cannot be opened for reading.

You can eliminate this error message by creating the file. To create the file, enter text in the text area and click Apply.

Edit this file to modify these mappings. Entries in the file are of the format:

object template

You can use the Tivoli Access Manager Web Portal Manager to edit this file remotely. In Web Portal Manager, select the "Dynamic URL Files" link from the "WebSEAL" menu. The Dynamic URL page allows you to select a WebSEAL server and then view, edit, and save the dynurl.conf configuration file located on that server.

Tivoli Access Manager uses a subset of UNIX shell pattern matching (including wildcards) to define the set of parameters that constitute one object in the object space. Any dynamic URL that matches those parameters is mapped to that object.

Tivoli Access Manager supports the following UNIX shell pattern-matching characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>The character that follows the backslash is part of a special sequence. For example, \t is the TAB character. Can also act as an escape character.</td>
</tr>
<tr>
<td>?</td>
<td>Wildcard that matches a single character. For example, the string “abcde” is matched by the expression “ab?de”</td>
</tr>
<tr>
<td>*</td>
<td>Wildcard that matches zero or more characters.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Defines a set of characters, from which any can match. For example, the string “abcde” is matched with the regular expression “ab[cty]de”</td>
</tr>
<tr>
<td>^</td>
<td>Indicates a negation. For example, the expression [^ab] matches anything but the ‘a’ or ‘b’ characters.</td>
</tr>
</tbody>
</table>

The following example illustrates the form of a dynamic URL that performs credit balance lookup:

http://server-name/home-bank/owa/acct.bal?acc=<account-number>

The object that represents this dynamic URL would appear as follows:

http://server-name/home-bank/owa/acct.bal?acc=*

Careful examination of the dynamic URL in this example shows that it describes a specific account number. The object for account balances at home-bank shows that the ACL and POP permissions apply to any account, because the last portion of the entry (acc=*) uses the asterisk wildcard which matches all characters.

The following figure illustrates a complete scenario of a specific dynamic URL mapped to a specific protected object:
Updating WebSEAL for dynamic URLs

Use the `dynurl update` command to update the WebSEAL protected object space with entries made in the `dynurl1.conf` configuration file.

1. Create, edit, or delete a dynamic URL entry in the `dynurl1.conf` configuration file.
2. After making your changes, use the `dynurl update` command to update the server:

   ```
   pdadmin> server task instance_name=webseald-host_name dynurl update
   ```

   The `server-name` argument represents the unqualified host name of the WebSEAL machine.

Resolving dynamic URLs in the object space

The resolving of a dynamic URL to an object is dependent upon the ordering of the entries in the `dynurl1.conf` configuration file.

When attempting to map a dynamic URL to an object entry, the list of mappings in the `dynurl1.conf` file is scanned from top to bottom until the first matching pattern is found. When the first match is found, the corresponding object entry is used for the subsequent authorization check.

If no matches are found, WebSEAL uses the URL itself, minus the `http://server` portion of the path.

Keep the mappings that correspond to the most restrictive ACLs higher up in the list. For example, if the `book.sales` procedure of a sales order application is to be restricted to just a book club group, but the rest of the sales order application can be accessed by all users, then the mappings should be in the order shown in the following table:
<table>
<thead>
<tr>
<th>Object Space Entry</th>
<th>URL Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ows/sales/bksale</td>
<td>/ows/db-apps/owa/book.sales*</td>
</tr>
<tr>
<td>/ows/sales/general</td>
<td>/ows/db-apps/owa/*</td>
</tr>
</tbody>
</table>

Note that if the mapping entries were in the reverse order, all stored procedures in the /ows/db-apps/owa directory would map to the /ows/sales/general object. This could lead to possible breaches of security, due to this incorrect object space resolution.

When you map a URL regular expression to an object space entry, the URL format should take on the format as produced from the GET method — regardless of whether the POST or GET method is being used.

In the GET method of data transmission, the dynamic data (such as the data supplied by a user in a form) is appended to the URL.

In the POST method of data transmission, the dynamic data is included in the body of the request.

**ACL and POP Evaluation**

As soon as the dynamic URL has been resolved to an object space entry, the standard ACL/POP inheritance model is used to determine if the request should be processed or forbidden (due to insufficient privilege).

**Configuring limitations on POST requests**

The content of a POST request is contained in the body of the request. In addition, a POST request contains the browser-determined length of this content and lists the value in bytes.

**request-body-max-read**

The `request-body-max-read` parameter in the `[server]` stanza of the WebSEAL configuration file limits the impact of large POST requests on WebSEAL by specifying the maximum number of bytes to read in as content from the body of POST requests. The content read in by WebSEAL is subject to authorization checks, as described earlier in this section.

The `request-body-max-read` parameter value is considered when the POST request is used for dynamic URL processing or Forms authentication. The default value is 4096 bytes:

```
[server]
request-body-max-read = 4096
```

Note that this parameter does not limit the maximum POST content size (which is unlimited). The parameter protects WebSEAL from processing a POST request of unreasonable size. For more information on modifying `request-body-max-read`, see "Modifying request-body-max-read" on page 202.

**dynurl-allow-large-posts**

Although the `request-body-max-read` parameter limits the amount of POST content read and processed by WebSEAL, it does not prevent the request, in its entirety, from being passed through to the application server. In this scenario,
non-validated content is passed through to the application server. If the application server does not have its own authorization capabilities, the situation might result in a security risk.

The **dynurl-allow-large-posts** parameter allows you to control the way WebSEAL handles POST requests that have a content length larger than that specified by **request-body-max-read**. If the parameter value is set to “no” (default), WebSEAL rejects, in total, any POST request with a content length larger than that specified by **request-body-max-read**.

```
[server]
dynurl-allow-large-posts = no
```

If the parameter value is set to “yes”, WebSEAL accepts the entire POST request, but only validates the amount of content equal to the **request-body-max-read** value.

```
[server]
dynurl-allow-large-posts = yes
```

**Example 1:**
- A large POST request is received (greater than the **request-body-max-read** value).
- `dynurl-allow-large-posts = no`
- Dynamic URLs are enabled.
- Result: 500 “Server Error”

**Example 2:**
- A large POST request is received (greater than the **post-request-body-max-read**).
- `dynurl-allow-large-posts = yes`
- Dynamic URLs are enabled.
- Result: WebSEAL compares the amount of content up to **request-body-max-read** with each of the regular expressions in the `dynurl.conf` configuration file, and performs an authorization check on the corresponding object if a match is found. Otherwise, the authorization check is performed on the object corresponding to the URL received, as usual. The portion of the request body past **request-body-max-read** is not validated.
- The following template contains the type of pattern matching arrangement that invites misuse by a large POST request:

```
/rtpi153/webapp/examples/HitCount/?action=reset
```

**Summary and technical notes**

**Summary:**
- To configure WebSEAL to securely handle dynamic URLs, create the following file:

```
/opt/pdweb/www/lib/dynurl.conf
```
- The file must contain one or more lines of the format:

```
object template
```
- If the file does not exist, or is empty, dynamic URL capability is not enabled.
- After the file has been processed, the object name appears as a child resource in the WebSEAL object space.
- The template can contain a subset of the standard pattern matching characters. The template can also be an exact string with no pattern matching characters.
The following sample dynur1.conf file defines three objects representing some of the sample Web applications that are part of the IBM WebSphere product:

<table>
<thead>
<tr>
<th>Object Entry</th>
<th>URL Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>/app_showconfig</td>
<td>/rtpi153/webapp/examples/ShowConfig*</td>
</tr>
<tr>
<td>/app_snoop</td>
<td>/rtpi153/servlet/snoop</td>
</tr>
<tr>
<td>/app_snoop</td>
<td>/rtpi025/servlet/snoop</td>
</tr>
<tr>
<td>/app_hitcount/ejb</td>
<td>/rtpi153/webapp/examples/HitCount?source=EJB</td>
</tr>
<tr>
<td>/app_hitcount</td>
<td>/rtpi153/webapp/examples/HitCount*</td>
</tr>
</tbody>
</table>

**Technical Notes:**

- Multiple URL templates can be mapped to the same object (for example, app_snoop maps to URLs on two different servers).
- Objects can be nested (for example, app_hitcount and app_hitcount/ejb).
- An incoming URL request is compared against templates in order, from top to bottom. When a match is found, processing stops. Therefore, place the more restrictive templates at the top of the file.
- To activate the definitions in the dynur1.conf file, issue the dynurl update command (use pdadmin server task).
  The update occurs immediately and the objects appear in the Web Portal Manager when you refresh the protected object space view.
- Avoid uppercase characters in the object name. Use lowercase characters only.
- Do not use an object name that already exists in the protected object space.
- Before deleting an object in the dynur1.conf file, remove any ACLs attached to the object.
Dynamic URL example: The Travel Kingdom

The following example illustrates how a corporate intranet can secure URLs generated by an Oracle Web Listener.

The dynamic URL Web server used in this example is the Oracle Web Listener. This technology can be applied equally to other dynamic URL Web servers.

The application

Travel Kingdom is an organization that offers clients a travel booking service over the Internet. The business intends to operate two Oracle database applications on its Web server — accessible from within the corporate firewall and across the Internet.

1. Travel Booking System

Authorized customers can make bookings remotely and query their own current bookings. Travel Kingdom staff can also make bookings for telephone customers, process changes, and perform many other transactions. Because external customers pay for services with credit cards, the transmission of that information must be strongly secured.

2. Administration Manager

Like most other companies, Travel Kingdom maintains an administration database containing salary, position, and experience information. This data is also accompanied by a photograph of each member of the staff.

The interface

An Oracle Web Server is configured to provide access to the following stored procedures in the database:

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/db-apps/owa/tr.browse</td>
<td>Gives all users the ability to inquire about travel destinations, prices, and so on.</td>
</tr>
<tr>
<td>/db-apps/owa/tr.book</td>
<td>Used to place a booking (travel agent staff or authenticated customers).</td>
</tr>
<tr>
<td>/db-apps/owa/tr.change</td>
<td>Used to review or change current bookings.</td>
</tr>
<tr>
<td>/db-apps/owa/admin.browse</td>
<td>Used by any staff member to view unrestricted staff information, such as extension number, email address, and photograph.</td>
</tr>
<tr>
<td>/db-apps/owa/admin.resume</td>
<td>Gives staff members the ability to view or change their own resume information in the Administration database.</td>
</tr>
<tr>
<td>/db-apps/owa/admin.update</td>
<td>Used by Administration staff to update information on staff.</td>
</tr>
</tbody>
</table>

Web space structure

A WebSEAL server is used to provide a secure interface to the unified Web space of Travel Kingdom.

- A junction (/ows) is made to the Oracle Web Server running both the travel booking application and the administration application.
The security policy

To provide suitable security to Web resources, while retaining an easy-to-use system, the business has established the following security goals:

- Travel agent staff have complete control over all bookings.
- Authenticated customers can make and change their own bookings, but cannot interfere with the travel data of other authenticated customers.
- Administration staff have complete access to all of the administration information.
- Travel Kingdom staff other than the Administration department can change their own resume information and view partial information of other members of staff.

Dynamic URL to object space mappings

To achieve the security goals described above, the mappings from dynamic URLs to ACL object entries need to be configured as shown in the following table.

Remember that the ordering of these mappings is an important part of achieving the security goals discussed earlier.

<table>
<thead>
<tr>
<th>Object Space Entry</th>
<th>URL Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ows/tr/browse</td>
<td>/ows/db-apps/owa/tr.browse?dest=*&amp;date=??/??/????</td>
</tr>
<tr>
<td>/ows/tr/auth</td>
<td>/ows/db-apps/owa/tr.book?dest=*&amp;depart=??/??/????&amp;return=??/??/????</td>
</tr>
<tr>
<td>/ows/tr/auth</td>
<td>/ows/db-apps/owa/tr.change</td>
</tr>
<tr>
<td>/ows/admin/forall</td>
<td>/ows/db-apps/owa/admin.resume</td>
</tr>
<tr>
<td>/ows/admin/forall</td>
<td>/ows/db-apps/owa/admin.browse?empid=th???</td>
</tr>
<tr>
<td>/ows/admin/auth</td>
<td>/ows/db-apps/owa/admin.update?empid=????</td>
</tr>
</tbody>
</table>

Secure clients

Client authenticate to WebSEAL over a secure, encrypted channel.

Customers who wish to use the Web interface must additionally register with the Travel Kingdom Webmaster to receive an account.

Account and group structure

Four groups are created on the system:

- **Staff**: Members of the Travel Kingdom organization.
- **TKStaff**: Travel Kingdom travel agents.
- **AdminStaff**: Members of the Travel Kingdom Administration Department. Note that Administration staff members are also in the Staff group.
- **Customer**: Customers of Travel Kingdom who want to make their travel bookings across the Internet.

Each user is given an account in the secure domain to be individually identified by the WebSEAL server. The user’s identity is also passed to the Oracle Web Servers to provide a single sign-on solution to all of the Web resources.

Access control

The following table lists the access controls resulting from application of the preceding information:
Customers and TKStaff have the same privileges on the booking and travel plan maintenance objects, except that the customers must encrypt information (privacy permission) to give them further security when submitting sensitive data (such as credit card information) across the untrusted Internet.

**Conclusion**

This simple example illustrates the concepts of deploying a system capable of:

- Securing sensitive information
- Authenticating users
- Authorizing access to sensitive information

In addition, the identities of the authenticated users of the system are known to both the WebSEAL and Oracle Web Servers, and are used to provide an auditable, single sign-on solution.
Chapter 13. Authorization decision information retrieval

This chapter contains information that describes how WebSEAL can provide, or acquire, authorization decision information (ADI) required to evaluate authorization rules that protect resources in the Tivoli Access Manager domain.

Topic Index:
- “Overview of ADI retrieval” on page 358
- “Retrieving ADI from the WebSEAL client request” on page 359
- “Retrieving ADI from the user credential” on page 362
- “Supplying a failure reason across a junction” on page 363
- “Dynamic ADI retrieval” on page 364
Overview of ADI retrieval

The Tivoli Access Manager authorization rules evaluator performs authorization decisions based on Boolean logic applied to specific access decision information (ADI). Detailed information on the construction of authorization rules (using Boolean logic) and authorization decision information (ADI) can be found in the *IBM Tivoli Access Manager Base Administration Guide*.

ADI required for rules evaluation can be retrieved from the following sources:

- **Authorization decision parameters provided to the authorization rule as ADI by the authorization service**
  Parameters include the target resource (protected object) and the requested action on the resource. Refer to the *IBM Tivoli Access Manager Base Administration Guide* for further information on this topic.

- **The user credential**
  The user credential is always included with the function call to the authorization rules evaluator, so it is immediately available.

- **The resource manager environment (application context)**
  A resource manager, such as WebSEAL, can be configured to provide ADI from its own environment. For example, WebSEAL has the capability to provide ADI contained in parts of the client request. A special prefix is used in the authorization rule to "trigger" this type of ADI source.

- **An external source through the Tivoli Access Manager attribute retrieval service**
  ADI can be obtained externally through the attribute retrieval service. A call is made to the attribute retrieval service through the resource manager’s entitlement service. ADI from the external source is returned in XML format to the authorization rules evaluator.
Retrieving ADI from the WebSEAL client request

In a WebSEAL environment, authorization rules can be written to require authorization decision information (ADI) contained in the client HTTP/HTTPS request. ADI can be contained in the request header, the request query string, and the request POST body.

Authorization decision information is referred to by an XML container name in authorization rules. A special WebSEAL-specific prefix in the container name is used to alert the authorization rules evaluation process that WebSEAL can interpret this parameter correctly and return a value.

Prefixes can be specific to any resource manager. Accordingly, the resource manager must be designed to respond appropriately to a request for ADI.

The following container names contain prefixes that are appropriate for WebSEAL:

- **AMWS_hd_name**
  Request header container name. The value of the HTTP header called `name` in the HTTP request is returned to the authorization rules evaluator as ADI.
- **AMWS_qs_name**
  Request query string container name. The value of `name` in the request query string is returned to the authorization rules evaluator as ADI.
- **AMWS_pb_name**
  Request POST body container name. The value of `name` in the request POST body is returned to the authorization rules evaluator as ADI.

The following process flow helps illustrate how prefixes enable the extraction of ADI from client requests:

1. An authorization rule is written that requires ADI from the client request (for example, a specific HTTP header in the request).
   In this example, the `AMWS_hd_` prefix is used in the container name specified in the rule. The prefix must be specified by the `resource-manager-provided-adi` parameter in the `[aznapi-configuration]` stanza of the WebSEAL configuration file. The authorization service incorporates this configuration information during its initialization. This WebSEAL-specific prefix alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

2. The authorization rules evaluation process tries to evaluate, for example, the `AMWS_hd_host` container name in a rule. The `AMWS_hd_` prefix alerts the authorization evaluation process that WebSEAL can interpret this container name correctly and return a value.

3. The `AMWS_hd_host` container name is sent to WebSEAL.
   WebSEAL is designed to recognize and interpret the `AMWS_hd_` prefix.

4. WebSEAL responds to the `AMWS_hd_host` container name by looking for the "host" header in the client request and extracting the value associated with that header.

5. WebSEAL returns the "host" header value (as an XML container) to the authorization rules evaluation process.

6. The authorization rules evaluation process uses the value as ADI in its evaluation of the rule.
The resource-manager-provided-adi parameter in the [aznapi-configuration] stanza of the WebSEAL configuration file specifies—the prefix(es) that can be used in container names specified by authorization rules. To specify multiple prefixes, use multiple entries of the resource-manager-provided-adi parameter:

```xml
[aznapi-configuration]
resource-manager-provided-adi = AMWS_qs_
resource-manager-provided-adi = AMWS_pb_
resource-manager-provided-adi = AMWS_hd_
```

The permission-infoReturned parameter in the [aznapi-configuration] stanza of the WebSEAL configuration file appears by default. This parameter specifies the permission information returned to the resource manager (for example, WebSEAL) from the authorization service. The following example is entered as one line, with a single space separating the two permission types:

```xml
[aznapi-configuration]
permission-info-returned = azn_perminfo_rules_adi_request
azn_perminfo_reason_rule_failed
```

The `azn_perminfo_rules_adi_request` setting allows the authorization service to request ADI from the current WebSEAL client request. The `azn_perminfo_reason_rule_failed` setting specifies that rule failure reasons be returned to the resource manager (this setting is required for junctions—see “Supplying a failure reason across a junction” on page 363).

**Example: Retrieving ADI from the request header**

The following example authorization rule requires the name of the client machine’s host name. The client request is set up to include the host name value in the "host" header of the request. The use of the `AMWS_hd_` prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

```xml
<xsl:if test='AMWS_hd_host = "machineA"'>!TRUE!</xsl:if>
```

WebSEAL is designed to know how to handle the extraction of ADI information from the request:

```xml
[aznapi-configuration]
resource-manager-provided-adi = AMWS_hd_
```

WebSEAL understands this information can be found in the request header name "host". WebSEAL extracts the value contained in the "host" header and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request’s "host" header is "machineA".

In a similar manner, information required to evaluate an authorization rule can come from the request POST body or the query string of the request.

**Example: Retrieving ADI from the request query string**

The following example authorization rule requires the name of the client’s zip code as passed in the query string of a GET request (as submitted in response to a form). The client request is set up to include the zip code value in the "zip" field of the request query string.

https://www.service.com/location?zip=99999
The use of the AMWS_qs_prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

WebSEAL is designed to know how to handle the extraction of ADI information from the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_qs_
```

WebSEAL understands this information can be found in the request query string under the field name “zip”. WebSEAL extracts the value contained in the “zip” field and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request’s query string "zip" field is "99999". In a similar manner, information required to evaluate an authorization rule can come from the request POST body or the request header.

**Example: Retrieving ADI from the request POST body**

The following example authorization rule requires the name of the client’s total purchase amount from a Web shopping cart as passed in the body of a POST request (as submitted in response to a form). The client request is set up to include the total purchase value in the "purchase-total" field of the request POST body.

The use of the AMWS_pb_prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

```
<xsl:if test="AMWS_pb_purchase-total &lt; "1000.00">!TRUE!</xsl:if>
```

WebSEAL is designed to know how to handle the extraction of ADI information from the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_pb_
```

WebSEAL understands this information can be found in the request POST body under the field name "purchase-total". WebSEAL extracts the value contained in the "purchase-total" field and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request’s POST body "purchase-total" field is less than "1000.00". In a similar manner, information required to evaluate an authorization rule can come from the request header or the query string of the request.
Retrieving ADI from the user credential

Authorization rules can be written to use ADI that is provided initially to the authorization rules evaluator as part of the credential. The initial call to the authorization service (azn_decision_access_allowed_ext0) actually contains the user’s credential information. The authorization rules evaluator always looks through this credential information for any ADI required by the rule being processed. The authorization rule can use the value from any field in the credential, including extended attributes added to the credential during authentication.
Supplying a failure reason across a junction

Authorization rules allow you to set up special, and often complex, conditions governing the ability to access a protected resource. However, the standard result of a failed authorization decision is to stop the progress of the request to the service application that controls the resource, and present the client with a "forbidden" message. If the authorization rule is written to include a failure reason, and is evaluated as FALSE by the Tivoli Access Manager authorization rules evaluator, WebSEAL receives the reason for the rule’s failure along with the standard "forbidden" message from the authorization service. The failure reason is usually ignored and the "forbidden" decision is enforced.

You can optionally configure WebSEAL to reject this standard response and allow denied requests to proceed across a junction to a back-end service application. The request is accompanied by the failure reason provided in the authorization rule. The back-end service application can then have the opportunity to proceed with its own response to the situation. This optional configuration occurs during the creation of the junction to the back-end service application.

Authorization rules are typically used in conjunction with service applications that can understand and handle this more sophisticated level of access control. In some cases, it is necessary for the service application to receive a request that is denied by the Tivoli Access Manager authorization service. Such an application is written to understand failure reason information and can provide its own response to a request that has failed a Tivoli Access Manager authorization rule.

For example, the order processing component of a shopping cart application can be governed by an authorization rule that denies action on an order if the total purchase price exceeds the user’s credit limit. It is important for the shopping cart application to receive the entire request and the reason for failure. Now the shopping cart application can take matters into its own hands and provide a user-friendly response, such as advising the user to eliminate a portion of the order. The interaction with the user is preserved rather than cut off.

To allow denied requests and failure reason information to proceed across a junction to the back-end service application, configure the junction with the –R option. When WebSEAL receives an access denied decision on a request for an object located on a –R junction, WebSEAL reverses the denial response, inserts the failure reason into an HTTP header called "AM_AZN_FAILURE", inserts that header into the request, and passes the request on to the back-end application.

Always use this option with caution. It is important to coordinate the use of failure reasons in authorization rules with a service application’s ability to interpret and respond to this information. You do not want to accidently create a situation where access is granted to a resource controlled by an application that cannot respond accurately to the AM_AZN_FAILURE header.
Dynamic ADI retrieval

Rules can be written requiring ADI that cannot be found in any of the information that the Tivoli Access Manager authorization service has access to. In these cases, it is necessary to retrieve the ADI from an outside source. This retrieval can be performed in real-time by a dynamic ADI entitlement retrieval service. The attribute retrieval service, currently provided with WebSEAL, is one type of entitlement retrieval service.

The attribute retrieval service provides communication and format translation services between the WebSEAL entitlement service library and an external provider of authorization decision information. The process flow for the attribute retrieval service is described below:

1. The client makes a request for resource protected by an authorization rule.
2. The authorization rules evaluator—part of the authorization service—determines that specific authorization decision information (ADI) is required to complete the evaluation of the rule. The ADI requested is not available from the user credential, the authorization service, or WebSEAL.
3. The task of ADI retrieval is sent to the attribute retrieval service through the entitlements service library. This service formats the request for ADI as a SOAP request. The SOAP request is sent over HTTP to the Web Service Description Language (WSDL) interface of the attribute retrieval service.
4. The attribute retrieval service formats the request appropriately for the external provider of ADI.
5. The external provider of ADI returns the appropriate ADI.
6. The ADI is formatted in another SOAP container and returned to the WebSEAL entitlements service. Now the authorization rules evaluator has the necessary information to evaluate the rule and make a decision to accept or deny the original client request.
Deploying the attribute retrieval service

These installation instructions assume that WebSphere Application Server, WebSEAL, and the attribute retrieval service are on the same computer.

Perform the following tasks to deploy the attribute retrieval service with WebSphere Application Server.

1. The Tivoli Access Manager attribute retrieval service is a separately installable package. Install the attribute retrieval service on the same system as WebSphere Application Server. Follow the instructions in *IBM Tivoli Access Manager Base Installation Guide*.

2. Tivoli Access Manager provides a script that programatically deploys the attribute retrieval service into the WebSphere Application Server environment. Follow the instructions in the Readme file.

<table>
<thead>
<tr>
<th>UNIX</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readme</td>
<td>C:\Program Files\Tivoli\AMWebARS\Readme.deploy</td>
</tr>
<tr>
<td>Script</td>
<td>C:\Program Files\Tivoli\AMWebARS\Deploy.bat</td>
</tr>
</tbody>
</table>

Perform the following tasks to configure WebSEAL to use the attribute retrieval service.

1. In the WebSEAL configuration file, specify the identification name (ID) of the attribute retrieval service that is queried when missing ADI is detected during a rules evaluation. In this case, the attribute retrieval service is specified:

   
   [aznapi-configuration]
   dynamic-adi-entitlement-services = AMWebARS_A

2. In the WebSEAL configuration file, use the service ID for the configured attribute retrieval service as a parameter to specify the appropriate built-in library that formats out-bound ADI requests and interprets incoming responses:

   For example:

   
   [aznapi-entitlement-services]
   AMWebARS_A = azn_ent_amwebars

3. In the WebSEAL configuration file, specify the URL to the attribute retrieval service located in the WebSphere environment.

   For a TCP connection:

   
   [amwebars]
   service-url = http://websphere_hostname:websphere_port
   /amwebars/amwebars/ServiceToIServicePortAdapter

4. Restart WebSEAL.
Chapter 14. Attribute retrieval service reference

This chapter contains the administration and configuration reference for the Tivoli Access Manager attribute retrieval service.

Topic Index:
• “Basic configuration” on page 368
• “Editing the data tables” on page 371
• “Creating custom protocol plug-ins” on page 375
Basic configuration

Basic configuration information.

Configuration files

The following attribute retrieval service configuration and XML files are located within the working directory of the supporting application. The current implementation of the attribute retrieval service is installed in a WebSphere environment:

`websphere-install-dir/WebSphere/AppServer/bin/`

**amwebars.conf**
The `amwebars.conf` configuration file contains parameters and values that specify the general configuration of the attribute retrieval service.

**ContainerDescriptorTable.xml**
The `ContainerDescriptorTable.xml` file contains a list of all container descriptors that can be retrieved by the attribute retrieval service. The service only recognizes containers that are described in this table. The table is XML-based.

**ProviderTable.xml**
The `ProviderTable.xml` file contains the description of the providers available for ADI retrieval. The XML-based file contains, for each provider, the provider’s URL and information necessary to connect to the provider and request containers (ADI) from it. You can only refer to providers that are named in this file.

**ProtocolTable.xml**
The `ProtocolTable.xml` file contains the description of the protocols used by the attribute retrieval service. The file contains each protocol’s full qualified class name and the protocol ID. You can only refer to protocols that are named in this file.

Descriptions of amwebars.conf configuration parameters

**Table locations**

**descriptor_table_filename**
The file name of the `ContainerDescriptorTable`. The `ContainerDescriptorTable` contains all `container_type_ids` the service can retrieve.

**provider_table_filename**
The file name of the `ProviderTable`. The `ProviderTable` contains information about the different attribute retrieval service providers used by the attribute retrieval service.

**protocol_table_filename**
The file name of the `ProtocolTable`. The `ProtocolTable` contains information about the different attribute retrieval service protocols used by the attribute retrieval service. Please note that the service only uses this file if the option `protocol_module_load_from_general_config` is set to “false”.

**key_store_filename**
File name of the attribute retrieval service’s keystore. The keystore is a central storage for all client keys used by the attribute retrieval service. It can be administrated with the Java tool keytool.

**key_store_password**

The password to unlock the keystore. Please note that the keys are unlocked independently. The password used to unlock them is stored in the provider’s description.

**Logging**

**exception_logfile_filename**

File name of the logfile where exceptions are logged. The exception logfile contains information about errors and invalid inputs.

**metering_logfile_filename**

File name of the metering logfile. The metering logfile contains one entry for each container retrieved from a provider.

**trace_logfile_filename**

File name of the trace logfile. The trace logfile contains a detailed trace of the service’s program operation.

**exception_logging**

Turns the exception logging on and off. Set the value to "true" to activate exception logging (recommended). Default is true.

**metering_logging**

Turns the metering logging on and off. Set the value to "true" to activate the logging of retrieved containers. Default is true.

**trace_logging**

Turns the trace logging on and off. Set the value to "true" to activate tracing. Be aware that the traces consume a large amount of disk space. Default is "false”.

**use_stderr_for_fatal**

If set to "true", fatal errors in exception logging are not only reported to the logfile, but also to stderr.

**use_stderr_for_exceptions**

If set to "true", all exceptions in exception logging are not only reported to the logfile, but also to stderr.

**traceVerbose_monitor_locks**

If set to "true", all entries of synchronized monitors are reported to trace. This option is used for the search of deadlocks.
trace_verbose_get_entitlement

If set to "true", the trace contains all inputs of the getEntitlement calls. Be aware that this kind of trace might contain personal information about the customer.

Limitation of client and session number
The following options can be used to influence the resource consumption of the attribute retrieval service. These options are for experts only.

limit_number_of_sessions

This value activates the limitation of the session number. If set to "true", the service only generates a limited number of sessions. Default is "false".

max_number_of_sessions

Sets the maximum number of sessions that is generated.

limit_number_of_clients_per_session

This value activates the limitation of the client number per session. If set to "true", a session can only create a fixed number of clients. Default is "false".

max_number_of_clients_per_session

Sets the maximum number of clients a session can generate.

Miscellaneous options
return_ids_full_qualified

The service returns the containers in an attribute list with the container_type_ids as key. By default, the service uses the same format (with namespace or without) as the app_context. By setting this value to "true", you can force the service to always return container_type_ids including the namespace. Default is "false".

Protocol modules to load at initialization
protocol_module_load_from_general_config

The attribute retrieval service dynamically loads protocol modules at initialization time. If this key is set to 'true', the service uses this config file otherwise it uses another XML file specified with the key "protocol_table_filename". If loaded from according to this file it’s based on the entries:

protocol_module_load.*

The package to load.

protocol_module_id.*

The protocol id that should be associated with the protocol.
Editing the data tables

The attribute retrieval service is configured using different data tables. These tables tell the service, for example, what providers can be accessed, what attribute retrieval service containers can be retrieved from them, and what protocol is required to communicate with the provider. The three primary tables include:

- **ContainerDescriptorTable**, which contains all information about the retrievable attribute retrieval service containers
- **ProviderTable**, which contains the attribute retrieval service providers available
- **ProtocolTable**, which describes the protocols used by the attribute retrieval service

### ProviderTable

This table contains information about the providers available to the service. A Provider entry is required in this table for each server that must connect to the attribute retrieval service.

<table>
<thead>
<tr>
<th>Filename</th>
<th>ProviderTable.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>XML</td>
</tr>
<tr>
<td>Table name</td>
<td>ProviderTable</td>
</tr>
<tr>
<td>Element name</td>
<td>Provider</td>
</tr>
</tbody>
</table>

### Provider sub-elements

A Provider element can contain the following sub-elements:

**provider_id**

The ID of the provider (required). The ContainerDescriptors use this ID to refer to a certain provider. The `provider_id` must be unique.

**name**

The name of the provider.

**provider_url**

The URL of the provider’s endpoint (required). This URL is connected by protocols that want to access the provider. To use an HTTPS URL, the Java HTTPS support has to be activated. For example, setting the virtual machine property:

```
Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
```

**client_key_alias**

The protocol uses this alias to lookup the private key and certificate corresponding to this provider in the service’s keystore.

**client_key_password**

The password assigned to the provider’s private key.

### Example ProviderTable

The following code illustrates a valid ProviderTable with one Provider entry:
<xml version="1.0" encoding="UTF-8"?>
<ProviderTable>
  <Provider>
    <provider_id>Erandt_Securities_Entitlements</provider_id>
    <name>ese</name>
    <provider_url>https://rse.erandt.com/responder</provider_url>
    <client_key_alias>erandt_test_account</client_key_alias>
    <client_key_password>changeit</client_key_password>
  </Provider>
</ProviderTable>

ContainerDescriptorTable
The ContainerDescriptorTable describes all containers the attribute retrieval service can retrieve. You have to add a ContainerDescriptor entry to this table if you want the service to retrieve another type of attribute retrieval service container.

<table>
<thead>
<tr>
<th>Filename</th>
<th>ContainerDescriptor.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>XML</td>
</tr>
<tr>
<td>Table name</td>
<td>ContainerDescriptorTable</td>
</tr>
<tr>
<td>Element name</td>
<td>ContainerDescriptor</td>
</tr>
</tbody>
</table>

ContainerDescriptor sub-elements
A ContainerDescriptor element can contain the following sub-elements:

container_type_id

The ID of this ContainerDescriptor and the corresponding container (required). You must refer to this ID to request a container from the attribute retrieval service. It is generated the following way, if the namespace is present:
container_type_id = namespace_prefix + ":" + container_name

If the namespace is not present, it is equal to the container_name. The container_type_id must be unique.

container_name

The name of this container descriptor (required). Within a particular namespace, the container_name must be unique. The container_name must not contain a colon (":"), character.

namespace_prefix

The URL of the namespace in which the container_name is valid (required). The namespace tag can be empty. If this is the case, the container_type_id equals the container_name.

cost

The per retrieval cost of a attribute retrieval service container corresponding to this descriptor. Don’t forget the currency type.

protocol_id
This ID (required) refers to the unique protocol ID of one of the attribute retrieval service protocols. The protocol given with this ID is used to retrieve the container from the provider. This element has to match an ID known to the service.

**provider_id**

This ID (required) refers to the attribute retrieval service provider which is capable of sending a container corresponding to the descriptor. The service connects to this provider when this container is requested.

**properties**

General client and protocol dependent properties. You add a property setting the following way:

Add an element called property with an attribute named key. The attribute contains the name or key of the property, the content of the element, and the corresponding value. Consider the client_init_properties in the example code below.

**client_init_properties**

Properties specific to the initialization of the attribute retrieval service clients. One property used by different protocols is the attribute mapping described below.

**ContainerPayloadFormat**

This element (required) describes the structure and contents of the containers corresponding to this descriptor. The content of this element is protocol dependent. The DynAdiProtocols currently available provides a list of elements named with the attribute names to be retrieved from the provider in this element. The containers are wrapped by a element named with the container_name.

**Attribute mapping**

Attribute mapping might be necessary, if the attribute retrieval service provider uses attribute names not compatible with XML element names. Such a mapping is generated the following way:

The key has the structure:

"map__attribute_name__" + source__provider_attribute_name

if you map one of the provider's attribute names to one of your own, or

"map_attribute_name__" + source_attribute_name

if you do a reverse mapping. The value of such a property contains the attribute name to map to. Please note the such an declaration is only one-way. You must add a second one to generate a reverse-mapping.

**Example ContainerDescriptorTable**

Example for a ContainerDescriptorTable with only one descriptor:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ContainerDescriptorTable>
  <ContainerDescriptor>
    <container_type_id>
      http://ese.erandt.com/attributes:ese__test_container_address_line
    </container_type_id>
    <container_name>ese__test_container_address_line</container_name>
  </ContainerDescriptor>
</ContainerDescriptorTable>
```
ProtocolTable

The ProtocolTable describes all protocols the attribute retrieval service uses. You have to add a Protocol entry to this table if you want the service to retrieve another protocol type.

<table>
<thead>
<tr>
<th>Filename</th>
<th>ProtocolTableer.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>XML</td>
</tr>
<tr>
<td>Table name</td>
<td>ProtocolTable</td>
</tr>
<tr>
<td>Element name</td>
<td>Protocol</td>
</tr>
</tbody>
</table>

**Protocol sub-elements**

A Protocol element can contain the following sub-elements:

- **protocol_id**

  Reference ID used by other tables (required).

- **class_name**

  Full qualified class name of the Java class that corresponds to the attribute retrieval service protocol (required). "Fully qualified" refers to the inclusive package path.

**Example ProtocolTable**

Example for a ProtocolTable with only one protocol:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ProtocolTable>
  <Protocol>
    <protocol_id>file_reader_protocol</protocol_id>
    <class_name>amwebarsentitlementservice.protocol.FileReaderProtocol</class_name>
  </Protocol>
</ProtocolTable>
```
Creating custom protocol plug-ins

Overview

The attribute retrieval service uses a special XML construct, known as a container, to retrieve and convey authorization decision information. An ADI request is always made in the form of a container name. When a request for ADI (as a container name) is received by the attribute retrieval service, the container name is compared against all container names described in the Container Descriptor Table (ContainerDescriptorTable.xml). If a match is found, the process of retrieving the ADI can continue. Information in the container description reveals what ADI is required, where the ADI can be found, and what protocol must be used to communicate with the external provider of the ADI. The ADI, enclosed within opening and closing container name XML tags, is known as a container.

The attribute retrieval service generates a client that uses the necessary protocol to retrieve the ADI from the external provider. If the ADI must be retrieved using a protocol that is not provided by the current release of the attribute retrieval service (included with Tivoli Access Manager WebSEAL), then a custom protocol plug-in must be created.

Creating the protocol plug-in

Custom protocols are written as Java classes that extend the public class FixedProviderProtocol, and must implement the following three abstract methods:

- public ProtocolInitStatus initialize()
- public ProtocolRunStatus run()
- public ProtocolShutdownStatus shutdown()

The initialize() method is called once, to initialize the protocol during the execution of the "initialize" method of the attribute retrieval service. For example, this method can be responsible for establishing a connection to a remote database or profiling service.

The run() method is called (by the "getEntitlement" method of the attribute retrieval service) each time a request is made for a container that must be retrieved by this protocol. This method must retrieve the requested container (or containers) specified by the _container_descriptors member variable of the client class’ HashMap. This container can be obtained using the elements() method of the client class.

The client class’ addContainer() method is then used to add the retrieved container (or containers) to the client class’ _session. How, and from where, the protocol acquires the container is specific to the individual protocol.

The shutdown() method is called once to shutdown the protocol during the execution of the "shutdown" method of the attribute retrieval service. For example, this method can be responsible for closing the connections to remote databases or profiling services that were opened during the "initialize" method.

The following resources are available to assist in creating a custom protocol plug-in:

- Attribute retrieval service class documentation
  /opt/pdwebars/amwebars_class_doc.zip
- Example protocol plug-in modules (Java)
/opt/pdwebars/protocol_plugin/exampleProtocol.java

- Compiled (built) version of the example module
  
/opt/pdwebars/protocol_plugin/exampleProtocol.class

- README file, that explains how to customize and compile the example code
  
/opt/pdwebars/protocol_plugin/README
Appendix A. WebSEAL configuration file reference

The operation of the WebSEAL server is controlled through the use of the WebSEAL configuration file. The file contains sections that control specific portions of WebSEAL functionality. Each section contains further divisions called stanzas.

Stanza labels appear within brackets, such as: [stanza_name]. For example, the [ss1] stanza defines the SSL configuration settings for use by the WebSEAL server.

Each stanza in a Tivoli Access Manager configuration file contains one or more stanza entries. A stanza entry consists of a key value pair, which contain information that is expressed as a paired set of parameters. Each stanza entry has the following format:

key = value

The initial installation of WebSEAL establishes many of the default values. Some values are static and will never change; other values can be modified to customize server functionality and performance.

Configuration file name and location

A unique WebSEAL configuration file is created for each WebSEAL server instance. The name of the configuration file includes the instance name. The format is:

/opt/pdweb/etc/webseald-instance_name.conf

The first WebSEAL server instance installed on a computer has an instance_name of default. The administrator has the option of changing this name during the server configuration. When the administrator accepts the default name, the configuration file is named as follows:

UNIX
/opt/pdweb/etc/webseald-default.conf

Windows
C:\Program Files\Tivoli\PDWeb\etc\webseald-default.conf

When additional WebSEAL server instances are configured, the administrator specifies the instance_name. This method for specifying the name is dependent on the configuration utility method, either as a field entry in the pdconfig GUI or as a command line argument to amwebcfg

- Interactive installation: As a GUI field entry in pdconfig.
- Command line installation: As a command line option to amwebcfg.
- Silent installation: As an entry in a response file used by amwebcfg.

In all cases, the configuration utility uses the entered instance_name to name the new WebSEAL configuration file. For example, if you name the new server instance webseal2, the following configuration file is created:

UNIX
/opt/pdweb/etc/webseald-webseal2.conf
The valid characters for a WebSEAL server instance are as follows:

- alphanumeric characters ([A-Z][a-z][0–9])
- underscore (_)
- hyphen (-)
- period (.)

No other characters are valid. Names must not exceed 20 characters in length.

For more information on WebSEAL server instance configuration, see “Server instance configuration” on page 16.

Guidelines for configuring stanzas

These guidelines are provided to help you make changes to the Tivoli Access Manager configuration files. The guidelines are divided into these types:

- General
- Default settings
- Strings
- Defined strings
- Lists
- File names
- Integers
- Boolean values

For instructions, see “Change configuration settings” on page 381.

General guidelines

Use the following general guidelines when making changes to the configuration settings:

- There is no order dependency or location dependency for stanzas in any configuration file.
- Stanza entries are marked as required or optional. When an entry is required, WebSEAL requires that the entry contain a valid key and value.
- Do not change the names of the keys in the configuration files. Changing the key might cause unpredictable results for the servers.

Note: WebSEAL supports some stanza entries that do not rely upon defined key names. See “Lists” on page 379.

- Capitalization of the keys is not important. For example, you can use UseSSL or useSSL.
- Spaces are not allowed for names of keys.
- For the key value pair format of key = value, the spaces surrounding the equal sign (=) are not required but they are recommended.
- WebSEAL ignores nonprintable characters (such as tabs, carriage returns, and line feeds) that occur at the end of a stanza entry. Nonprintable characters are ASCII characters with a decimal value less than 32.
• Entries in the configuration file should be ASCII characters. WebSEAL does not expect to read non-ASCII characters, such as those supported by multi-byte locales, in the values for configuration file entries.

**Default values**

• Many values are created or modified only by using configuration programs. You should not manually edit these stanzas or values.
• Some values are filled in automatically during WebSEAL configuration. These values are needed for the initialization of the server after the configuration.
• The default values for a stanza entry might be different, depending on the server configuration. Some stanza entries are not applicable to certain configurations and are omitted from the default configuration file for this server.

**Strings**

Some values accept a string value. When you manually edit the configuration file, use the following guidelines to change configuration settings that require a string:

• String values are expected to be characters that are part of the local codeset.
• Some WebSEAL strings impose additional or different restrictions on the set of allowable string characters. For example, many strings are restricted to ASCII characters. The restrictions applicable to each string are listed under the appropriate stanza entry discussion later in this chapter. Consult each stanza entry description for any additional restrictions.
• Double quotation marks are sometimes, but not always, required if spaces or more than one word are used for values. Refer to the descriptions or examples for each stanza entry when in doubt.
• The minimum and maximum lengths of user registry-related string values, if there are limits, are imposed by the underlying registry. For example, for Active Directory the maximum length is 256 alphanumeric characters.

**Defined strings**

Some values accept a string value, but the value must be one of a set of defined strings. When you manually edit the configuration file, make sure that the string value you type matches one of the valid defined strings values.

For example, the [ba] stanza in the authentication mechanisms section contains the following entry:

```plaintext
ba-auth = { none | http | https | both }
```

WebSEAL expects the value of ba-auth to be either none or http or https or both. Any other value is invalid and will result in an error.

**Lists**

The WebSEAL configuration file contains some stanzas that use stanza entries that do not have defined key names. The contents of these stanzas are called *lists*. Lists are configurable, and are specified by the administrator. Some lists are included by default and can be appended by the administrator. Other lists are empty by default and can be created by the administrator.

Lists are used by WebSEAL for a number of purposes. WebSEAL filters incoming data based on document type, event handler type, MIME type, and header request data. WebSEAL also uses lists for content management in a number of areas,
including tracking CGI types, compiling lists of icons for indexing, defining and sizing content caches, defining MIME types, and listing content encodings.

Examples of stanzas that use lists are [filter-url], [filter-events], and [content-mime-types]. There are additional stanzas that support lists. Consult the stanza descriptions in this chapter for complete information.

**File names**

Some values are file names. For each stanza entry that expects a file name as a value, the description of the stanza entry specifies which of the following constructs are valid:

- **File name**
  No directory path included.

- **Relative file name**
  A directory path is allowed, but not mandatory.

These files typically are expected to be relative to the location of a standard WebSEAL directory, such as the WebSEAL server-root or the document root, doc-root. The stanza entry for each relative lists the root directory to which the file name is relative.

- **Fully qualified (absolute) path**
  An absolute directory path is required.

**Note:** Some stanza entries allow more than one of the above choices.

The set of characters permitted in a file name is determined by the file system and by the local codeset. WebSEAL does not impose additional limitations on the set of allowable characters in a file name. For Windows, file names cannot have these characters: a backslash (\), a colon (:), a question mark (?), or double quotation marks (").

**Integers**

Many stanza entries expect the value for the entry to be expressed as an integer.

- **Stanza entries that take an integer value** expect integer values within a valid range. The range is described in terms of a minimum value and a maximum value.
  For example, in the [logging] stanza, the logflush stanza entry has a minimum value of 1 second and a maximum value of 600 seconds.

- **For some entries, the integer value must be positive, and the minimum value is 1.** For other entries, a minimum integer value of 0 is allowed.
  Use caution when setting an integer value to 0. For some entries, an integer value of 0 disables the feature controlled by the stanza entry. For example, in the [junction] stanza, the entry max-webseal-header-size limits the maximum size, in bytes, of HTTP headers generated by the WebSEAL server. A value of zero (0) disables this support for limiting header size.

- **For some entries requiring integer values, WebSEAL does not impose an upper limit on the maximum number allowed.** For example, there is typically no maximum for timeout-related values, such as client-connect-timeout in the [server] stanza.
  For this type of entry, the maximum number is limited only by the size of memory allocated for an integer data type. This number can vary based on operating system type. For systems that allocate 4 bytes for an integer, this value is 2147483647.
However, an administrator should not set integer values to the maximum size of
allocated memory. Instead, administrators should determine logical values for
each setting based on the feature that is being configured.

**Boolean values**

Many stanza entries represent a boolean value. WebSEAL recognizes the boolean
values yes and no.

Some of the entries in webseald.conf are read by other servers and utilities. For
example, many entries in the [ldap] stanza are read by the LDAP client. Some of
these other programs recognize additional boolean characters:
- yes or true
- no or false

The recognized boolean entries are listed for each stanza entry. Refer to the
individual descriptions to determine when other values, such as true or false, are
also recognized.

**Change configuration settings**

To change a configuration setting, do the following:

1. Make a backup copy of the configuration file you plan to modify.
   
   This allows you to return the configuration file to a known working state,
   should you encounter an error later.

2. Stop the WebSEAL server.

3. Make the changes by using an ASCII text editor to edit the configuration file
   (webseald.conf). Save your changes.

4. Restart the WebSEAL server.
The remainder of this appendix contains the following sections and stanzas:

<table>
<thead>
<tr>
<th>Section</th>
<th>Stanzas</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Server configuration” on page 384</td>
<td>[server] [process-root-filter]</td>
</tr>
<tr>
<td>“User registry” on page 393</td>
<td>[ldap]</td>
</tr>
<tr>
<td>“LDAP” on page 393</td>
<td>[ldap]</td>
</tr>
<tr>
<td>“Active Directory” on page 400</td>
<td>[uraf-ad]</td>
</tr>
<tr>
<td>“IBM Lotus Domino” on page 403</td>
<td>[uraf-domino]</td>
</tr>
<tr>
<td>“Secure Socket Layer” on page 406</td>
<td>[ssl]</td>
</tr>
<tr>
<td>“Authentication” on page 413</td>
<td>[ssl]</td>
</tr>
<tr>
<td>“Authentication methods” on page 414</td>
<td>[ba] [forms] [spnego] [token] [certificate] [http-headers] [auth-headers] [ipaddr] [authentication-levels] [step-up] [mpa]</td>
</tr>
<tr>
<td>“Authentication libraries” on page 419</td>
<td>[authentication-mechanisms]</td>
</tr>
<tr>
<td>“Reauthentication” on page 423</td>
<td>[reauthentication]</td>
</tr>
<tr>
<td>“Authentication failover” on page 424</td>
<td>[failover] [failover-attributes] [failover-add-attributes] [failover-restore-attributes]</td>
</tr>
<tr>
<td>“Cross-domain single sign-on” on page 428</td>
<td>[cdsso] [cdsso-peers]</td>
</tr>
<tr>
<td>“e-community single sign-on” on page 431</td>
<td>[e-community-sso] [e-community-domain-keys]</td>
</tr>
<tr>
<td>“Quality of protection” on page 435</td>
<td>[ssl-qop] [ssl-qop-mgmt-hosts] [ssl-qop-mgmt-networks] [ssl-qop-mgmt-default]</td>
</tr>
<tr>
<td>“Session” on page 437</td>
<td>[session]</td>
</tr>
<tr>
<td>“Content” on page 440</td>
<td></td>
</tr>
<tr>
<td>Topic</td>
<td>Reference</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Content management</td>
<td>[content]</td>
</tr>
<tr>
<td>Account management</td>
<td>[acnt-mgt]</td>
</tr>
<tr>
<td>Automatic redirect</td>
<td>[content]</td>
</tr>
<tr>
<td>[enable-redirects]</td>
<td></td>
</tr>
<tr>
<td>Local CGI</td>
<td>[cgi]</td>
</tr>
<tr>
<td>[cgi-types]</td>
<td>[cgi-environment-variables]</td>
</tr>
<tr>
<td>Icons</td>
<td>[content-index-icons]</td>
</tr>
<tr>
<td>[icons]</td>
<td></td>
</tr>
<tr>
<td>Content caching</td>
<td>[content-cache]</td>
</tr>
<tr>
<td>Content MIME types</td>
<td>[content-mime-types]</td>
</tr>
<tr>
<td>Content encodings</td>
<td>[content-encodings]</td>
</tr>
<tr>
<td>Junctions</td>
<td>[junction]</td>
</tr>
<tr>
<td>Junction management</td>
<td></td>
</tr>
<tr>
<td>Document filtering</td>
<td>[filter-url]</td>
</tr>
<tr>
<td>Event handler filtering</td>
<td>[filter-events]</td>
</tr>
<tr>
<td>Scheme filtering</td>
<td>[filter-schemes]</td>
</tr>
<tr>
<td>MIME types and header filtering</td>
<td>[filter-content-types]</td>
</tr>
<tr>
<td>[filter-request-headers]</td>
<td></td>
</tr>
<tr>
<td>Script filtering</td>
<td>[script-filtering]</td>
</tr>
<tr>
<td>[preserve-cookie-names]</td>
<td></td>
</tr>
<tr>
<td>Credential refresh</td>
<td>[credential-refresh-attributes]</td>
</tr>
<tr>
<td>Header names</td>
<td>[header-names]</td>
</tr>
<tr>
<td>Global Sign-On cache</td>
<td>[gso-cache]</td>
</tr>
<tr>
<td>Lightweight Third Party Authentication cache</td>
<td>[ltpa-cache]</td>
</tr>
<tr>
<td>Logging</td>
<td>[logging]</td>
</tr>
<tr>
<td>Auditing</td>
<td>[aznapi-configuration]</td>
</tr>
<tr>
<td>Policy database</td>
<td>[aznapi-configuration]</td>
</tr>
<tr>
<td>Entitlement services</td>
<td>[aznapi-entitlements-services]</td>
</tr>
<tr>
<td>Policy server</td>
<td>[policy-director]</td>
</tr>
<tr>
<td>[manager]</td>
<td></td>
</tr>
</tbody>
</table>
Server configuration

- [server]
- [process-root-filter]

<table>
<thead>
<tr>
<th>[server] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>unix-user = <em>user_name</em></td>
</tr>
<tr>
<td>UNIX user account for the WebSEAL server. This must be a valid UNIX user name. It is possible for a UNIX user account and a UNIX group to have the same name.</td>
</tr>
<tr>
<td>The validity of the user name specified depends on the requirements of the UNIX platform. Leading and trailing spaces are removed.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: ivmgr</td>
</tr>
<tr>
<td>Example: unix-user = ivmgr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>unix-group = <em>group_name</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX group account for the WebSEAL server. This must be a valid UNIX group name. It is possible for a UNIX user account and a UNIX group to have the same name.</td>
</tr>
<tr>
<td>The validity of the group name specified depends on the requirements of the UNIX platform. Leading and trailing spaces are removed.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: ivmgr</td>
</tr>
<tr>
<td>Example: unix-group = ivmgr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>unix-pid-file = <em>fully_qualified_path</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Location and name of a file into which WebSEAL will place its process ID (PID). Applies to UNIX and Windows systems. This value is set automatically during WebSEAL configuration. Typically there is no need to change this file name.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Example: unix-pid-file = C:/Program Files/Tivoli/PDWeb/log/webseald.pid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>server-root = <em>fully_qualified_path</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Root directory for the WebSEAL server. This value is set during WebSEAL configuration.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value is operating-system specific:</td>
</tr>
<tr>
<td>- Windows</td>
</tr>
<tr>
<td>C:/Program Files/Tivoli/PDWeb/www-instance_name</td>
</tr>
<tr>
<td>- UNIX</td>
</tr>
<tr>
<td>/opt/pdweb/www-instance_name</td>
</tr>
<tr>
<td>Example: server-root = /opt/pdweb/www</td>
</tr>
</tbody>
</table>

<p>| server-name = <em>WebSEAL_server_instance_name</em> |</p>
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>server-name</code></td>
<td>The WebSEAL server instance name. This value is set by the administrator during WebSEAL configuration. WebSEAL instance names must be alphanumeric. The maximum number of characters allowed is 20.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Example first WebSEAL server with the default instance name accepted, on a host named diamond:</td>
</tr>
<tr>
<td></td>
<td><code>server-name = diamond-default</code></td>
</tr>
<tr>
<td></td>
<td>Example instance WebSEAL server instance, specified as web2, on a host named diamond:</td>
</tr>
<tr>
<td></td>
<td><code>server-name = diamond-web2</code></td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>There is no default value.</td>
</tr>
<tr>
<td></td>
<td>Example: <code>server-name = diamond</code></td>
</tr>
<tr>
<td><code>worker-threads</code></td>
<td>Number of WebSEAL worker threads. The minimum allowed value is 1. The maximum number of threads is based on the number of file descriptors set for WebSEAL at compile time. Note that this number varies per operating system. If the value is set to a number larger than the WebSEAL-determined limit, WebSEAL reduces the value to the acceptable limit and issues a warning message.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Default: 50</td>
</tr>
<tr>
<td></td>
<td>Example: <code>worker-threads = 50</code></td>
</tr>
<tr>
<td></td>
<td>See also the <em>IBM Tivoli Access Manager for e-business Performance Tuning Guide</em>.</td>
</tr>
<tr>
<td><code>client-connect-timeout</code></td>
<td>Initial client connection timeout, in seconds. Must be a positive integer. Other values have unpredictable results and should not be used. Maximum allowed value: 2147483647.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Default value: 120</td>
</tr>
<tr>
<td></td>
<td>Example: <code>client-connect-timeout = 120</code></td>
</tr>
<tr>
<td><code>persistent-con-timeout</code></td>
<td></td>
</tr>
</tbody>
</table>

**Table of Stanza Entries**

- **Server Name**: Must be alphanumeric. Maximum allowed number of characters is 20.
- **Worker Threads**: Minimum allowed value is 1. The maximum number of threads is based on the number of file descriptors set for WebSEAL at compile time. Note that this number varies per operating system. If the value is set to a number larger than the WebSEAL-determined limit, WebSEAL reduces the value to the acceptable limit and issues a warning message.
- **Connect Timeout**: Initial client connection timeout, in seconds. Must be a positive integer. Other values have unpredictable results and should not be used. Maximum allowed value: 2147483647.
**HTTP/1.1 connection timeout**, in seconds. Must be a positive integer. Other values have unpredictable results and should not be used. Maximum allowed value: 2147483647.

This setting affects connections to clients, not to backend server systems.

This stanza entry is required.

Default value: 5

Example: `persistent-con-timeout = 5`

<table>
<thead>
<tr>
<th>Chunk-responses = {yes|no}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables WebSEAL to write chunked data to HTTP/1.1 clients. This can improve performance by allowing connections to be reused even when the exact response length is not known before the response is written.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value is yes</td>
</tr>
<tr>
<td>Example: <code>chunk-responses = yes</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>https = {yes|no}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies whether HTTPS requests will be accepted by the WebSEAL server. This is a boolean value. Valid values are yes or no. This value is set by the administrator during WebSEAL server configuration.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value.</td>
</tr>
<tr>
<td>Example: <code>https = yes</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>https-port = <code>port_number</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Port on which WebSEAL listens for HTTPS requests. This value is set during WebSEAL configuration. When the default port is already in use, WebSEAL configuration suggests the next available (unused) port number. The administrator can modify this number. Valid values include any port number not already in use on the host.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: 443</td>
</tr>
<tr>
<td>Example: <code>https-port = 443</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>http = {yes|no}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies whether HTTP requests will be accepted by the WebSEAL server. Valid values are yes or no. This value is set by the administrator during WebSEAL server configuration.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value.</td>
</tr>
<tr>
<td>Example: <code>http = yes</code></td>
</tr>
</tbody>
</table>

| http-port = `port_number` |
Port on which WebSEAL listens for HTTPS requests. This value is set during WebSEAL configuration. When the default HTTP port is already in use, WebSEAL configuration suggests the next available (unused) port number. The administrator can modify this number. Valid values include any port number not already in use on the host.

This stanza entry is required.
Default value: 80
Example: http-port = 80

<table>
<thead>
<tr>
<th>max-client-read = number_of_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the maximum number of bytes that WebSEAL will hold in internal buffers while reading from a client. Affects the maximum size of URLs, HTTP Headers, and the size of a request that will be cached. Must be set to at least twice the value of request-body-max-read. Minimum number of bytes: 32678. If the value is set to a number below 32678, the value is ignored and a value of 32678 is used. There is no maximum.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: 32768</td>
</tr>
<tr>
<td>Example: max-client-read = 32768</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>request-body-max-read = number_of_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of bytes to read in as content from the body of POST requests. Used for dynurl, authentication, and request caching. Minimum number of bytes: 512. Maximum number of bytes: 32768. However, the maximum can be increased by raising the value of max-client-read. See “Configuring server-side caching parameters” on page 201.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: 4096</td>
</tr>
<tr>
<td>Example: request-body-max-read = 4096</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>request-max-cache = number_of_bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum amount of data to cache. This is used to cache request data when a user is prompted to authenticate before a request can be fulfilled.</td>
</tr>
<tr>
<td>This value should be a positive integer. If set to zero (0), the user login succeeds but the request fails because WebSEAL cannot cache the request data. There is no maximum value.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: 8192</td>
</tr>
<tr>
<td>Example: request-max-cache = 8192</td>
</tr>
<tr>
<td>See also “Configuring server-side caching parameters” on page 201.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dynurl-map = relative_pathname</th>
</tr>
</thead>
</table>

Appendix A. WebSEAL configuration file reference 387
Location of the file that contains mappings for URLs to protected objects. The path is relative to the value of the server-root key in the [server] stanza.

The administrator can specify an alternate file name, and an alternate directory location. The file name can be any file name that is valid for the operating system file system. On Windows systems, both backslashes (\) and forward slashes (/) are supported in the directory path.

This stanza entry is optional.

Default: None, but usually is configured to lib/dynurl.conf

Example: dynurl-map = lib/dynurl.conf

dynurl-allow-large-posts = {yes|no}

Allows or disallows POST requests larger than the current value for the stanza entry request-body-max-read in the [server] stanza.

When set to no, WebSEAL disallows POST requests with a body larger than request-body-max-read. When set to yes, WebSEAL compares only up to request-body-max-read bytes of POST request to the URL mappings contained in dynurl configuration file (dynurl.conf).

This stanza entry is required.

Default value: no

Example: dynurl-allow-large-posts = no

decode-query = {yes|no}

When decode-query is set to yes WebSEAL validates the query string in requests according to the utf8-qstring-support-enabled parameter. Otherwise, WebSEAL does not validate the query string. When decode-query is set to no, then dynurl must be disabled.

This stanza entry is required.

Default value: yes

Example: decode-query = yes

utf8-url-support-enabled = {yes|no|auto}
Enable or disable support for UTF-8 encoded characters in URLs.

The following values are valid:

- **yes**
  WebSEAL only recognizes UTF-8 encoding in URLs and the data is used without modification.

- **no**
  WebSEAL does not recognize UTF-8 encoding in URLs. Used for local code page only.

- **auto**
  When set to auto, WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. When encoding is not recognized as UTF-8, WebSEAL processes the coding as non-UTF-8.

This stanza entry is required.

Default value: yes

Example: utf8-url-support-enabled = yes

See also "Multi-locale support with UTF-8" on page 44.

| utf8-form-support-enabled = {yes|no|auto} |
|------------------------------------------|
| Enable or disable support for UTF-8 encoded characters in data read from forms, such as user names during login. |
| The following values are valid: |
| - **yes**
  WebSEAL only recognizes UTF-8 encoding in forms and the data is used without modification. |
| - **no**
  WebSEAL does not recognize UTF-8 encoding in forms. Used for local code page only. |
| - **auto**
  When set to auto, WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. When encoding is not recognized as UTF-8, WebSEAL processes the coding as non-UTF-8. |
| This stanza entry is required. |
| Default value: yes |
| Example: utf8-url-support-enabled = yes |
| See also "Multi-locale support with UTF-8" on page 44. |

| utf8-qstring-support-enabled = {yes|no|auto} |
|---------------------------------------------|
| Enable or disable support for UTF-8 encoded characters in data read from forms, such as user names during login. |
| The following values are valid: |
| - **yes**
  WebSEAL only recognizes UTF-8 encoding in forms and the data is used without modification. |
| - **no**
  WebSEAL does not recognize UTF-8 encoding in forms. Used for local code page only. |
| - **auto**
  When set to auto, WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. When encoding is not recognized as UTF-8, WebSEAL processes the coding as non-UTF-8. |
| This stanza entry is required. |
| Default value: yes |
| Example: utf8-url-support-enabled = yes |
| See also "Multi-locale support with UTF-8" on page 44. |
Enable or disable support for UTF-8 encoded characters in query strings data read from client requests.

The following values are valid:

- **yes**
  WebSEAL only recognizes UTF-8 encoding in strings and the data is used without modification.

- **no**
  WebSEAL does not recognize UTF-8 encoding in strings. Used for local code page only.

- **auto**
  When set to auto, WebSEAL attempts to distinguish between UTF-8 and other forms of language character encoding. When encoding is not recognized as UTF-8, WebSEAL processes the string as non-UTF-8.

This stanza entry is required.

Default value: no

Example: `utf8-qstring-support-enabled = no`

See also: "Multi-locale support with UTF-8" on page 44.

<table>
<thead>
<tr>
<th>Table: double-byte-encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies whether WebSEAL will assume that encoded characters within URLs are always encoded in Unicode, and do not contain UTF-8 characters.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: no</td>
</tr>
<tr>
<td>Example: <code>double-byte-encoding = no</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: suppress-server-identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppresses the identity of the WebSEAL server from HTTP responses. These responses normally include the line: <strong>Server: WebSEAL/version_number</strong></td>
</tr>
<tr>
<td>Setting this value to yes deletes the above line from the server response.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: no</td>
</tr>
<tr>
<td>Example: <code>suppress-server-identity = no</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: pre-410-compatible-tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSEAL supports a common method of generating tokens for cross-domain single sign-on, failover, and e-community single sign-on. The security of these tokens was increased for Version 4.1. This increase is not backwards compatible with previous versions of WebSEAL. When the Tivoli Access Manager deployment includes multiple WebSEAL servers, and some of the WebSEAL servers are Version 3.9 or prior, set this value to yes.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: no</td>
</tr>
<tr>
<td>Example: <code>pre-410-compatible-tokens = no</code></td>
</tr>
<tr>
<td>Configuration Parameter</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>**pre-510–compatible-tokens = {yes</td>
</tr>
<tr>
<td>**preserve-base-href = {yes</td>
</tr>
<tr>
<td>**http-method-trace-enabled = {yes</td>
</tr>
<tr>
<td>**http-method-trace-enabled-remote = {yes</td>
</tr>
<tr>
<td>**process-root-requests = {never</td>
</tr>
</tbody>
</table>
Specifies how WebSEAL responds to requests for resources located at the root ("/") junction. Valid values are:

- **never**
  Root junction requests are never processed at the root junction.

- **always**
  Always attempt to process requests for the root junction at the root junction first before attempting to use a junction mapping mechanism.

- **filter**
  Examine all root junction requests to determine whether they start with the patterns specified in the [process-root-filter] stanza.

This stanza entry is required.

Default value: always

Example: `process-root-requests = always`

See also:

- "Processing root junction requests” on page 294

### process-root-filter stanza

**root = pattern**

This stanza is used only when `process-root-requests = filter`. Values for `pattern` must be standard WebSEAL wildcard patterns.

Entries in this stanza are optional.

Default entries:

```
root = /index.html
root = /cgi-bin*
```
User registry

This section contains the following topics and stanzas:

<table>
<thead>
<tr>
<th>Section</th>
<th>Stanzas</th>
</tr>
</thead>
<tbody>
<tr>
<td>“LDAP”</td>
<td>[ldap]</td>
</tr>
<tr>
<td>“Active Directory” on page 400</td>
<td>[uraf-ad]</td>
</tr>
<tr>
<td>“IBM Lotus Domino” on page 403</td>
<td>[uraf-domino]</td>
</tr>
</tbody>
</table>

**LDAP**

**[ldap] stanza**

<table>
<thead>
<tr>
<th>ldap-server-config = fully_qualified_path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location of the 1dap.conf configuration file, represented as a fully qualified path.</td>
</tr>
<tr>
<td>The fully_qualified_path value must be an alphanumeric string.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value is based on the operating system type.</td>
</tr>
</tbody>
</table>

Example for UNIX: ldap-server-config = /opt/PolicyDirector/etc/ldap.conf

| enabled = {yes|true|no|false}                    |
|-----------------------------------------------|
| Indicates whether or not LDAP is being used as the user registry. |
| Valid values are:                              |
| yes | true | Enable LDAP user registry support. |
| no  | false| Disable LDAP user registry support. Anything other than yes|true, including a blank value, is interpreted as no|false, |
| This stanza entry is required when LDAP is the user registry. |

The default value for a WebSEAL installation is determined from the configuration of the Tivoli Access Manager runtime. The Tivoli Access Manager runtime component is a prerequisite for WebSEAL.

Example: enabled = yes

| host = host_name |
Host name of the LDAP server.

The WebSEAL configuration utility gets the `host_name` value from the `pd.conf` file. The `pd.conf` file is created when the Tivoli Access Manager runtime component is configured on the machine. The Tivoli Access Manager runtime component is a prerequisite for WebSEAL.

Valid values for `host_name` include any valid IP host name. The `host_name` does not have to be a fully qualified domain name.

This stanza entry is required.

The default value is the value entered by the administrator during the configuration of the Tivoli Access Manager runtime component.

Examples:

```plaintext
host = surf
host = surf.santacruz.ibm.com
```

<table>
<thead>
<tr>
<th><strong>port</strong> = <em>port_number</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of the TCP/IP port used for communicating with the LDAP server. Note that this is not for SSL communication.</td>
</tr>
<tr>
<td>A valid port number is any positive integer that is allowed by TCP/IP and that is not currently being used by another application.</td>
</tr>
<tr>
<td>This stanza entry is required when LDAP is enabled.</td>
</tr>
<tr>
<td>Default value: 389.</td>
</tr>
<tr>
<td>Example: <code>port = 389</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>bind-dn</strong> = <em>LDAP_dn</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP user distinguished name (DN) that is used when binding (or signing on) to the LDAP server. This is the name that represents the WebSEAL server daemon.</td>
</tr>
<tr>
<td>This stanza entry is required when LDAP is enabled.</td>
</tr>
<tr>
<td>The default value is built by combining the daemon name <code>webseald</code> with the <code>host_name</code> that was specified by the administrator during the configuration of the Tivoli Access Manager runtime component.</td>
</tr>
<tr>
<td>Example: <code>bind-dn = cn=webseald/surf,cn=SecurityDaemons,secAuthority=Default</code></td>
</tr>
<tr>
<td>See also the <em>IBM Tivoli Access Manager for e-business Performance Tuning Guide</em>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>bind-pwd</strong> = <em>LDAP_password</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Password for the LDAP user distinguished name declared in the <code>bind-dn</code> stanza entry.</td>
</tr>
<tr>
<td>This stanza entry is required when LDAP is enabled.</td>
</tr>
<tr>
<td>The default value of this stanza entry is set during WebSEAL configuration. The WebSEAL configuration reads the <code>LDAP_password</code> that was specified by the administrator during the configuration of the Tivoli Access Manager runtime component. This value is read from the Tivoli Access Manager configuration file, <code>pd.conf</code>.</td>
</tr>
<tr>
<td>Example: <code>bind-pwd = zs77WkoLSZn1rKrL</code></td>
</tr>
</tbody>
</table>

| **cache-enabled** = {yes|true|no|false} |
Enables or disable LDAP client-side caching. Caching is used to improve performance for similar LDAP queries.

Valid values are:

- **yes|true**
  - Enable LDAP client-side caching.

- **no|false**
  - Disable LDAP client-side caching.

Anything other than yes|true, including a blank value, is interpreted as no|false.

This stanza entry is optional.

Default value: disabled. When no value is specified (empty value), client-side caching is disabled.

Example: cache-enabled = yes

See also the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

### cache-user-size = number

Specifies the number of entries in the LDAP user cache.

This entry is used only when cache-enabled = {yes|true}.

This stanza entry is optional.

There is no default value, but when not set the default value used is 256.

Example: cache-user-size = 256

### cache-group-size = number

Specifies the number of entries in the LDAP group cache.

This entry is used only when cache-enabled = {yes|true}.

This stanza entry is optional.

There is no default value, but when not set the default value used is 64.

Example: cache-group-size = 64

### cache-policy-size = number

Specifies the number of entries in the LDAP policy cache.

This entry is used only when cache-enabled = {yes|true}.

This stanza entry is optional.

There is no default value, but when not set the default value used is 20.

Example: cache-policy-size = 20

### cache-user-expire-time = number_of_seconds

Specifies the amount of time to elapse before a user entry in the cache is discarded.

This entry is used only when cache-enabled = {yes|true}.

This stanza entry is optional.

There is no default value, but when not set the default value used is 30 seconds.

Example: cache-user-expire-time = 30
### cache-group-expire-time = *number_of_seconds*

Specifies the amount of time to elapse before a group entry in the cache is discarded.

This entry is used only when `cache-enabled = {yes|true}`.

This stanza entry is optional.

There is no default value, but when not set the default value used is 300 seconds.

Example: `cache-group-expire-time = 300`

### cache-policy-expire-time = *number_of_seconds*

Specifies the amount of time to elapse before a policy entry in the cache is discarded.

This entry is used only when `cache-enabled = {yes|true}`.

This stanza entry is optional.

There is no default value, but when not set the default value used is 30 seconds.

Example: `cache-policy-expire-time = 30`

### cache-group-membership = {yes|no}

Indicates whether group membership information should be cached.

This entry is used only when `cache-enabled = {yes|true}`

This stanza entry is optional.

There is no default value, but when not set the group information is cached.

Example: `cache-group-membership = yes`

### cache-use-user-cache = {yes|no}

Indicates whether to use the user cache information or not.

This entry is used only when `cache-enabled = {yes|true}`

This stanza entry is optional.

There is no default value, but when not set the user cache information is used.

Example: `cache-use-user-cache = yes`

### prefer-readwrite-server = {yes|true|no|false}


<table>
<thead>
<tr>
<th>Allow or disallows the client to question the Read/Write LDAP server before querying any replica Read-only servers configured in the domain.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid values are:</td>
</tr>
<tr>
<td>*yes</td>
</tr>
<tr>
<td>Enable the choice.</td>
</tr>
<tr>
<td>*no</td>
</tr>
<tr>
<td>Disable the choice.</td>
</tr>
<tr>
<td>Anything other than *yes</td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td>If this value is not specified, the <code>prefer-readwrite-server</code> option is disabled.</td>
</tr>
<tr>
<td>Example: <code>prefer-readwrite-server = no</code></td>
</tr>
</tbody>
</table>

### ssl-enabled = {yes|true|no|false}

Enables or disables SSL communication between WebSEAL and the LDAP server.

Valid values are:

- *yes|true*
  - Enable SSL communication.
- *no|false*
  - Disable SSL communication.

This stanza entry is optional.

SSL communication is disabled by default. During WebSEAL server configuration, the WebSEAL administrator can choose to enable it.

Example: `ssl-enabled = yes`

### ssl-keyfile = fully_qualified_path

SSL key file name and location. The SSL key file handles certificates that are used in LDAP communication.

The file name must be a fully qualified path. The file name can be any arbitrary choice, but the extension is usually `.kdb`.

This stanza entry is required when SSL communication is enabled, as specified in the `ssl-enabled` stanza entry. The WebSEAL administrator specifies this file name during WebSEAL configuration.

Example for UNIX:

```
ssl-keyfile = /var/pdweb/keytabs/webseald.kdb
```

Example for Windows:

```
ssl-keyfile = c:\keytabs\pd_ldapkey.kdb
```

For information on SSL keyfile management, see "Managing client-side and server-side certificates" on page 228.

### ssl-keyfile-label = keyLabel
String that specifies the key label of the client personal certificate within the SSL key file. This key label is used to identify the client certificate that is presented to the LDAP server.

This stanza entry is optional. A label is not required when one of the certificates in the keyfile has been identified as the default certificate. The decision whether to identify a certificate as the default was made previously by the LDAP administrator when configuring the LDAP server. The WebSEAL configuration utility prompts the WebSEAL administrator to supply a label. When the administrator knows that the certificate contained in the keyfile is the default certificate, the administrator does not have to specify a label.

Example: ssl-keyfile-dn = "PD_LDAP"

ssl-keyfile-pwd = password

Password to access the SSL key file.

The password associated with the default SSL keyfile is gsk4ikm

Required only when SSL communication between WebSEAL and LDAP is enabled, as specified in the ssl-enabled stanza entry. The WebSEAL administrator specifies this password during WebSEAL configuration.

Example: ssl-keyfile-pwd = gsk4ikm

auth-using-compare = {yes|true|no|false}

Enables or disables authentication using password comparison. When disabled, authentication using LDAP bind is performed.

For those LDAP servers that allow it, a compare operation might perform faster than a bind operation.

Valid values are:

yes|true

A password compare operation is used to authenticate LDAP users.

no|false

A bind operation is used to authenticate LDAP users.

This stanza entry is optional.

The default value, when LDAP is enabled, is yes.

Example: auth-using-compare = yes

See also the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

default-policy-override-support = {yes|true|no|false}
Indicates whether default policy overrides user level policy during LDAP searches. When this stanza entry is set to yes, only the default policy is checked.

Valid values are:

- **yes | true**
  
  User policy support is disabled and only the global (default) policy is checked. This option allows the user policy to be ignored, even when it is specified.

- **no | false**
  
  User policy support is enabled. When a user policy is specified by the administrator, it overrides the global policy.

This stanza entry is optional.

By default, the value is not specified during WebSEAL configuration. When the value is not specified, the default behavior is enable user policy support. This is equivalent to setting this stanza entry to no.

Example: default-policy-override-support = yes

**user-and-group-in-same-suffix = \{yes\true\no\false\}**

Indicates whether the groups, in which a user is a member, are defined in the same LDAP suffix as the user definition.

When a user is authenticated, the groups in which the user is a member must be determined in order to build a credential. Normally, all LDAP suffixes are searched to locate the groups of which the user is a member.

Valid values are:

- **yes | true**
  
  The groups are assumed to be defined in same LDAP suffix as the user definition. Only that suffix is searched for group membership. This behavior can improve the performance of group lookup because only a single suffix is searched for group membership. This option should only be specified if group definitions are restricted to the same suffix as the user definition.

- **no | false**
  
  The groups might be defined in any LDAP suffix.

This stanza entry is optional.

The value is not specified by default during WebSEAL configuration. When the value is not specified, the default value is no.

Example: user-and-group-in-same-suffix = yes

See also the *IBM Tivoli Access Manager for e-business Performance Tuning Guide.*
### Active Directory

#### [uraf-ad] stanza

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ad-server-config = <em>fully_qualified_path</em></td>
<td>Location and file name of the Active Directory registry activedir.conf configuration file. This value is generated, but it can be changed. The <em>fully_qualified_path</em> value represents an alphanumeric, non-case sensitive string. The maximum string length for the Active Directory user registry is 256 alphanumeric characters. This stanza entry is required when your user registry is Microsoft Active Directory and is required only for configuration files other than activedir.conf. Default value for UNIX: /opt/PolicyDirector/etc/activedir.conf Default value for Windows: c:\Program files\tivoli\Policy Director\etc\activedir.conf Example for UNIX: ad-server-config = /opt/PolicyDirector/etc/activedir.conf</td>
</tr>
<tr>
<td>enabled = {yes</td>
<td>no}</td>
</tr>
<tr>
<td>multi-domain = {true</td>
<td>false}</td>
</tr>
<tr>
<td>hostname = hostname</td>
<td></td>
</tr>
</tbody>
</table>
Active Directory domain name system (DNS) host name. The value is filled in automatically, based on information supplied during the runtime configuration. The *hostname* is an alphanumeric, non-case sensitive string. The dot (.) cannot be the last character of the host name. The maximum string length for the Active Directory user registry is 256 alphanumeric characters.

This stanza entry is required when your user registry is Microsoft Active Directory.

There is no default value.

Example: hostname = adserver.tivoli.com

<table>
<thead>
<tr>
<th>domain = root_domain_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Directory root (primary) domain. The value is filled in automatically, based on information supplied during the runtime configuration. The <em>root_domain_name</em> is an alphanumeric, non-case sensitive string. The maximum length for the domain name is user registry dependent. For Active Directory that maximum length is 256 alphanumeric characters.</td>
</tr>
<tr>
<td>This stanza entry is required when multi-domain = true.</td>
</tr>
<tr>
<td>There is no default behavior.</td>
</tr>
<tr>
<td>Example: domain = dc=tivoli,dc=com</td>
</tr>
</tbody>
</table>

| useEncryption = {true|false} |
|-----------------------------|
| Indication of whether encryption communication to Active Directory is being used. This value is filled in automatically, based on information supplied during the runtime configuration. |
| Valid values: |
| true | Enables encryption communication. |
| false | Disables encryption communication. |
| This stanza entry is required when your user registry is Microsoft Active Directory. |
| There is no default behavior. |
| Example: useEncryption = false |

<table>
<thead>
<tr>
<th>bind-id = ad_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Directory administrator or user log-in identity that is used to bind (sign on) to the registry server. If the ID belongs to a user rather than an administrator, the Active Directory user must have enough privileges to update and modify data in the user registry.</td>
</tr>
<tr>
<td>The <em>ad_id</em> value is an alphanumeric, non-case sensitive string. The minimum and maximum lengths of the ID, if there are limits, are imposed by the underlying registry. For Active Directory the maximum length is 256 alphanumeric characters.</td>
</tr>
<tr>
<td>This value is filled in automatically, based on information supplied during server configuration. Whenever you change this value after the configuration is completed, a conflict might occur.</td>
</tr>
<tr>
<td>This stanza entry is required when your user registry is Microsoft Active Directory.</td>
</tr>
<tr>
<td>The default value is generated; do not change it.</td>
</tr>
<tr>
<td>Example: bind-id = adpdadmin</td>
</tr>
<tr>
<td>bind-pwd</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td><strong>Encoded administrator log-in password</strong> that is used to bind (or sign on) to the Active Directory registry server. Additional password requirements can be specified for Tivoli Access Manager after the product is installed. However, the initial password does not necessarily conform to those requirements.</td>
</tr>
<tr>
<td>The <code>admin_password</code> value is filled in automatically, based on information that is supplied during server configuration. The password is an encrypted string.</td>
</tr>
<tr>
<td>This stanza entry is required when your user registry is Microsoft Active Directory.</td>
</tr>
<tr>
<td>The default value is generated; do not change it.</td>
</tr>
<tr>
<td>Example: <code>bind-pwd = MyADbindPwd</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dnforpd</th>
<th>ad_dn</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distinguished name</strong> that is used by Active Directory to store Tivoli Access Manager data. The <code>ad_dn</code> value is an alphanumeric, non-case sensitive string. The minimum and maximum lengths of the distinguished name, if there are limits, are imposed by the underlying registry. For Active Directory the maximum length is 256 alphanumeric characters.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is required when your user registry is Microsoft Active Directory.</td>
<td></td>
</tr>
<tr>
<td>The default value is generated; do not change it.</td>
<td></td>
</tr>
<tr>
<td>Example: <code>dnforpd = dc=child2,dc=com</code></td>
<td></td>
</tr>
</tbody>
</table>
### IBM Lotus Domino

#### [uraf-domino] stanza

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domino-server-config = fully_qualified_path</td>
<td>Name and location of the Lotus Domino registry <code>domino.conf</code> configuration file. The <code>fully_qualified_path</code> represents an alphanumeric string. This stanza entry is required when your user registry is Lotus Domino and is required only for configuration files other than <code>domino.conf</code>. Default value for UNIX: <code>/opt/PolicyDirector/etc/domino.conf</code> Default value for Windows: <code>c:\Program files\tivoli\Policy Director\etc\domino.conf</code> Example for Windows: <code>domino-server-config = c:\Program files\tivoli\Policy Director\etc\domino.conf</code></td>
</tr>
<tr>
<td>enabled = (yes</td>
<td>no)</td>
</tr>
<tr>
<td>server = server_name</td>
<td>Name of the Lotus Domino™ server. The <code>server_name</code> value represents an alphanumeric, non-case sensitive string. The minimum and maximum lengths of the name are imposed by the underlying registry. This stanza entry is required when your user registry is Lotus Domino. There is no default value. Example: server = grizzly/Austin/IBM Where grizzly is the Domino server machine host name and the remainder is the Domino domain name.</td>
</tr>
<tr>
<td>hostname = hostname</td>
<td>Lotus Domino server TCP/IP host name. The <code>hostname</code> value is manually input during configuration. The <code>hostname</code> value must be an alphanumeric, non-case sensitive string. The format is the same as a typical TCP/IP host name. This stanza entry is required when your user registry is Lotus Domino. There is no default value. Example: hostname = myhost.austin.ibm.com</td>
</tr>
<tr>
<td>LDAPPort = port_number</td>
<td></td>
</tr>
</tbody>
</table>

---

Appendix A. WebSEAL configuration file reference 403
LDAP port number for the Lotus Domino server. The *port_number* value is manually input during configuration. A valid port number is any positive number that is allowed by TCP/IP and that is not currently being used by another application.

This stanza entry is required.

Default value: 389

Example: LDAPPort = 389

| UseSSL = {yes|no} |
|------------------|
| Indication of whether to use SSL. |
| Value values: |
| yes | Specifies that you want to use SSL. |
| no | Specifies that you do not want to use SSL. |

This stanza entry is required.

Default value: yes

Example: UseSSL = no

<table>
<thead>
<tr>
<th>keyfile = <em>filename</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL key file name and location. Use the SSL key file to handle certificates that are used in LDAP communication. The <em>filename</em> is an alphanumeric, non-case sensitive string that must conform to the underlying (Windows) file system naming convention. The file must be an existing file on the client machine. The file type can be anything but the extension is usually .kdb.</td>
</tr>
</tbody>
</table>

This stanza entry is required when UseSSL = yes

The default value is server-dependent.

Example for Windows: keyfile = C:/pd/keytab/ivacld.kdb

<table>
<thead>
<tr>
<th>KeyFile_PW = <em>password</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Password for the SSL key file.</td>
</tr>
</tbody>
</table>

This stanza entry is required when UseSSL = yes

The default value is server-dependent.

Example: KeyFile_PW = mykeyfilepw

<table>
<thead>
<tr>
<th>KeyFile_DN = <em>cert_label</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinguished name (DN) of the SSL key file private key. The <em>cert_label</em> value is an alphanumeric string that represents the certificate label of the client personal certificate within the SSL key file. This key label is used to identify the client certificate that is presented to the Domino server.</td>
</tr>
</tbody>
</table>

This stanza entry is required when UseSSL = yes

There is no default value.

| password = *password* |
**Domino server password**

That is used to bind (or sign on) to the Domino registry server. The password is an encrypted string.

This stanza entry is required when enabled = yes

The value is generated; do not change it.

Example: `password = myEncryptedSrvrPwd`

<table>
<thead>
<tr>
<th><strong>NAB</strong> = names.nsf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lotus Domino Name and Address Book (NAB) database. The names.nsf database is set at configuration time and cannot be changed.</td>
</tr>
<tr>
<td>This stanza entry is required when enabled = yes</td>
</tr>
<tr>
<td>Default value: names.nsf</td>
</tr>
<tr>
<td>Example: NAB = names.nsf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PDM</strong> = nsf_filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tivoli Access Manager meta-data database. The nsf_filename represents a Domino database file name. The file name conforms to the underlying operating system file naming conventions of the Domino server. The recommended file extension is .nsf. This file is created on the Domino server during configuration.</td>
</tr>
<tr>
<td>This stanza entry is required when enabled = yes</td>
</tr>
<tr>
<td>Default value: PDMdata.nsf</td>
</tr>
<tr>
<td>Example: PDM = PDMdata.nsf</td>
</tr>
</tbody>
</table>
## [ssl] stanza

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webseal-cert-keyfile = <em>fully_qualified_path</em></td>
<td>Path name to the WebSEAL certificate keyfile. This is the certificate that WebSEAL exchanges with browsers when negotiating SSL sessions. This stanza entry is required. This path is set during WebSEAL configuration. The path consists of the WebSEAL installation directory plus: <em>www-instance_name/certs/pdsrv.kdb</em>. Example: <code>webseal-cert-keyfile = C:/Program Files/Tivoli/PDWeb/www-web1/certs/pdsrv.kdb</code> This path is typically not modified by the WebSEAL administrator after initial WebSEAL configuration.</td>
</tr>
<tr>
<td>webseal-cert-keyfile-stash = <em>fully_qualified_path</em></td>
<td>Name of the file containing an obfuscated version of the password used to protect private keys in the keyfile. This stanza entry is optional. This path is set during WebSEAL configuration. The path consists of the WebSEAL installation directory plus: <em>www-instance_name/certs/pdsrv.sth</em>. Example: <code>webseal-cert-keyfile-stash = C:/Program Files/Tivoli/PDWeb/www-web1/certs/pdsrv.sth</code></td>
</tr>
<tr>
<td>webseal-cert-keyfile-pwd = <em>password</em></td>
<td>Password used to protect private keys in WebSEAL certificate file. When this stanza entry is assigned a value, that value is used instead of any password that is contained in the stash file specified by <code>webseal-cert-keyfile-stash</code>. This stanza entry is optional. This stanza entry stores the password in plain text. For optimum security practice, use of the stash file is recommended. There is no default value. Example: <code>webseal-cert-keyfile-pwd = j73R45huu</code></td>
</tr>
<tr>
<td>webseal-cert-keyfile-label = <em>label_name</em></td>
<td>String specifying a label to use for WebSEAL certificate keyfile. When this is not specified, the default label is used. This stanza entry is optional, but is set by default during WebSEAL configuration. Default value: <code>WebSEAL-Test-Only</code> Example: <code>webseal-cert-keyfile-label = WebSEAL-Test-Only</code></td>
</tr>
<tr>
<td>ssl-keyfile = <em>fully_qualified_path</em></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ssl-keyfile</td>
<td>String specifying the path to the keystore WebSEAL uses for communicating with other Tivoli Access Manager servers over SSL. This stanza entry is required. The default value is set during WebSEAL configuration. The WebSEAL installation directory path is combined with the following path: keytabs/webseald.kdb Example: ssl-keyfile = C:/Program Files/Tivoli/PDWeb/keytabs/webseald.kdb This stanza entry is typically modified only by the WebSEAL configuration utility.</td>
</tr>
<tr>
<td>ssl-keyfile-pwd</td>
<td>String containing the password to protect the private keys in the SSL keyfile. When this stanza entry is assigned a value, that value is used instead of any password that is contained in the stash file specified by ssl-keyfile-stash. This stanza entry is optional. This stanza entry stores the password in plain text. For optimum security practice, use of the ssl-keyfile-stash is recommended. There is no default value. Example: ssl-keyfile-pwd = myPassw0rd This stanza entry is typically modified only by the WebSEAL configuration utility.</td>
</tr>
<tr>
<td>ssl-keyfile-stash</td>
<td>Name of the file containing an obfuscated version of the password used to protect private keys in the SSL keyfile. This stanza entry is required. This path is set during WebSEAL configuration. The path consists of the WebSEAL installation directory plus: keytabs/webseald.sth. Example: ssl-keyfile-stash = C:/Program Files/Tivoli/PDWeb/keytabs/webseald.sth This stanza entry is typically modified only by the WebSEAL configuration utility.</td>
</tr>
<tr>
<td>ssl-keyfile-label</td>
<td>String containing a label for the SSL certificate keyfile. When this label is not specified, the default label is used. This stanza entry is optional, but is assigned during WebSEAL configuration. Default value: PD Server Example: ssl-keyfile-label = PD Server This stanza entry is typically modified only by the WebSEAL configuration utility.</td>
</tr>
<tr>
<td>disable-ssl-v2</td>
<td>{yes</td>
</tr>
</tbody>
</table>
### disable-ssl-v2
Disables support for SSL Version 2. Support for SSL V2 is enabled by default. The value yes means support is disabled. The value no means the support is enabled.

This stanza entry is optional. When not specified, the default is no.

Default value: no. The WebSEAL configuration sets this value.

Example: `disable-ssl-v2 = no`

### disable-ssl-v3
Disables support for SSL Version 3. Support for SSL V3 is enabled by default. The value yes means support is disabled. The value no means the support is enabled.

This stanza entry is optional. When not specified, the default is no.

Default value: no. The WebSEAL configuration sets this value.

Example: `disable-ssl-v3 = no`

### disable-tls-v1
Disables support for TLS Version 1. Support for TLS V1 is enabled by default. The value yes means support is disabled. The value no means the support is enabled.

This stanza entry is optional. When not specified, the default is no.

Default value: no. The WebSEAL configuration sets this value.

Example: `disable-tls-v1 = no`

### ssl-v2-timeout
Session timeout in seconds for SSL v2 connections between clients and servers. This timeout value controls how often a full SSL handshake is completed between clients and WebSEAL.

Valid range of values for `number_of_seconds` is from 1-100 seconds.

This value is set by the WebSEAL configuration utility.

This stanza entry is required when SSL is enabled.

Default value: 100

Example: `ssl-v2-timeout = 100`

### ssl-v3-timeout
Session timeout in seconds for SSL v3 connections between clients and servers. This timeout value controls how often a full SSL handshake is completed between clients and WebSEAL.

Valid range of values for `number_of_seconds` is from 1-86400 seconds, where 86400 seconds is equal to 1 day. If you specify a number outside this range, the default number of 7200 seconds will be used.

This value is set by the WebSEAL configuration utility.

This stanza entry is required when SSL is enabled.

Default value: 7200

Example: `ssl-v3-timeout = 7200`

### ssl-max-entries
408
Integer value indicating the maximum number of concurrent entries in the SSL cache. The minimum value is zero (0) which means that caching is unlimited. Entries between 0 and 256 are set to 256. There is no maximum limit.

This stanza entry is optional.

When the stanza entry is not assigned a value, WebSEAL uses a default value of 0. The WebSEAL configuration utility, however, assigns a default value of 4096.

Example: `ssl-max-entries = 4096`

See also the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

### gsk-crl-cache-size = number_of_entries

Integer value indicating the maximum number of entries in the GSKit CRL cache. Minimum value is 0. A value of 0 means that no entries are cached.

Neither WebSEAL nor GSKit impose a maximum value on this cache. See the discussion on maximum values for integers in “Guidelines for configuring stanzas” on page 378.

See the Secure Socket Layer Introduction and iKeyman User’s Guide for more information on GSKit. See also the standards documents for SSL V3 and TLS V1 (RFC 2246) for more information on CRLs.

This stanza entry is required.

Default value: 0

Example: `gsk-crl-cache-size = 0`

### gsk-crl-cache-entry-lifetime = number_of_seconds

Integer value specifying the lifetime timeout, in seconds, for individual entries in the GSKit CRL cache. The minimum value is 0. The maximum value is 86400.

Neither WebSEAL nor GSKit impose a maximum value on the cache entry lifetime. See the discussion on maximum values for integers in “Guidelines for configuring stanzas” on page 378.

See the Secure Socket Layer Introduction and iKeyman User’s Guide for more information on GSKit. See also the standards documents for SSL V3 and TLS V1 (RFC 2246) for more information on CRLs.

This stanza entry is required.

Default value: 0

Example: `gsk-crl-cache-entry-lifetime = 0`

### crl-ldap-server = server_name

Name of the LDAP server to be referenced for Certificate Revocation List (CRL) checking during SSL authentication.

This stanza entry is optional.

There is no default value.

Example: `crl-ldap-server = surf.santacruz.ibm.com`

### crl-ldap-server-port = port_number
<table>
<thead>
<tr>
<th>Port number for communication with the LDAP server specified in cr1-ldap-server. The LDAP server is referenced for Certificate Revocation List (CRL) checking during SSL authentication. This stanza entry is optional. When cr1-ldap-server is set, this stanza entry is required. There is no default value. Example: cr1-ldap-server-port = 389</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr1-ldap-user = user_DN</td>
</tr>
<tr>
<td>Fully qualified distinguished name (DN) of an LDAP user that has access to the Certificate Revocation List. This stanza entry is optional. A null value for cr1-ldap-user indicates that the SSL authenticator should bind to the LDAP server anonymously. There is no default value. Example: cr1-ldap-user = cn=webseal/surf,cn=SecurityDaemons,secAuthority=Default</td>
</tr>
<tr>
<td>cr1-ldap-user-password = password</td>
</tr>
<tr>
<td>Password for the user specified in cr1-ldap-user. This stanza entry is optional. There is no default value. Example: cr1-ldap-user-password = mypasswd</td>
</tr>
<tr>
<td>pkcs11-driver-path = fully_qualified_path</td>
</tr>
<tr>
<td>Path to a shared library that provides GSKit support for external PKCS#11 device drivers. This stanza entry is optional. There is no default value. Example: pkcs11-driver-path = /usr/lib/pkcs11/PKCS11_API.so See “Configure WebSEAL and GSKit to use the PKCS#11 shared library” on page 36.</td>
</tr>
<tr>
<td>pkcs11-token-label = name_of_label</td>
</tr>
<tr>
<td>Label for the token device that stores the WebSEAL public/private key pair. This stanza entry is optional. There is no default value: Example: pkcs11-token-label = websealToken See “Configure WebSEAL and GSKit to use the PKCS#11 shared library” on page 36.</td>
</tr>
<tr>
<td>pkcs11-token-pwd = password</td>
</tr>
</tbody>
</table>
String containing the password to protect the private keys in the token keyfile.

This stanza entry is optional.

There is no default value.

Example: pkcs11-token-pwd = secret

disable-ncipher-bsafe = {yes|no}

Disables or permits the automatic use by WebSEAL of nCipher hardware cards for SSL acceleration over BHAPI (Bsafe). WebSEAL detects this hardware when present, and uses it unless this stanza entry is set to yes.

This stanza entry is optional.

Default value: no

Example: disable-ncipher-bsafe = no

See “Configuring WebSEAL for cryptographic hardware over BHAPI” on page 34.

disable-rainbow-bsafe = {yes|no}

Disables or permits the automatic use by WebSEAL of Rainbow Cryptoswift hardware cards for SSL acceleration over BHAPI (Bsafe) WebSEAL detects this hardware when present, and uses it unless this stanza entry is set to yes.

This stanza entry is optional.

Default value: no

Example: disable-rainbow-bsafe = no

See “Configuring WebSEAL for cryptographic hardware over BHAPI” on page 34.

base-crypto-library = {Default|RSA|ICC}

Specifies the cipher engine used by GSKit. Valid values are Default, RSA, and ICC. Note that setting it to RSA affects the settings possible for fips-mode-processing.

The value Default tells GSKit to use the optimal cryptographic base.

This stanza entry is required.

Default value: Default

Example: base-crypto-library = Default

fips-mode-processing = {yes|no}

Enables or disables FIPS mode processing. A value of yes enables it. A value of no disables it.

When base-crypto-library = RSA, this value must be no.

This stanza entry is required.

Default value: yes

Example: fips-mode-processing = {yes|no}

ssl-auto-refresh = {yes|no}
Indicates whether automatic refresh of the SSL certificate and the key database file password occur.

Valid values are:
- **yes**: Automatic refresh is enabled. When enabled, the certificate and password are regenerated if either is in danger of expiration (less than half the time left). This value is the default value.
- **no**: Turn off automatic certificate and password refresh.

This stanza entry is required only when SSL is enabled.

Default value: yes

Example: `ssl-auto-refresh = yes`

**ssl-listening-port = (0|port_number)**

TCP port to listen on for incoming requests.

Valid values are:
- **0**: Disables listening. The default value is 0 if not specified during configuration.
- **port_number**: Enables listening at the specified port number. The valid range for port numbers is any positive number that is allowed by TCP/IP and is not currently being used by another application.

This stanza entry is required only when SSL is enabled.

The default value is supplied by the WebSEAL configuration utility, and is dependent on the available TCP/IP ports. A typical value during WebSEAL configuration is 7234.

Example: `ssl-listening-port = 7234`

**ssl-pwd-life = number_of_days**

Password lifetime for the key database file, specified in the number of days.

For automatic password renewal, the value for the lifetime of a password is controlled by the `number_of_days` value when the server is started.

**Note**: If a certificate and the password to the keyring database file containing that certificate are both expired, then the password must be refreshed first.

Valid values for the `number_of_days` is from 1 to 7,299 days.

This stanza entry is required only if SSL is enabled.

Default value: 183

Example: `ssl-pwd-life = 183`

**ssl-authn-type = authentication_type**

Type of authentication.

This stanza entry is required only when SSL is enabled.

Default value: certificate

Example: `ssl-authn-type = certificate`
Authentication

This section contains the following subsections and stanzas:

<table>
<thead>
<tr>
<th>Section</th>
<th>Stanzas</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Authentication methods” on page 414</td>
<td>[ba]</td>
</tr>
<tr>
<td></td>
<td>[forms]</td>
</tr>
<tr>
<td></td>
<td>[spnego]</td>
</tr>
<tr>
<td></td>
<td>[token]</td>
</tr>
<tr>
<td></td>
<td>[certificate]</td>
</tr>
<tr>
<td></td>
<td>[http-headers]</td>
</tr>
<tr>
<td></td>
<td>[auth-headers]</td>
</tr>
<tr>
<td></td>
<td>[ipaddr]</td>
</tr>
<tr>
<td></td>
<td>[authentication-levels]</td>
</tr>
<tr>
<td></td>
<td>[step-up]</td>
</tr>
<tr>
<td></td>
<td>[mpa]</td>
</tr>
<tr>
<td>“Authentication libraries” on page 419</td>
<td>[authentication-mechanisms]</td>
</tr>
<tr>
<td>“Reauthentication” on page 423</td>
<td>[reauthentication]</td>
</tr>
<tr>
<td>“Authentication failover” on page 424</td>
<td>[failover]</td>
</tr>
<tr>
<td></td>
<td>[failover-attributes]</td>
</tr>
<tr>
<td></td>
<td>[failover-add-attributes]</td>
</tr>
<tr>
<td></td>
<td>[failover-restore-attributes]</td>
</tr>
<tr>
<td>“Cross-domain single sign-on” on page 428</td>
<td>[cdsso]</td>
</tr>
<tr>
<td></td>
<td>[cdsso-peers]</td>
</tr>
<tr>
<td>“e-community single sign-on” on page 431</td>
<td>[e-community-sso]</td>
</tr>
<tr>
<td>“Quality of protection” on page 435</td>
<td>[ssl-qop]</td>
</tr>
<tr>
<td></td>
<td>[ssl-qop-mgmt-hosts]</td>
</tr>
<tr>
<td></td>
<td>[ssl-qop-mgmt-networks]</td>
</tr>
<tr>
<td></td>
<td>[ssl-qop-mgmt-default]</td>
</tr>
</tbody>
</table>
## Authentication methods

### [ba] stanza

| ba-auth = {none|http|https|both} |
|----------------------------------|
| Enables authentication using the Basic Authentication mechanism. Specifies which protocols are supported. The value both means both HTTP and HTTPS. |
| When basic authentication is enabled, you must also configure an appropriate authentication library by setting a key=value pair in the [authentication-mechanisms] stanza. See “Authentication libraries” on page 419 for more information. |
| This stanza entry is required. |
| Default value: https |
| Example: ba-auth = https |

<table>
<thead>
<tr>
<th>basic-auth-realm = Realm_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>String value that specifies the realm name. This name is displayed in the browser’s dialog box when the user is prompted for login information. The string must consist of ASCII characters, and can contain spaces.</td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td>Default value: Access Manager</td>
</tr>
<tr>
<td>Example: basic-auth-realm = Access Manager</td>
</tr>
</tbody>
</table>

### [forms] stanza

| forms-auth = {none|http|https|both} |
|-----------------------------------|
| Enables authentication using the Forms Authentication mechanism. Specifies which protocols are supported. The value both means both HTTP and HTTPS. |
| When forms authentication is enabled, you must also configure an appropriate authentication library by setting a key=value pair in the [authentication-mechanisms] stanza. See “Authentication libraries” on page 419 for more information. |
| This stanza entry is required. |
| Default value: none |
| Example: forms-auth = none |

### [spnego] stanza

| spnego-auth = {none|http|https|both} |
|----------------------------------|

Enables authentication using the SPNEGO authentication mechanism. Specifies which protocols are supported. The value both means both HTTP and HTTPS.

When SPNEGO authentication is enabled, you must also configure an appropriate authentication library by setting a key=value pair in the [authentication-mechanisms] stanza. See "Authentication libraries" on page 419 for more information.

This stanza entry is required.
Default value: none
Example: spnego-auth = none

<table>
<thead>
<tr>
<th>spnego-krb-service-name = kerberos_server_principal_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the Kerberos service-principal-name (SPN) for the server. This name is created by combining the string HTTP with the hostname. The syntax is: HTTP@host_name</td>
</tr>
<tr>
<td>The host name is the DNS name by which browsers contact the Web server. In most cases, host name should be fully qualified.</td>
</tr>
<tr>
<td>This stanza entry is required on UNIX platforms only.</td>
</tr>
<tr>
<td>Default value: (none)</td>
</tr>
<tr>
<td>Example: spnego-auth = <a href="mailto:HTTP@diamond.subnet2.ibm.com">HTTP@diamond.subnet2.ibm.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>spnego-krb-keytab-file = fully_qualified_path</th>
</tr>
</thead>
<tbody>
<tr>
<td>The path to the Kerberos keytab file for the WebSEAL server.</td>
</tr>
<tr>
<td>This stanza entry is required on UNIX platforms only.</td>
</tr>
<tr>
<td>Default value: (none)</td>
</tr>
<tr>
<td>Example: spnego-krb-keytab-file = /opt/pdweb/etc/diamond_HTTP.keytab</td>
</tr>
</tbody>
</table>

**[token] stanza**

token-auth = {none|http|https|both}

Enables authentication using the token authentication mechanism. Specifies which protocols are supported. The value both means both HTTP and HTTPS.

When token authentication is enabled, you must also configure an appropriate authentication library by setting a key=value pair in the [authentication-mechanisms] stanza. See "Authentication libraries" on page 419 for more information.

This stanza entry is required.
Default value: none
Example: token-auth = none

**[certificate] stanza**

accept-client-certs = {never|required|optional|prompt_as_needed}
Specifies how to handle certificates from HTTPS clients. Options are:

- **never**
  - Never request a client certificate

- **required**
  - Always request a client certificate. Do not accept the connection if the client does not present a certificate. When this value is set to required, all other authentication settings are ignored for HTTPS clients.

- **optional**
  - Always request a client certificate. If presented, use it.

  - **prompt_as_needed**
    - Do not prompt for a client certificate until the client attempts to access a resource that requires certificate authentication.
    - **Note**: When this value is set, ensure that the ssl-1d-sessions key in the [ssl] stanza is set to no.

When certificate authentication is enabled, you must also configure an appropriate authentication library by setting a `key=value` pair in the [authentication-mechanisms] stanza. See “Authentication libraries” on page 419 for more information.

This stanza entry is required.

Default value: never

Example accept-client-certs = never

cert-cache-max-entries = number_of_entries

Maximum number of concurrent entries in the Certificate SSL ID cache.

There is no absolute maximum size for the cache. However, the size of the cache cannot exceed the size of the SSL ID cache.

A maximum size of 0 allows an unlimited cache size.

This stanza entry is required only when the accept-client-certs key is set to prompt_as_needed.

The default value is 1024.

Example: cert-cache-max-entries = 1024.

See also “Enable and configure the Certificate SSL ID cache” on page 153.

cert-cache-timeout = number_of_seconds

Maximum lifetime, in seconds, for an entry in the Certificate SSL ID cache.

The minimum value is zero. A value of zero means that when the cache is full, the entries are cleared based on a Least Recently Used algorithm.

This stanza entry is required only when the accept-client-certs key is set to prompt_as_needed.

The default value is 120.

Example: cert-cache-timeout = 120

See also “Set the timeout for Certificate SSL ID cache” on page 154.

[http-headers] stanza

http-headers-auth = {none|http|https|both}
### [http-headers-auth] stanza

**header = header_name**

Enables authentication using an HTTP header authentication mechanism. Specifies which protocols are supported. The value both means both HTTP and HTTPS.

When HTTP header authentication is enabled, you must also configure an appropriate authentication library by setting a `key=value` pair in the `[authentication-mechanisms]` stanza. See “Authentication libraries” on page 419 for more information.

This stanza entry is required.

Default value: none

Example: `http-headers-auth = none`

### [ipaddr] stanza

**ipaddr-auth = {none|http|https|both}**

Enables authentication using a client’s IP address. Specifies which protocols are supported. The value both means both HTTP and HTTPS.

When IP address authentication is enabled, you must also configure an appropriate authentication library by setting a `key=value` pair in the `[authentication-mechanisms]` stanza. See “Authentication libraries” on page 419 for more information.

This stanza entry is required.

The default value is none.

Example: `ipaddr-auth = none`

### [authentication-levels] stanza

**level = {unauthenticated|password|token-card}**
Step-up authentication levels. WebSEAL enables authenticated users to increase the authentication level by use of step-up authentication. This key=value pair specifies which step-up authentication levels are supported by this WebSEAL server.

Do not specify an authentication level unless the authentication method is enabled. For example, you must enable either basic authentication or forms authentication before setting level = password.

Enter a separate key=value pair for each supported level.

This stanza entry is required.

Default values:
level = authenticated
level = password

Example: level = password

[step-up] stanza
verify-step-up-user = {yes|no}

Enforces policy requiring that the identity of the user performing the step-up operation match the identity of the user that performed the original authentication operation. To enforce this policy, set the value to yes.

This stanza entry is required.

Default value: no

Example: verify-step-up-user = yes

[mpa] stanza
mpa = {yes|no}

Enables support for multiplexing proxy agents.

This stanza entry is required.

Default value: no

Example: mpa = no
## Authentication libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>passwd-cdas</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for the library that implements a CDAS library for either basic authentication or forms authentication.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td><code>passwd-ldap</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements basic authentication with an LDAP user registry.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td></td>
<td>Example (entered as one line):</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>passwd-ldap</code> = C:\PROGRA<del>1\Tivoli\POLICY</del>1\bin\ldapauthn.dll</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp; -cfgfile [C:/Program Files/Tivoli/PDWeb/etc/webseald-default.conf]</td>
<td></td>
</tr>
<tr>
<td><code>passwd-uraf</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements basic authentication using the Tivoli Access Manager URAF interface to underlying user registry types.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td><code>cert-ldap</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements certificate authentication.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>The default value for the built-in library on Solaris is:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/opt/PolicyDirector/lib/libcertauthn.so &amp; -chfgfile \</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[/opt/pdweb/etc/webseal-default.conf]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example on Windows (entered as one continuous line):</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>cert-ldap</code> = C:\PROGRA<del>1\Tivoli\POLICY</del>1\bin\libcertauthn.dll</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp; -cfgfile [C:/Program Files/Tivoli/PDWeb/etc/webseal-default.conf]</td>
<td></td>
</tr>
<tr>
<td><code>token-cdas</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements token authentication.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td><code>cert-ssl</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements certificate authentication.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td><code>http-request</code> = <code>fully_qualified_path</code></td>
<td>Fully qualified path for a library that implements forms authentication.</td>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is optional.</td>
<td>There is no default value.</td>
</tr>
</tbody>
</table>
**sso-create** = *fully_qualified_path*

Fully qualified path for a library that implements HTTP header or IP address authentication.

This stanza entry is optional.

There is no default value.

---

**sso-consume** = *fully_qualified_path*

Fully qualified path for a library that implements WebSEAL single sign-on creation authentication.

This stanza entry is optional.

There is no default value.

---

**kerberosv5** = *fully_qualified_path*

Fully qualified path for a library that implements WebSEAL single sign-on consumption authentication. This library is used to provide WebSEAL’s Windows desktop single signon.

This stanza entry is optional.

There is no default value.

Example:

```
kerberosv5 = /opt/PolicyDirector/lib/stliauthn.so
```

---

**passwd-strength** = *fully_qualified_path*

Fully qualified path for a library that implements password strength authentication.

This stanza entry is optional.

There is no default value.

---

**cred-ext-attrs** = *fully_qualified_path*

Fully qualified path for a library that implements credential extended attributes authentication.

This stanza entry is optional.

There is no default value.

---

**su-passwd** = *fully_qualified_path*

Fully qualified path for a library that implements switch user authentication for basic authentication or forms authentication.

This stanza entry is optional.

There is no default value.

See "Switch user authentication" on page 188.

---

**su-token-card** = *fully_qualified_path*
### su-certificate = fully_qualified_path

Fully qualified path for a library that implements switch user authentication for certificate authentication.

This stanza entry is optional.

There is no default value.

See “Switch user authentication” on page 188.

### su-http-request = fully_qualified_path

Fully qualified path for a library that implements switch user authentication for HTTP header or IP address authentication.

This stanza entry is optional.

There is no default value.

See “Switch user authentication” on page 188.

### su-cdsso = fully_qualified_path

Fully qualified path for a library that implements switch user authentication for cross-domain single sign-on authentication.

This stanza entry is optional.

There is no default value.

See “Switch user authentication” on page 188.

### su-kerberosv5 = fully_qualified_path

Fully qualified path for a library that implements switch user authentication for SPNEGO (Kerberos) authentication.

This stanza entry is optional.

There is no default value.

See “Switch user authentication” on page 188.

### failover-password = fully_qualified_path

Fully qualified path for a library that implements failover cookie authentication for basic authentication or forms authentication.

This stanza entry is optional.

There is no default value.

### failover-token-card = fully_qualified_path

Fully qualified path for a library that implements failover cookie authentication for token card authentication.

This stanza entry is optional.

There is no default value.
<table>
<thead>
<tr>
<th>Description</th>
<th>Path Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully qualified path for a library that implements failover cookie</td>
<td><code>failover-certificate = fully_qualified_path</code></td>
</tr>
<tr>
<td>authentication for certificate authentication.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
<td></td>
</tr>
<tr>
<td>There is no default value.</td>
<td></td>
</tr>
<tr>
<td>Fully qualified path for a library that implements failover cookie</td>
<td><code>failover-http-request = fully_qualified_path</code></td>
</tr>
<tr>
<td>authentication for HTTP header authentication or IP address authentication.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
<td></td>
</tr>
<tr>
<td>There is no default value.</td>
<td></td>
</tr>
<tr>
<td>Fully qualified path for a library that implements failover cookie</td>
<td><code>failover-cdsso = fully_qualified_path</code></td>
</tr>
<tr>
<td>authentication for cross-domain single sign-on authentication.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
<td></td>
</tr>
<tr>
<td>There is no default value.</td>
<td></td>
</tr>
<tr>
<td>Fully qualified path for a library that implements failover cookie</td>
<td><code>failover-kerberosv5 = fully_qualified_path</code></td>
</tr>
<tr>
<td>authentication for SPNEGO authentication.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
<td></td>
</tr>
<tr>
<td>There is no default value.</td>
<td></td>
</tr>
<tr>
<td>Fully qualified path for a library that implements post password change</td>
<td><code>post-pwdchg-process = fully_qualified_path</code></td>
</tr>
<tr>
<td>processing. This is called by WebSEAL when the user changes a password</td>
<td></td>
</tr>
<tr>
<td>using the <code>pkms</code> password change page.</td>
<td></td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
<td></td>
</tr>
<tr>
<td>There is no default value.</td>
<td></td>
</tr>
</tbody>
</table>

The following example shows the path name to the built-in post password change library, as supplied with WebSEAL on Solaris Operating Environment:

```
pwdchg-process = /opt/PolicyDirector/lib/libxauthn.so
```
## Reauthentication

<table>
<thead>
<tr>
<th>[reauthentication] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>reauth-for-inactive = {yes</td>
</tr>
<tr>
<td>Enables WebSEAL to prompt users to reauthenticate when their entry in the WebSEAL credential cache has timed out due to inactivity.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: no</td>
</tr>
<tr>
<td>Example: reauth-for-inactive = no</td>
</tr>
</tbody>
</table>

| reauth-reset-lifetime = {yes|no} |
| Enables WebSEAL to reset the lifetime timer for WebSEAL credential cache entries following successful reauthentication. |
| This stanza entry is required. |
| Default value: no |
| Example: reauth-reset-lifetime = no |

| reauth-extend-lifetime = number_of_seconds |
| Integer value expressing the time in seconds that the credential cache timer should be extended to allow clients to complete a reauthentication. |
| When the value is zero (0), the lifetime timer is not extended. |
| WebSEAL imposes no maximum. The maximum value is limited only by the integer data type. See the discussion of integer maximum values in "Guidelines for configuring stanzas" on page 378. |
| This stanza entry is required. |
| Default value: 0 |
| Example: reauth-extend-lifetime = 300 |
## Authentication failover

- [failover]
- [failover-attributes]
- [failover-add-attributes]
- [failover-restore-attributes]

### [failover] stanza

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failover-auth = {none</td>
<td>http</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failover-cookies-keyfile = fully_qualified_path</td>
<td>A Key file for failover cookie encryption. The cdsso_key_gen utility must be used to generate this file. This stanza entry is optional. There is no default value. Example: failover-cookies-keyfile = /tmp/failover.key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failover-cookie-lifetime = number_of_minutes</td>
<td>An integer value specifying the number of minutes that failover cookie contents are valid. Must be a positive integer. There is no maximum value. See the discussion of integer maximum values in “Guidelines for configuring stanzas” on page 378. This stanza entry is required. Default: 60 Example: failover-cookie-lifetime = 60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-utf8 = {yes</td>
<td>no}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable-failover-cookie-for-domain = {yes</td>
<td>no}</td>
</tr>
<tr>
<td>Failover Feature</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>enable-failover-cookie-for-domain</td>
<td>Enables the failover cookie for the domain. This stanza entry is required. Default: no Example: enable-failover-cookie-for-domain = no</td>
</tr>
<tr>
<td>failover-update-cookie</td>
<td>The maximum interval, in number of seconds, allowed between updates of the session activity timestamp in the failover cookies. The value is an integer. When the server receives a request, if the number of seconds specified for this parameter has passed, the session activity timestamp is updated. When the value is 0, the session activity timestamp is updated on every request. When the value is less than zero (negative number), the session activity timestamp is never updated. There is no maximum value. This stanza entry is required. Default value: 60 Example: failover-cookie-update = 60</td>
</tr>
<tr>
<td>failover-require-lifetime-timestamp-validation</td>
<td>Enables or disables the requirement of a session lifetime timestamp validation in the failover cookie. For backwards compatibility with versions of WebSEAL server prior to Version 5.1, set this stanza entry to no. Versions prior to Version 5.1 did not create the session lifetime timestamp in the failover cookie. This stanza entry is required. Default value: no Example: failover-require-lifetime-timestamp-validation = no</td>
</tr>
<tr>
<td>failover-require-activity-timestamp-validation</td>
<td>Enables or disables the requirement of a session activity timestamp validation in the failover cookie. For backwards compatibility with versions of WebSEAL server prior to Version 5.1, set this stanza entry to no. Versions prior to Version 5.1 did not create the session activity timestamp in the failover cookie. This stanza entry is required. Default value: no Example: failover-require-activity-timestamp-validation = no</td>
</tr>
</tbody>
</table>

[failover-add-attributes] stanza

attribute_pattern = add
List of attributes from the original credential that must be preserved in the failover cookie. The format is:

\[\text{attribute_pattern} = \text{add}\]

The attribute pattern is a case-insensitive wildcard pattern.

The order of entries in the stanza is important. Rules (patterns) that appear earlier in the stanza take precedence over those that appear later in the stanza. Attributes that do not match any pattern will not be added to the failover cookie.

Entries in this stanza are optional.

There are no default entries in this stanza. However, the attribute AUTHENTICATION_LEVEL is added to the failover cookie by default. This attribute does not need to be included in the configuration stanza.

Example:

\[\text{tagvalue_*} = \text{add}\]

\[\text{session-lifetime-timestamp} = \text{add}\]

This entry specifies that the timestamp for creation of the original session be taken from the failover cookie and added to the new session on the replicated server.

This attribute cannot be specified by pattern matching. This entry must be added exactly as it is written.

This stanza entry is optional.

Default entry: none

Example:

\[\text{session-lifetime-timestamp} = \text{add}\]

\[\text{session-activity-timestamp} = \text{add}\]

This entry specifies that the timestamp for the last user activity be taken from the failover cookie and added to the new session on the replicated server.

This attribute cannot be specified by pattern matching. This entry must be added exactly as it is written.

This stanza entry is optional.

Default entry: none

Example:

\[\text{session-activity-timestamp} = \text{add}\]

\[\text{[failover-restore-attributes]} \text{ stanza}\]

\[\text{attribute_pattern} = \text{preserve}\]
<table>
<thead>
<tr>
<th>attribute_pattern</th>
<th>refresh</th>
</tr>
</thead>
</table>
| A list of failover cookie attributes to omit from the recreated user credential. The format is:  
attribute_pattern = refresh  
This list is not needed in all configurations. The default behavior when recreating a user credential is to omit all attributes that are not specified with a value of preserve. In some cases it might be necessary to specify an exception to a wildcard pattern matching, to ensure that a specific attribute gets refreshed, not preserved. This specification might be necessary, for example, when using a custom CDAS.  
The attribute pattern is a case-insensitive wildcard pattern.  
The order of entries in the stanza is important. Rules (patterns) that appear earlier in the stanza take precedence over those that appear later in the stanza. Attributes that do not match any pattern will not be added to the credential.  
Entries in this stanza are optional.  
Default entries: none  
Example:  
tagvalue_special_* = refresh |

| preserve |
|-------------------|---------|
| List of attributes to put in the new credential when recreating a credential from a failover cookie. The format is:  
attribute_pattern = preserve  
The attribute pattern is a case-insensitive wildcard pattern.  
The order of entries in the stanza is important. Rules (patterns) that appear earlier in the stanza take precedence over those that appear later in the stanza. Attributes that do not match any pattern will not be added to the credential.  
When WebSEAL recreates a credential, all failover cookie attributes are ignored unless specified by an entry with the value preserve.  
Entries in this stanza are optional.  
Default entries: none  
Example:  
tagvalue_* = preserve |
Cross-domain single sign-on

- [cdsso]
- [cdsso-peers]

<table>
<thead>
<tr>
<th>[cdsso] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdsso-auth = {none</td>
</tr>
<tr>
<td>Enables WebSEAL to accept tokens. Requires that an authentication mechanism is specified for the token consume (sso-consume) library in the [authentication-mechanisms] stanza. Specifies which protocols are supported. The value both means both HTTP and HTTPS.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: none</td>
</tr>
<tr>
<td>Example: cdsso-auth = none</td>
</tr>
</tbody>
</table>

| cdsso-create = {none|http|https|both} |
| Enables WebSEAL to accept tokens. Requires that an authentication mechanism is specified for the token create (sso-create) library in the [authentication-mechanisms] stanza. Specifies which protocols are supported. The value both means both HTTP and HTTPS. |
| This stanza entry is required. |
| Default value: none |
| Example: cdsso-create = none |

| authtoken-lifetime = number_of_seconds |
| Positive integer that expresses the number of seconds for which the single sign-on authentication token is valid. Minimum value: 1. There is no maximum value. See the discussion of integer maximum values in "Guidelines for configuring stanzas" on page 378. |
| This stanza entry is required. |
| Default value: 180 |
| Example: authtoken-lifetime = 180 |

| cdsso-argument = argument_name |
| Name of the argument containing the cross-domain single sign-on token in a query string in a request. This is used to identify incoming requests that contain CDSSO authentication information. |
| Valid characters are any ASCII characters, except for question mark ( ? ), ampersand ( & ), and equals sign ( = ). |
| This stanza entry is required. |
| Default value: PD-ID |
| Example: cdsso-argument = PD-ID |

| use-utf8 = {true|false} |
Use UTF-8 encoding for tokens used in cross domain single sign-on.

Beginning with Version 5.1, WebSEAL servers use UTF-8 encoding by default. When this stanza entry is set to true, tokens can be exchanged with other WebSEAL servers that use UTF-8 encoding. This enables tokens to used across different code pages (such as for a different language).

For backwards compatibility with tokens created by WebSEAL servers from version prior to 5.1, set this stanza entry to false.

This stanza entry is required.

Default: true

Example: use-utf8 = true

See also “Multi-locale support with UTF-8” on page 44.

[cdsso-peers] stanza

```plaintext
fully_quired_hostname = fully_quired_path
```

List of peer servers that are participating in cross-domain single-sign on. The path name must specify the location of server’s key file.

This stanza entry is optional.

There is no default value

Example: webhost2.ibm.com = /tmp/cdsso.key

[cdsso-token-attributes] stanza

```plaintext
domain_name = pattern1 [pattern2,] [patternN, .... ]
```

Credential attributes to include in CDSSO authentication tokens.

The domain_name specifies the destination domain containing the server that will consume the token. The value for domain_name can be one or more entries. The value can be either a specific value or can be a pattern that uses standard Tivoli Access Manager wildcard characters (*, [, ^, \, ?).

This stanza entry is optional.

There is no default value

Example: example1.com = my_cdas_attr_*

```plaintext
<default> = pattern1 [pattern2,] [patternN, .... ]
```

Credential attributes to include in CDSSO authentication tokens.

When WebSEAL cannot find a domain_name entry to match the domain, the entries in <default> are used. The word <default> is a key word and must not be modified.

This stanza entry is optional.

There is no default value

Example: <default> = my_cdas_attr_*

[cdsso-incoming-attributes] stanza
| attribute_pattern = \{preserve|refresh\} |
|------------------------------------------------|
| Extended attributes to extract from incoming CDSSO authentication tokens. |
| The attributes typically match those declared in the [cdsso-token-attributes] stanza for the WebSEAL server in the source domain. |
| The attribute_pattern can be either a specific value or can be a pattern that uses standard Tivoli Access Manager wildcard characters ( *, [ ], ^, \, ?). |
| The order of attribute_pattern entries is important. The first entry that matches the attribute is used. Other entries are ignored. |
| This stanza entry is optional. |
| There is no default value |
| Example: my_cred_attr1 = preserve |
# e-community single sign-on

- [e-community-sso]
- [e-community-domain-keys]
- [ecsso-token-attributes]
- [ecsso-incoming-attributes]

## [e-community-sso] stanza

### e-community-sso-auth = \{none|http|https|both\}

Enables participation in e-community single sign-on. Specifies which protocols are supported. The value both means both HTTP and HTTPS.

This stanza entry is required.

Default value: none

Example: e-community-sso-auth = none

### e-community-name = name

String value that specifies an e-community name. When e-community single sign on is supported, this name must match any vouch-for tokens or e-community cookies that are received.

The string must not contain the equals sign (=) or ampersand (&).

This stanza entry is optional.

There is no default value

Example: e-community-name = company1

### is-master-authn-server = \{yes\|no\}

Specifies whether this WebSEAL server accepts vouch-for requests from other WebSEAL instances. The WebSEAL instances must have domain keys listed in the [e-community-domain-keys] stanza. When this value is yes, this WebSEAL server is the master authentication server.

This stanza entry is optional.

There is no default value

Example: is-master-authn-server = no

### master-authn-server = fully_qualified_hostname

Location of the master authentication server. This value must be specified when is-master-authn-server is set to no. If a local domain login has not been performed then authentication attempts are routed through the master machine. The master machine will vouch for the user identity. The domain key for the master-authn-server needs to be listed in the [e-community-domain-keys] stanza.

This stanza entry is optional.

There is no default value.

Example: master-authn-server = diamond.dev.ibm.com

### master-http-port = port_number

---

Appendix A. WebSEAL configuration file reference 431
### master-http-port = port_number

Integer value specifying the port number on which the master-authn-server listens for HTTP requests. The setting is necessary when e-community-sso-auth permits use of the HTTP protocol, and the master-authn-server listens for HTTP requests on a port other than the standard HTTP port (port 80). This stanza entry is ignored if this WebSEAL server is the master authentication server.

This stanza entry is optional.

There is no default value.

Example: `master-http-port = 81`

### master-https-port = port_number

Integer value specifying the port number on which the master-authn-server listens for HTTPS requests. The setting is necessary when e-community-sso-auth permits use of the HTTPS protocol, and the master-authn-server listens for HTTPS requests on a port other than the standard HTTPS port (port 443). This stanza entry is ignored if this WebSEAL server is the master authentication server.

This stanza entry is optional.

There is no default value.

Example: `master-https-port = 444`

### vf-token-lifetime = number_of_seconds

Positive integer indicating the lifetime, in seconds, of the vouch-for token. This is set to account for clock skew between participant servers. The minimum value is 1 second. There is no maximum value. See the discussion of integer maximum values in ["Guidelines for configuring stanzas" on page 378](#). This stanza entry is optional.

Default value: 180

Example: `vf-token-lifetime = 180`

### vf-url = URL_designation

Designator for vouch-for URL. This specifies the start of a URL relative to the server root. This is used to construct vouch-for requests for participating e-community single sign-on servers, and to distinguish requests for vouch-for information from other requests by the master authentication server.

The `URL_designation` string can contain alphanumerics and the following special characters: dollar sign ($), hyphen (-), underscore (_), period (.), plus sign (+), exclamation point (!), asterisk (*), single quote (’), parentheses “()” and comma (,).

Questions marks (?) are not allowed

This stanza entry is optional.

When the stanza entry is not present in the configuration file, the default value is /pkmsvouchfor.

Example: `vf-url = /pkmsvouchfor`

### vf-argument = vouch-for_token_name

Example: `vf-argument = vouch-for_token_name`
String value containing the name of the vouch-for token contained in a vouch-for reply. This is used to construct the vouch-for replies by the master authentication server, and to distinguish incoming requests as ones with vouch-for information by participating e-community single sign-on servers. Valid characters for the string are ASCII characters except for ampersand ( & ), equals sign ( = ), and question mark ( ? ).

This stanza entry is optional.
Default value: PD-VF
Example: vf-argument = PD-VF

<table>
<thead>
<tr>
<th>stanza entry</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ec-cookie-lifetime</strong></td>
<td><strong>number_of_minutes</strong></td>
</tr>
<tr>
<td>Positive integer value indicating the lifetime of an e-community cookie. Minimum value is 1. There is no maximum value. See the discussion of integer maximum values in “Guidelines for configuring stanzas” on page 378.</td>
<td></td>
</tr>
</tbody>
</table>
This stanza entry is required.
Default: 300
Example: ec-cookie-lifetime = 300

<table>
<thead>
<tr>
<th>stanza entry</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ecsso-allow-unauth</strong></td>
<td>**{yes</td>
</tr>
<tr>
<td>Enables or disables unauthenticated access to unprotected resources on an e-community SSO slave server. The value yes enables unauthenticated access. The value no disables access. For compatibility with versions of WebSEAL prior to 5.1 set this to no.</td>
<td></td>
</tr>
</tbody>
</table>
This stanza entry is required.
Default: yes
Example: ecsso-allow-unauth = yes

<table>
<thead>
<tr>
<th>stanza entry</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>use-utf8</strong></td>
<td>**{yes</td>
</tr>
<tr>
<td>Use UTF-8 encoding for tokens used in e-community single sign-on. Beginning with Version 5.1, WebSEAL servers use UTF-8 encoding by default. When this stanza entry is set to true, tokens can be exchanged with other WebSEAL servers that use UTF-8 encoding. This enables tokens to used across different code pages (such as for a different language). For backwards compatibility with tokens created by WebSEAL servers from version prior to 5.1, set this stanza entry to no.</td>
<td></td>
</tr>
</tbody>
</table>
This stanza entry is required.
Default: yes
Example: use-utf8 = yes

See also “Multi-locale support with UTF-8” on page 44.

### [e-community-domain-keys] stanza

<table>
<thead>
<tr>
<th>stanza entry</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>domain_name</strong></td>
<td><strong>fully_qualified_path</strong></td>
</tr>
</tbody>
</table>

Appendix A. WebSEAL configuration file reference 433
File names for keys for any domains that are participating in the e-community. This includes the domain in which the WebSEAL server is running. These are shared on a pair-wise-by-domain basis.

This stanza entry is optional.

There is no default value.

Example: ecssoserver.subnet.ibm.com = /tmp/ecsso.key

### [ecsso-token-attributes] stanza

```
domain_name = pattern1 [pattern2] [patternN, .... ]
```

Credential attributes to include in eCSSO authentication tokens.

The `domain_name` specifies the destination domain containing the server that will consume the token. The value for `domain_name` can be one or more entries. The value can be either a specific value or can be a pattern that uses standard Tivoli Access Manager wildcard characters ( *, [], ^, \, ?).

This stanza entry is optional.

There is no default value

Example: example1.com = my_cdas_attr_*

```
<default> = pattern1 [pattern2] [patternN, .... ]
```

Credential attributes to include in eCSSO authentication tokens.

When WebSEAL cannot find a `domain_name` entry to match the domain, the entries in "<default>" are used. The word `<default>` is a key word and must not be modified.

This stanza entry is optional.

There is no default value

Example: `<default>` = my_cdas_attr_*

### [ecsso-incoming-attributes] stanza

```
attribute_pattern = {preserve|refresh}
```

Extended attributes to extract from incoming eCSSO authentication tokens.

The attributes typically match those declared in the `[cdsso-token-attributes]` stanza for the WebSEAL server in the source domain.

The `attribute_pattern` can be either a specific value or can be a pattern that uses standard Tivoli Access Manager wildcard characters ( *, [], ^, \, ?).

The order of `attribute_pattern` entries is important. The first entry that matches the attribute is used. Other entries are ignored.

This stanza entry is optional.

There is no default value

Example: `my_cred_attr1` = preserve
Quality of protection

- [ssl-qop]
- [ssl-qop-mgmt-hosts]
- [ssl-qop-mgmt-networks]
- [ssl-qop-mgmt-default]

### [ssl-qop] stanza

**ssl-qop-mgmt = {yes|no}**

Enables or disables SSL quality of protection management. The value yes enables SSL quality of protection management. The value no disables it.

This stanza entry is required.

Default value: no

Example: `ssl-qop-mgmt = no`

### [ssl-qop-mgmt-hosts] stanza

**host-ip = {ALL|NONE|cipher_level}**

List of string values to specify the allowed encryption levels for HTTPS access for a specific IP address. The value ALL allows all ciphers. The value NONE disables all ciphers and uses an MD5 MAC check sum.

To specify allowable ciphers for a selected group of IP addresses, create a separate entry for each address. For example:

- `111.222.333.444 = RC4-128`
- `222.666.333.111 = RC2-128`

This stanza entry is optional.

There is no default value.

Example: `111.222.333.444 = ALL`

Note that this stanza has been deprecated and is retained only for backwards compatibility. For more information, including a list of supported cipher_levels, see "Quality of protection levels" on page 38.

### [ssl-qop-mgmt-networks] stanza

**network/netmask = {ALL|NONE|cipher_level}**
List of string values to specify the allowed encryption levels for HTTPS access for a specific combination of IP address and netmask. The value ALL allows all ciphers. The value NONE disables all ciphers and uses an MD5 MAC check sum.

To specify allowable ciphers for a selected group of IP addresses and netmasks, create a separate entry for each address/netmask combination. For example:

111.222.333.444/255.255.255.0 = RC4-128
222.666.333.111/255.255.0.0 = RC2-128

This stanza entry is optional.

There is no default value.

Example: 111.222.333.444/255.255.255.0 = RC4-128

Note that this stanza has been deprecated and is retained only for backwards compatibility. For more information, including a list of supported cipher_levels, see "Quality of protection levels" on page 38.

---

**[ssl-qop-mgmt-default] stanza**

default = {ALL|NONE|cipher_level}

List of string values to specify the allowed encryption levels for HTTPS access. The value ALL allows all ciphers. The value NONE disables all ciphers and uses an MD5 MAC check sum.

To specify a selected group of ciphers, create a separate entry for each cipher. For example:

default = RC4-128
default = RC2-128
default = DES-168

Values specified in this stanza entry are used for all IP addresses that are not matched in either the [ssl-qop-mgmt-hosts] stanza entries or the [ssl-qop-mgmt-networks] stanza entries.

This stanza entry is required.

Default value: ALL

Example: default = ALL

For more information, including a list of supported cipher_levels, see "Quality of protection levels" on page 38.
## Session

### [session] stanza

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-entries = number_of_entries</td>
<td>Maximum number of concurrent entries in the credentials cache. The minimum number for this value is 0. A value of zero means that the cache is of unlimited size. WebSEAL does not impose a maximum value. See the guidelines on maximum size of integer values in &quot;Guidelines for configuring stanzas&quot; on page 378. This stanza entry is required. Default value: 4096 Example: max-entries = 4096 See also &quot;Configuring the WebSEAL session/credentials cache&quot; on page 133 and the IBM Tivoli Access Manager for e-business Performance Tuning Guide.</td>
</tr>
<tr>
<td>timeout = number_of_seconds</td>
<td>Integer value for maximum lifetime, in seconds, for an entry in the credential cache. The minimum number for this value is 0. A value of 0 means that the lifetime is unlimited. WebSEAL does not impose a maximum value. See the guidelines on maximum size of integer values in &quot;Guidelines for configuring stanzas&quot; on page 378. This stanza entry is required. Default value: 3600 Example: timeout = 3600 See also &quot;Configuring the WebSEAL session/credentials cache&quot; on page 133</td>
</tr>
<tr>
<td>inactive-timeout = number_of_seconds</td>
<td></td>
</tr>
</tbody>
</table>
Integer value for lifetime, in seconds, of inactive entries in the credential cache.

The minimum number for this value is 0. A value of 0 means that when the cache is full, the entries are cleared based on a Least Recently Used algorithm.

WebSEAL does not impose a maximum value. See the guidelines on maximum size of integer values in "Guidelines for configuring stanzas" on page 378.

This stanza entry is required.

Default value: 600

Example: inactive-timeout = 600

See also "Configuring the WebSEAL session/credentials cache" on page 133.

ssl-id-sessions = {yes|no}

Indicates whether to use the SSL ID to maintain a user's HTTP login session.

The Opera browser, in its default configuration, does not maintain SSL IDs across SSL connections. When using the Opera browser, ssl-id-sessions must be set to no.

This stanza entry is required.

Default value: yes

Example: ssl-id-sessions = yes

See also "Maintaining state with session cookies" on page 135.

Usage note:

This value must be set to no when the following key = value pair is set:

[certificate]
accept-client-certs = prompt_as_needed

For more information, see "Specify the certificate authentication mechanism" on page 152.

use-same-session = {yes|no}

Indicates whether to use the same session for SSL and HTTP clients. When set to yes, a user who has authenticated over HTTP will be authenticated when connecting over HTTPS. Likewise, the user who has authenticated over HTTPS will be authenticated when connecting over HTTP.

Using yes will override ssl-id-sessions = yes, because HTTP clients do not read an SSL ID to maintain sessions.

This stanza entry is required.

Default value: no

Example: use-same-session = no

See also "Maintaining state with session cookies" on page 135.

resend-webseal-cookies = {yes|no}
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resend-webseal-cookies</code></td>
<td>Indicates whether to send the WebSEAL cookies with every response.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Default value: no</td>
</tr>
<tr>
<td></td>
<td>Example: <code>resend-webseal-cookies = no</code></td>
</tr>
<tr>
<td></td>
<td>See also “Maintaining state with session cookies” on page 135</td>
</tr>
<tr>
<td><code>allow-backend-domain-cookies</code></td>
<td>Indicates whether WebSEAL is allowed to send domain cookies from a back-end</td>
</tr>
<tr>
<td></td>
<td>server to a client.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Default value: no</td>
</tr>
<tr>
<td></td>
<td>Example: <code>allow-backend-domain-cookies = no</code></td>
</tr>
<tr>
<td></td>
<td>See also “Handling domain cookies” on page 308</td>
</tr>
<tr>
<td><code>user-session-ids</code></td>
<td>Enables or disables the creation and handling of user session IDs.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>Default value: no</td>
</tr>
<tr>
<td></td>
<td>Example: <code>user-session-ids = yes</code></td>
</tr>
<tr>
<td></td>
<td>See also “Maintaining session state between client and back-end applications” on page 343</td>
</tr>
</tbody>
</table>
Content

This section contains the following topics and stanzas:

<table>
<thead>
<tr>
<th>Section</th>
<th>Stanzas</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Content management” on page 441</td>
<td>[content]</td>
</tr>
<tr>
<td>“Account management” on page 443</td>
<td>[acnt-mgt]</td>
</tr>
<tr>
<td>“Automatic redirect” on page 446</td>
<td>[content] [enable-redirects]</td>
</tr>
<tr>
<td>“Local CGI” on page 447</td>
<td>[cgi] [cgi-types] [cgi-environment-variables]</td>
</tr>
<tr>
<td>“Icons” on page 449</td>
<td>[content-index-icons] [icons]</td>
</tr>
<tr>
<td>“Content caching” on page 451</td>
<td>[content-cache]</td>
</tr>
<tr>
<td>“Content compression” on page 452</td>
<td>[compress-mime-types]</td>
</tr>
<tr>
<td>“Content MIME types” on page 453</td>
<td>[content-mime-types]</td>
</tr>
<tr>
<td>“Content encodings” on page 454</td>
<td>[content-encodings]</td>
</tr>
</tbody>
</table>
# Content management

## [content] stanza

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc-root = <em>fully_qualified_path</em></td>
<td>Root directory of the WebSEAL document tree. The administrator can either accept the default value of <code>www/docs</code> or specify an alternate location. When the administrator accepts the default value, the WebSEAL configuration appends the default value onto the WebSEAL installation directory. Thus the default <code>fully_qualified_path</code> is <code>WebSEAL_installation_directory/www/docs</code>. When the administrator changes this value during configuration, the administrator must specify a fully qualified path. In this case, WebSEAL does not append the entry on to the WebSEAL installation directory. In this case, the <code>doc-root</code> entry is matches the administrator’s input. This stanza entry is required. Default value: <code>WebSEAL_installation_directory/www/docs</code> Example: <code>doc-root = C:/Program Files/Tivoli/PDWeb/www/docs</code></td>
</tr>
<tr>
<td>directory-index = <em>filename</em></td>
<td>Name of a directory index file. When a request is made for a directory located on the local WebSEAL server, WebSEAL looks for this file before attempting a directory listing. This stanza entry is required. Default value: <code>index.html</code> Example: <code>directory-index = index.html</code></td>
</tr>
<tr>
<td>delete-trash-dir = <em>fully_qualified_path</em></td>
<td>Path name to a directory in which files removed by the DELETE method are moved. Files are held there until an administrator removes them. This key does not apply to empty directories, which are always deleted. This key is inactive (commented out in the configuration file) by default. When the key is inactive, files removed by the DELETE method are immediately removed from the file system. This stanza entry is optional. There is no default value: Example: <code>delete-trash-dir = /tmp/trashdir</code></td>
</tr>
<tr>
<td>user-dir = <em>filename</em></td>
<td>Directory in users’ home directories that contain public HTML documents. This stanza entry is required. Default value: <code>public_html</code> Example: <code>user-dir = public_html</code></td>
</tr>
<tr>
<td>utf8-template-macros-enabled = {yes</td>
<td>no}</td>
</tr>
</tbody>
</table>
Specifies how standard WebSEAL HTML files, such as login.html, have data inserted into them when `%MACRO%` strings are encountered. When set to `yes`, data is inserted in UTF-8 format. When set to `no`, data is inserted in the local code page format.

This affects files located in the directories specified by the `error-dir` and `mgt-pages-root` (in the `[acnt-mgt]` stanza) configuration file entries.

WebSEAL HTML pages use a UTF8 charset by default. If you modify charset to specify the local code page, set this entry to `no`.

This stanza entry is required.

Default value: `yes`

Example: `utf8-template-macros-enabled = yes`

<table>
<thead>
<tr>
<th><strong>error-dir = relative_directory</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory where HTML error pages are located. The directory is relative to the directory specified by the <code>server-root</code> entry. When WebSEAL needs to access the error pages, it automatically appends the name of the locale-specific directory.</td>
</tr>
<tr>
<td>For example: /opt/pdweb/www-webseal1/lib/errors/C</td>
</tr>
</tbody>
</table>
| The `error-dir` entry specifies only the portion of the path shown in the example above as `lib/errors`.
| This stanza entry is required.
| Default value: `lib/errors`
| Example: `error-dir = lib/errors` |
## Account management

<table>
<thead>
<tr>
<th>[acnt-mgt] stanza</th>
<th>mgt-pages-root = relative_pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root of account management pages. The actual directory used is a subdirectory of this root directory, as determined by localization settings. This path is relative to the server-root value in the [server] stanza. This stanza entry is required. Default value: lib/html</td>
<td></td>
</tr>
<tr>
<td>Example: mgt-pages-root = lib/html</td>
<td></td>
</tr>
</tbody>
</table>

| login = filename                  | Standard login form. This stanza entry is required. Default value: login.html |
| Example: login = login.html       |

| login-success = filename          | Page displayed after successful login. This stanza entry is required. Default value: login_success.html |
| Example: login-success = login_success.html |

| logout = filename                 | Page displayed after successful logout. This stanza entry is required. Default value: logout.html |
| Example: logout = logout.html     |

| account-locked = filename         | Page displayed when the user authentication fails due to a locked user account. This stanza entry is required. Default value: acct_locked.html |
| Example: account-locked = acct_locked.html |

| passwd-expired = filename         | Page displayed when the user authentication fails due to an expired user password. This stanza entry is required. Default value: passwd_exp.html |
| Example: passwd-expired = passwd_exp.html |

<p>| passwd-change = filename          | |
| Example: passwd-change = passwd_exp.html |</p>
<table>
<thead>
<tr>
<th>Entry</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd-change</td>
<td>Page containing a change password form. This stanza entry is required.</td>
<td>passwd-change = passwd.html</td>
</tr>
<tr>
<td>default</td>
<td>Default value: passwd.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: passwd-change = passwd.html</td>
<td></td>
</tr>
<tr>
<td>passwd-change-success</td>
<td>Page displayed when password change request succeeds. This stanza entry is</td>
<td>passwd-change-success = passwd_rep.html</td>
</tr>
<tr>
<td>default</td>
<td>required. Default value: passwd_rep.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: passwd-change-success = passwd_rep.html</td>
<td></td>
</tr>
<tr>
<td>passwd-change-failure</td>
<td>Page displayed when password change request fails. This stanza entry is</td>
<td>passwd-change-failure = passwd.html</td>
</tr>
<tr>
<td>default</td>
<td>required. Default value: passwd.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: passwd-change-failure = passwd.html</td>
<td></td>
</tr>
<tr>
<td>help</td>
<td>Page containing links to valid administration pages. This stanza entry is</td>
<td>help = help.html</td>
</tr>
<tr>
<td>default</td>
<td>required. Default value: help.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: help = help.html</td>
<td></td>
</tr>
<tr>
<td>token-login</td>
<td>Token login form. This stanza entry is required. Default value: tokenlogin.html</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>tokenlogin.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: token-login = tokenlogin.html</td>
<td></td>
</tr>
<tr>
<td>next-token</td>
<td>Next-token form. This stanza entry is required. Default value: nexttoken.html</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>nexttoken.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: next-token = nexttoken.html</td>
<td></td>
</tr>
<tr>
<td>stepup-login</td>
<td>Step-up authentication login form. This stanza entry is required. Default</td>
<td>stepup-login = stepuplogin.html</td>
</tr>
<tr>
<td>default</td>
<td>value: stepuplogin.html</td>
<td></td>
</tr>
<tr>
<td>example</td>
<td>Example: stepup-login = stepuplogin.html</td>
<td></td>
</tr>
<tr>
<td>certificate-login</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cert-stepup-http = certstepuphttp.html</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WebSEAL displays this HTML page when a client attempts to increase authentication strength level (step-up) to certificates while using HTTP protocol.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This stanza entry is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default value: certstepuphttp.html</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example: cert-stepup-http = certstepuphttp.html</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>switch-user = filename</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch user management form.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: switchuser.html</td>
</tr>
<tr>
<td>Example: switch-user = switchuser.html</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>cert-failure = filename</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page displayed when certificates are required and a client fails to authenticate with a certificate.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: certfailure.html</td>
</tr>
<tr>
<td>Example: cert-failure = certfailure.html</td>
</tr>
</tbody>
</table>

| **client-notify-tod = {yes|no}** |
|----------------------------|
| Enable the display of an error page when authorization is denied due to a POP time of day check. The error page is 38cf08cc.html. |
| This stanza entry is required. |
| Default value: no |
| Example: client-notify-tod = yes |

| **account-expiry-notification = {yes|no}** |
|------------------------------------------|
| Specifies whether WebSEAL informs the user of the reason for a login failure when the failure is due to an invalid or expired account. When this entry is set to no, the user receives the same error message as that which is sent when a login fails due to invalid authentication information, such as an invalid user name or password. |
| This stanza entry is required. |
| Default value: no |
| Example: account-expiry-notification = no |
**Automatic redirect**

- [acnt-mgt]
- [enable-redirects]

<table>
<thead>
<tr>
<th>[acnt-mgt] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>login-redirect-page = <em>Uniform_Resource_Locator</em></td>
</tr>
</tbody>
</table>

Uniform Resource Locator (URL) to which users are automatically redirected after login. The URL can be either server-relative, or can be an absolute URL.

This stanza entry is optional.

There is no default value.

Example of a server relative URL:

login-redirect-page = /jct/page.html

Example of an absolute URL:

login-redirect-page = http://www.ibm.com/

See also "Automatic redirection during user login" on page 212.

<table>
<thead>
<tr>
<th>[enable-redirects] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>redirect = {forms-auth</td>
</tr>
</tbody>
</table>

Enables redirection for use with one or more authentication mechanism. Redirection is supported for:

- Forms authentication
- Basic authentication
- Token authentication

The configuration file must contain a separate entry for each authentication mechanism for which redirection is enabled.

This stanza entry is optional.

There is no default value.

Example entries that enables redirection for forms authentication and basic authentication:

redirect = forms-auth
redirect = basic-auth
Local CGI

- [cgi]
- [cgi-types]
- [cgi-environment-variables]

### [cgi] stanza

**cgi-timeout = number_of_seconds**

| Integer value indicating the timeout, in seconds, for writing to and reading from child CGI processes |
| This setting does not apply to Windows systems |
| This stanza entry is required. |
| Minimum value is 0. This disables the timeout. Disabling the timeout is not recommended. |
| WebSEAL imposes no maximum value. See the discussion on maximum size for integers in “Guidelines for configuring stanzas” on page 378. |
| Example: cgi-timeout = 120 |

### [cgi-types] stanza

**file_extension = command**

| This stanza contains entries that map CGI file extensions to Windows commands. On Windows servers, CGI files that have an extension that is not in this list are not executed, with the exception of files with the .EXE extension. |
| This stanza is used on Windows servers only. |
| There are no default entries. The WebSEAL configuration file contains by default a number of entries that are commented out (inactive). |
| Administrators can add additional entries. Additional entries must consist of ASCII characters. |
| This stanza entry is optional. |
| Examples of uncommented WebSEAL configuration file entries: |
| bat = cmd |
| cmd = cmd |
| pl = perl |
| sh = sh |
| tcl = tclsh76 |

### [cgi-environment-variables] stanza

**ENV = environment_variable**
List containing system environment variables that need to be exported to CGI applications. The administrator can add variables to this list, as needed by specific applications. Entries must be valid environment variable names.

These entries are optional.

The following entries are provided in the default WebSEAL configuration file:

```
ENV = SystemRoot
ENV = PATH
ENV = LANG
ENV = LC_ALL
ENV = LC_CTYPE
ENV = LC_MESSAGES
ENV = LOCSPATH
ENV = NLS\PATH
```

The default WebSEAL configuration file provides one inactive (commented out) variable:

```
#ENV = SystemDrive
```
## Icons

- [content-index-icons]
- [icons]

### [content-index-icons] stanza

<table>
<thead>
<tr>
<th>type = relative_pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries in this stanza specify icons to use in directory indices. The <em>relative_pathname</em> is the path name to the location of the icon.</td>
</tr>
<tr>
<td>The <em>type</em> indicates a wildcard pattern for a collection of MIME types.</td>
</tr>
<tr>
<td>The path name is relative to the WebSEAL protected object space, as set in the doc-root entry in the [content] stanza.</td>
</tr>
<tr>
<td>Administrators can add additional entries. The <em>type</em> must refer to valid MIME types. The wildcard character (<em>) is limited to entries of one collection of MIME types. For example, image/</em>. No further wildcard expansion is done. For a list of MIME types, see the [content-mime-types] stanza.</td>
</tr>
<tr>
<td>The <em>relative_pathname</em> can be any valid URI within the WebSEAL protected object space, as defined is doc-root.</td>
</tr>
<tr>
<td>The entries in this stanza are optional.</td>
</tr>
<tr>
<td>The WebSEAL configuration file provides the following default entries:</td>
</tr>
<tr>
<td>image/* = /icons/image2.gif</td>
</tr>
<tr>
<td>video/* = /icons/movie.gif</td>
</tr>
<tr>
<td>audio/* = /icons/sound2.gif</td>
</tr>
<tr>
<td>text/html = /icons/generic.gif</td>
</tr>
<tr>
<td>text/* = /icons/text.gif</td>
</tr>
<tr>
<td>application/x-tar = /icons/tar.gif</td>
</tr>
<tr>
<td>application/* = /icons/binary.gif</td>
</tr>
</tbody>
</table>

### [icons] stanza

<table>
<thead>
<tr>
<th>diricon = relative_pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative path name to a graphics file to display to indicate a subdirectory. This is used when displaying a directory index.</td>
</tr>
<tr>
<td>The <em>relative_pathname</em> can be any valid URI within the WebSEAL protected object space, as defined is doc-root.</td>
</tr>
<tr>
<td>This stanza entry is required. If the value is not specified, the value of unknownicon is used.</td>
</tr>
<tr>
<td>Default value: /icons/folder2.gif</td>
</tr>
<tr>
<td>Example: diricon = /icons/folder2.gif</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>backicon = relative_pathname</th>
</tr>
</thead>
</table>

Appendix A. WebSEAL configuration file reference 449
<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Relative path name</strong></td>
<td>Relative path name to a graphics file used to indicate the parent directory. This is used when displaying a directory index. The <em>relative_pathname</em> can be any valid URI within the WebSEAL protected object space, as defined is doc-root.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required. If the value is not specified, the value of <code>unknownicon</code> is used. Default value: <code>/icons/back.gif</code> Example: <code>backicon = /icons/back.gif</code></td>
</tr>
<tr>
<td><strong>unknownicon = relative_pathname</strong></td>
<td>Relative path name to a graphics file used to indicate an unknown file type. This is used when displaying a directory index. The <em>relative_pathname</em> can be any valid URI within the WebSEAL protected object space, as defined is doc-root.</td>
</tr>
<tr>
<td></td>
<td>This stanza entry is required. If this value is not specified, a broken link GIF image is displayed. Default value: <code>/icons/unknown.gif</code> Example: <code>unknownicon = /icons/unknown.gif</code></td>
</tr>
</tbody>
</table>
Content caching

<table>
<thead>
<tr>
<th>[content-cache] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIME_type = cache_type:cache_size</td>
</tr>
</tbody>
</table>

List of entries that define the caches which WebSEAL uses to store documents in memory.

- **MIME_type**
  
  Any valid MIME type conveyed in an HTTP Content-Type: response header. This value may contain an asterisk to denote a wildcard (*). A value of */* represents a default object cache that holds any object that does not correspond to an explicitly configured cache.

- **cache_type**
  
  Defines the type of backing store to use for the cache. Only memory caches are supported.

- **cache_size**
  
  The maximum size, in kilobytes, to which the cache grows before objects are removed according to a least-recently-used algorithm. The minimum allowable value is 1 kilobyte. WebSEAL reports an error and fails to start if the value is less than or equal to zero (0). WebSEAL does not impose a maximum allowable value.

This stanza entry is optional.

There are no default caches defined.

Examples:

- `text/html = memory:2000`
- `# image/* = memory:5000`
- `# */* = memory:1000`
## Content compression

- [compress-mime-types]
- [compress-user-agents]

### [compress-mime-types] stanza

```
mime_type = minimum_doc_size:[compression_level]
```

Enables or disables HTTP compression based on the mime-type of the response and the size of the returned document.

The `mime_type` can contain a wild card pattern such as an asterisk (*) for the sub-type, or it can be "/*" to match all mime-types.

The `minimum_doc_size` is an integer than can be positive, negative or zero. A size of -1 means do not compress this mime-type. A size of 0 means to compress the document regardless of its size. A size greater than 0 means to compress document only when its initial size is greater than or equal to `minimum_doc_size`.

The `compression_level` is an integer value between 1 and 9. The larger number results in a higher amount of compression. When `compression-level` is not specified, a default level of 1 is used.

This stanza entry is optional.

There is one default entry:

```
*/* = -1
```

Examples:

```
image/* = -1
text/html = 1000
```

### [compress-user-agents] stanza

```
pattern = {yes|no}
```

Enables or disables HTTP compression based on the user-agent header sent by clients. This entry is used to disable compression for clients which send an "accept-encoding: gzip" HTTP header but don’t actually handle gzip content-encodings properly. An example of a user agent is a browser, such as Microsoft Internet Explorer 6.0.

This stanza entry is optional.

There is no default entry.

Example:

```
*MSIE 6.0* = yes
```
### Content MIME types

<table>
<thead>
<tr>
<th>[content-mime-types] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>extension = MIME_type</strong></td>
</tr>
</tbody>
</table>

This stanza defines the MIME type for specific document extensions. The stanza contains a list of `extension = MIME_type` pairs. Many common MIME types are defined by default. Administrators can add additional entries. Both `extensions` and MIME types must be declared using the ASCII character set.

The entry of invalid MIME types does not affect WebSEAL, but may cause difficulty for client browsers.

- **extension**
  The file name extension of documents of this MIME type

- **MIME_type**
  The corresponding MIME type

There following MIME types are defined by default:

<table>
<thead>
<tr>
<th>Extension</th>
<th>MIME TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>text/html</td>
</tr>
<tr>
<td>htm</td>
<td>text/html</td>
</tr>
<tr>
<td>gif</td>
<td>image/gif</td>
</tr>
<tr>
<td>jpeg</td>
<td>image/jpeg</td>
</tr>
<tr>
<td>ps</td>
<td>application/postscript</td>
</tr>
<tr>
<td>shtml</td>
<td>text/x-server-parsed-html</td>
</tr>
<tr>
<td>jpg</td>
<td>image/jpeg</td>
</tr>
<tr>
<td>jpe</td>
<td>image/jpeg</td>
</tr>
<tr>
<td>mpeg</td>
<td>video/mpeg</td>
</tr>
<tr>
<td>mpe</td>
<td>video/mpeg</td>
</tr>
<tr>
<td>bin</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>exe</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>Z</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>EXE</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>dll</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>DLL</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>4srv</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>pdf</td>
<td>application/pdf</td>
</tr>
<tr>
<td>au</td>
<td>audio/basic</td>
</tr>
<tr>
<td>snd</td>
<td>audio/basic</td>
</tr>
<tr>
<td>aiff</td>
<td>audio/x-aiff</td>
</tr>
<tr>
<td>aifc</td>
<td>audio/x-aiff</td>
</tr>
<tr>
<td>aif</td>
<td>audio/x-aiff</td>
</tr>
<tr>
<td>wav</td>
<td>audio/x-wav</td>
</tr>
<tr>
<td>ai</td>
<td>application/postscript</td>
</tr>
<tr>
<td>eps</td>
<td>application/postscript</td>
</tr>
<tr>
<td>rtf</td>
<td>application/rtf</td>
</tr>
<tr>
<td>zip</td>
<td>application/zip</td>
</tr>
<tr>
<td>ief</td>
<td>image/ief</td>
</tr>
<tr>
<td>tiff</td>
<td>image/tiff</td>
</tr>
<tr>
<td>ras</td>
<td>image/x-cmu-raster</td>
</tr>
<tr>
<td>pnm</td>
<td>image/x-portable-anymap</td>
</tr>
<tr>
<td>pbm</td>
<td>image/x-portable-bitmap</td>
</tr>
<tr>
<td>ppm</td>
<td>image/x-portable-pixmap</td>
</tr>
<tr>
<td>rgb</td>
<td>image/x-rgb</td>
</tr>
<tr>
<td>xbm</td>
<td>image/x-xbitmap</td>
</tr>
<tr>
<td>xpm</td>
<td>image/x-xpmixmap</td>
</tr>
<tr>
<td>xwd</td>
<td>image/x-xwindowdump</td>
</tr>
<tr>
<td>txt</td>
<td>text/plain</td>
</tr>
<tr>
<td>rtx</td>
<td>text/richtext</td>
</tr>
<tr>
<td>tsv</td>
<td>text/tab-separated-values</td>
</tr>
<tr>
<td>etx</td>
<td>text/x-setext</td>
</tr>
<tr>
<td>qt</td>
<td>video/quicktime</td>
</tr>
<tr>
<td>mov</td>
<td>video/quicktime</td>
</tr>
<tr>
<td>avi</td>
<td>video/x-msvideo</td>
</tr>
<tr>
<td>movie</td>
<td>video/x-sgi-movie</td>
</tr>
<tr>
<td>js</td>
<td>application/x-javascript</td>
</tr>
<tr>
<td>ls</td>
<td>application/x-javascript</td>
</tr>
<tr>
<td>mocha</td>
<td>application/x-javascript</td>
</tr>
<tr>
<td>wrl</td>
<td>x-world/x-vrml</td>
</tr>
<tr>
<td>dir</td>
<td>application/x-director</td>
</tr>
<tr>
<td>dcr</td>
<td>application/x-director</td>
</tr>
<tr>
<td>crt</td>
<td>application/x-x509-ca-cert</td>
</tr>
<tr>
<td>tar</td>
<td>application/x-tar</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deftype = MIME_type</th>
</tr>
</thead>
</table>

Default type to assign to pages that don’t match any of the `extension = MIME_type` entries defined in this stanza.

The administrator should not change this value.

This stanza entry is required.

Default value is `text/plain`.

Example: deftype = text/plain
## Content encodings

### [content-encodings] stanza

<table>
<thead>
<tr>
<th>extension = encoding_type</th>
</tr>
</thead>
</table>

Entries in this stanza map a document extension to an encoding type. This mapping is used by WebSEAL to report the correct MIME type in its response content-type header for local junction files. This mapping is necessary so that WebSEAL can communicate to a browser that encoded (binary) data is being returned.

The MIME types defined in this stanza must also be defined in [content-mime-types].

This stanza entry is required.

Default values are:

- `gz = x-gzip`
- `Z = x-compress`

When WebSEAL encounters a document with two extensions, such as: `.txt.Z`, it produces two headers:

- `content-type: text/plain`
- `content-encoding: x-compress`

Thus even though the data is compressed, the response to the browser says `text/plain`. However, the extra content-encoding header tells the browser that the data is compressed `text/plain`.

In most cases, the administrator does not need to add additional entries. However, if the administrator introduces a new extension type that requires more than a text/plain response, the extension and encoding_type should be added to this stanza.
This sections contains the following topics and stanzas:

<table>
<thead>
<tr>
<th>Section</th>
<th>Stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Junction management&quot; on page 456</td>
<td>[junction]</td>
</tr>
<tr>
<td>&quot;Document filtering&quot; on page 460</td>
<td>[filter-url]</td>
</tr>
<tr>
<td>&quot;Event handler filtering&quot; on page 461</td>
<td>[filter-events]</td>
</tr>
<tr>
<td>&quot;Scheme filtering&quot; on page 462</td>
<td>[filter-schemes]</td>
</tr>
<tr>
<td>&quot;MIME types and header filtering&quot; on page 463</td>
<td>[filter-content-types]</td>
</tr>
<tr>
<td></td>
<td>[filter-request-headers]</td>
</tr>
<tr>
<td>&quot;Script filtering&quot; on page 464</td>
<td>[script-filtering]</td>
</tr>
<tr>
<td></td>
<td>[preserve-cookie-names]</td>
</tr>
<tr>
<td>&quot;Credential refresh&quot; on page 466</td>
<td>[credential-refresh-attributes]</td>
</tr>
<tr>
<td>&quot;Header names&quot; on page 466</td>
<td>[header-names]</td>
</tr>
<tr>
<td>&quot;Global Sign-On cache&quot; on page 467</td>
<td>[gso-cache]</td>
</tr>
<tr>
<td>&quot;Lightweight Third Party Authentication cache&quot; on page 469</td>
<td>[ltpa-cache]</td>
</tr>
</tbody>
</table>
**Junction management**

<table>
<thead>
<tr>
<th>[junction] stanza</th>
<th>Relative path name containing the location of the junction database. This value is relative to the value set in server-root key in the [server] stanza.</th>
</tr>
</thead>
</table>
| junction-db = relative_pathname | This stanza entry is required.                                                                                                                             Default value: jct
| Example: junction-db = jct |                                                                                                                                                                                                 |

<table>
<thead>
<tr>
<th>jmt-map = relative_pathname</th>
<th>Relative path name containing the location of the Junction-to- Request Mapping Table (JMT). This value is relative to the value set in server-root key in the [server] stanza.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The administrator can rename this file if necessary. The file name can be any file name valid for the operating system file system.</td>
</tr>
</tbody>
</table>
|                            | This stanza entry is required.                                                                                                                         Default value: lib/jmt.conf
|                            | Example: jmt-map = lib/jmt.conf                                                                                                                        |

<table>
<thead>
<tr>
<th>http-timeout = number_of_seconds</th>
<th>Integer value indicating the timeout, in seconds, for sending to and reading from a TCP junction. The minimum value is 0. When the value is 0, there is no timeout.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WebSEAL does not impose a maximum value. See the discussion of maximum values for integers in &quot;Guidelines for configuring stanzas&quot; on page 378.</td>
</tr>
</tbody>
</table>
|                                 | This stanza entry is required.                                                                                                                         Default value: 120
|                                 | Example: http-timeout = 120                                                                                                                             |

<table>
<thead>
<tr>
<th>https-timeout = number_of_seconds</th>
<th>Integer value indicating the timeout, in seconds, for sending to and reading from a Secure Socket Layer (SSL) junction. The minimum value is 0. When the value is 0, there is no timeout. WebSEAL does not impose a maximum value. See the discussion of maximum values for integers in &quot;Guidelines for configuring stanzas&quot; on page 378.</th>
</tr>
</thead>
</table>
|                                  | This stanza entry is required.                                                                                                                         Default value: 120
|                                  | Example: http-timeout = 120                                                                                                                             |

<table>
<thead>
<tr>
<th>ping-time = number_of_seconds</th>
<th>Integer value indicating the timeout, in seconds, for sending to and reading from a Secure Socket Layer (SSL) junction. The minimum value is 0. When the value is 0, there is no timeout. WebSEAL does not impose a maximum value. See the discussion of maximum values for integers in &quot;Guidelines for configuring stanzas&quot; on page 378.</th>
</tr>
</thead>
</table>
|                              | This stanza entry is required.                                                                                                                         Default value: 120
|                              | Example: http-timeout = 120                                                                                                                             |
Integer value indicating the number of seconds between pings issued by the WebSEAL server. The pings are issued periodically in the background to verify that junctioned WebSEAL servers are running.

The minimum value is 1. WebSEAL does not impose a maximum value. See the discussion of maximum values for integers in “Guidelines for configuring stanzas” on page 378.

This stanza entry is required.
Default value: 300
Example: ping-time = 300

basicauth-dummy-passwd = dummy_password

Global password used when supplying basic authentication data over junctions that were created with the -b supply argument.

Passwords must consist of ASCII characters.

This stanza entry is required.
Default value: dummy
Example: basicauth-dummy-passwd = dummy

worker-thread-hard-limit = number_of_threads

Integer value indicating the limit, expressed as a percentage, of the total worker threads that are to be used for processing requests for junctions. The default value of 100 means that there is no limit.

When the value of worker-thread-hard-limit is less than 100, and the limit is exceeded, WebSEAL generates an error message.

This stanza entry is required.
Default value: 100
Example: worker-thread-hard-limit = 100

See “ Allocating worker threads for junctions (junction fairness)” on page 42 and the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

worker-thread-soft-limit = number_of_threads

Integer value indicating the limit, expressed as a percentage, of the total worker threads that are to be used for processing requests for junctions.

When the value of worker-thread-soft-limit is less than 100, and the limit is exceeded, WebSEAL generates a warning message.

This stanza entry is required.
Default value: 90
Example: worker-thread-soft-limit = 100

See “ Allocating worker threads for junctions (junction fairness)” on page 42 and the IBM Tivoli Access Manager for e-business Performance Tuning Guide.

io-buffer-size = number_of_bytes
### io-buffer-size

Positive integer value indicating the buffer size, in bytes, for reading from and writing to a junction.

The minimum value is 1. WebSEAL does not impose a maximum value. See the discussion of maximum values for integers in **Guidelines for configuring stanzas** on page 378.

This stanza entry is required.

Default value: 4096

Example: \( \text{io-buffer-size} = 4096 \)

### max-webseal-header-size = number_of_bytes

Integer value indicating the maximum size, in bytes, of HTTP headers generated by the WebSEAL server. Headers greater in size than this value are split across multiple HTTP Headers.

A value of zero (0) disables this support. WebSEAL imposes no maximum on this value. See the discussion on maximum values for integer data types in **Guidelines for configuring stanzas** on page 378.

This stanza entry is required.

Default value: 0

Example: \( \text{max-webseal-header-size} = 0 \)

### crl-ldap-server = server_name

Name of the LDAP server to be referenced for Certificate Revocation List (CRL) checking during authentication across SSL junctions.

This stanza entry is optional.

There is no default value.

Example: \( \text{crl-ldap-server} = \text{surf.santacruz.ibm.com} \)

### crl-ldap-server-port = port_number

Port number for communication with the LDAP server specified in \( \text{crl-ldap-server} \). The LDAP server is referenced for Certificate Revocation List (CRL) checking during authentication across SSL junctions.

This stanza entry is optional. When \( \text{crl-ldap-server} \) is set, this stanza entry is required.

There is no default value.

Example: \( \text{crl-ldap-server-port} = 389 \)

### crl-ldap-user = user_DN

Fully qualified distinguished name (DN) of an LDAP user who has permissions to retrieve the Certificate Revocation List.

This stanza entry is optional. A null value for \( \text{crl-ldap-user} \) indicates that the SSL authenticator should bind to the LDAP server anonymously.

There is no default value.

### crl-ldap-user-password = user_password
| **disable-ssl-v2** = {yes|no} |
|--------------------------------|
| Disables support for SSL Version 2 for junction connections. Support for SSL V2 is enabled by default. The value yes means support is disabled. The value no means the support is enabled. |
| This stanza entry is optional. When not specified, the default is no. |
| Default value: no. The WebSEAL configuration sets this value. |
| Example: disable-ssl-v2 = no |

| **disable-ssl-v3** = {yes|no} |
|--------------------------------|
| Disables support for SSL Version 3 for junction connections. Support for SSL V3 is enabled by default. The value yes means support is disabled. The value no means the support is enabled. |
| This stanza entry is optional. When not specified, the default is no. |
| Default value: no. The WebSEAL configuration sets this value. |
| Example: disable-ssl-v3 = no |

| **disable-tls-v1** = {yes|no} |
|--------------------------------|
| Disables support for TLS Version 1 for junction connections. Support for TLS V1 is enabled by default. The value yes means support is disabled. The value no means the support is enabled. |
| This stanza entry is optional. When not specified, the default is no. |
| Default value: no. The WebSEAL configuration sets this value. |
| Example: disable-tls-v1 = no |
## Document filtering

<table>
<thead>
<tr>
<th>HTML_tag</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>HREF</td>
</tr>
<tr>
<td>A</td>
<td>SRC</td>
</tr>
<tr>
<td>APPLET</td>
<td>CODEBASE</td>
</tr>
<tr>
<td>INPUT</td>
<td>SRC</td>
</tr>
<tr>
<td>INPUT</td>
<td>USEMAP</td>
</tr>
<tr>
<td>AREA</td>
<td>HREF</td>
</tr>
<tr>
<td>INS</td>
<td>CITE</td>
</tr>
<tr>
<td>BASE</td>
<td>HREF</td>
</tr>
<tr>
<td>ISINDEX</td>
<td>ACTION</td>
</tr>
<tr>
<td>ISINDEX</td>
<td>HREF</td>
</tr>
<tr>
<td>BGSOUND</td>
<td>SRC</td>
</tr>
<tr>
<td>LAYER</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>BODY</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>LAYER</td>
<td>SRC</td>
</tr>
<tr>
<td>DEL</td>
<td>CITE</td>
</tr>
<tr>
<td>LINK</td>
<td>HREF</td>
</tr>
<tr>
<td>DIV</td>
<td>EMPTYURL</td>
</tr>
<tr>
<td>LINK</td>
<td>SRC</td>
</tr>
<tr>
<td>DIV</td>
<td>IMEAGEPATH</td>
</tr>
<tr>
<td>OBJECT</td>
<td>CODEBASE</td>
</tr>
<tr>
<td>DIV</td>
<td>URL</td>
</tr>
<tr>
<td>OBJECT</td>
<td>DATA</td>
</tr>
<tr>
<td>DIV</td>
<td>VIEWCLASS</td>
</tr>
<tr>
<td>OBJECT</td>
<td>USEMAP</td>
</tr>
<tr>
<td>EMBED</td>
<td>PLUGINSPipe</td>
</tr>
<tr>
<td>Q</td>
<td>CITE</td>
</tr>
<tr>
<td>EMBED</td>
<td>SRC</td>
</tr>
<tr>
<td>SCRIPT</td>
<td>SRC</td>
</tr>
<tr>
<td>FORM</td>
<td>ACTION</td>
</tr>
<tr>
<td>TABLE</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>FRAME</td>
<td>LONGDESC</td>
</tr>
<tr>
<td>TD</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>FRAME</td>
<td>SRC</td>
</tr>
<tr>
<td>TH</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>HEAD</td>
<td>PROFILE</td>
</tr>
<tr>
<td>TR</td>
<td>BACKGROUND</td>
</tr>
<tr>
<td>IFRAME</td>
<td>LONGDESC</td>
</tr>
<tr>
<td>WM:CALENDAR picker</td>
<td>FOLDERURL</td>
</tr>
<tr>
<td>IFRAME</td>
<td>SRC</td>
</tr>
<tr>
<td>WM:CALENDAR picker</td>
<td>IMAGEPREVARROW</td>
</tr>
<tr>
<td>IFRAME</td>
<td>SRC</td>
</tr>
<tr>
<td>WM:CALENDAR VIEW</td>
<td>FOLDERURL</td>
</tr>
<tr>
<td>IFRAME</td>
<td>SRC</td>
</tr>
<tr>
<td>WM:CALENDAR picker</td>
<td>IMAGENEXTARROW</td>
</tr>
<tr>
<td>LAYER</td>
<td>SRC</td>
</tr>
<tr>
<td>WM:MESSAGE</td>
<td>DRAFTSURL</td>
</tr>
<tr>
<td>IMG</td>
<td>SRC</td>
</tr>
<tr>
<td>WM:MESSAGE</td>
<td>URL</td>
</tr>
<tr>
<td>IMG</td>
<td>LOWSRC</td>
</tr>
<tr>
<td>WM:MESSAGE</td>
<td>URL</td>
</tr>
<tr>
<td>IMG</td>
<td>LONGDESC</td>
</tr>
<tr>
<td>WM:MESSAGE</td>
<td>Folder</td>
</tr>
<tr>
<td>IMG</td>
<td>USEMAP</td>
</tr>
<tr>
<td>WM:REMINDER</td>
<td>Folder</td>
</tr>
<tr>
<td>IMG</td>
<td>DYNsrc</td>
</tr>
<tr>
<td>?IMPORT</td>
<td>IMPLEMENTATION</td>
</tr>
</tbody>
</table>

List of URL attributes that WebSEAL server filters in responses from junctioned servers.

Administrators can add additional entries when necessary. New entries must consist of valid HTML tags and attributes. When adding new entries, maintain alphabetical order.

This list is required. Although not all tags are required by all applications, the unused tags do no harm. The recommended practice is to leave the default entries in this list.

Default HTML tags and attributes:
### Event handler filtering

**[filter-events] stanza**

<table>
<thead>
<tr>
<th>HTML_tag</th>
<th>event_handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ONCLICK</td>
</tr>
<tr>
<td>A</td>
<td>ONDBLCLICK</td>
</tr>
<tr>
<td>A</td>
<td>ONMOUSEDOWN</td>
</tr>
<tr>
<td>A</td>
<td>ONMOUSEOUT</td>
</tr>
<tr>
<td>A</td>
<td>ONMOUSEOVER</td>
</tr>
<tr>
<td>A</td>
<td>ONMOUSEUP</td>
</tr>
<tr>
<td>AREA</td>
<td>ONCLICK</td>
</tr>
<tr>
<td>AREA</td>
<td>ONMOUSEOUT</td>
</tr>
<tr>
<td>AREA</td>
<td>ONMOUSEOVER</td>
</tr>
<tr>
<td>BODY</td>
<td>ONBLUR</td>
</tr>
<tr>
<td>BODY</td>
<td>ONCLICK</td>
</tr>
<tr>
<td>BODY</td>
<td>ONDRAGDROP</td>
</tr>
<tr>
<td>BODY</td>
<td>ONFOCUS</td>
</tr>
<tr>
<td>BODY</td>
<td>ONKEYDOWN</td>
</tr>
<tr>
<td>BODY</td>
<td>ONKEYPRESS</td>
</tr>
<tr>
<td>BODY</td>
<td>ONKEYUP</td>
</tr>
<tr>
<td>BODY</td>
<td>ONLOAD</td>
</tr>
<tr>
<td>BODY</td>
<td>ONMOUSEDOWN</td>
</tr>
<tr>
<td>BODY</td>
<td>ONMUSEUP</td>
</tr>
<tr>
<td>BODY</td>
<td>ONMOVE</td>
</tr>
<tr>
<td>BODY</td>
<td>ONRESIZE</td>
</tr>
<tr>
<td>BODY</td>
<td>ONULOAD</td>
</tr>
<tr>
<td>FORM</td>
<td>ONSUBMIT</td>
</tr>
<tr>
<td>FORM</td>
<td>ONSIZE</td>
</tr>
<tr>
<td>FORM</td>
<td>ONULOAD</td>
</tr>
<tr>
<td>FRAME</td>
<td>ONBLUR</td>
</tr>
<tr>
<td>FRAME</td>
<td>ONDRAGDROP</td>
</tr>
<tr>
<td>FRAME</td>
<td>ONFOCUS</td>
</tr>
<tr>
<td>FRAME</td>
<td>ONLOAD</td>
</tr>
<tr>
<td>FRAME</td>
<td>ONMOVE</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td>ONBLUR</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td>ONCHANGE</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td>ONFOCUS</td>
</tr>
</tbody>
</table>

List of HTML tags used by WebSEAL to identify and filter absolute URLs embedded in JavaScript. JavaScript allows HTML tags to contain *event handlers* that are invoked when certain events occur. For example, the HTML tag:

```html
<form onsubmit="javascript:doSomething()"
```

causes the JavaScript function `doSomething()` to be called when the form is submitted.

The entries in this stanza are used to identify HTML tags that may contain JavaScript code. When such a tag is discovered, WebSEAL searches the tag to filter any absolute URLs embedded in the JavaScript. For example, if the "form onsubmit" example looked like:

```html
<form onsubmit="javascript:doSomething('http://junction.server.com')"
```

WebSEAL HTML filtering would modify the tag to look like:

```html
<form onsubmit="javascript:doSomething('/junction')"
```

Administrators can add additional entries when necessary. New entries must consist of valid HTML tags that are built into JavaScript. When adding new entries, maintain alphabetical order.

This list is required. Although not all tags are required by all applications, the unused tags do no harm. The recommended practice is to leave the default entries in this list.

Default HTML tags and event handlers:

| A         | ONCLICK |
| A         | ONDBLCLICK |
| A         | ONMOUSEDOWN |
| A         | ONMOUSEOUT |
| A         | ONMOUSEOVER |
| A         | ONMOUSEUP |
| AREA      | ONCLICK |
| AREA      | ONMOUSEOUT |
| AREA      | ONMOUSEOVER |
| BODY      | ONBLUR |
| BODY      | ONCLICK |
| BODY      | ONDRAGDROP |
| BODY      | ONFOCUS |
| BODY      | ONKEYDOWN |
| BODY      | ONKEYPRESS |
| BODY      | ONKEYUP |
| BODY      | ONLOAD |
| BODY      | ONMOUSEDOWN |
| BODY      | ONMUSEUP |
| BODY      | ONMOVE |
| BODY      | ONRESIZE |
| BODY      | ONULOAD |
| FORM      | ONSUBMIT |
| FORM      | ONSIZE |
| FORM      | ONULOAD |
| FRAME     | ONBLUR |
| FRAME     | ONDRAGDROP |
| FRAME     | ONFOCUS |
| FRAME     | ONLOAD |
| FRAME     | ONMOVE |
## Scheme filtering

### [filter-schemes] stanza

<table>
<thead>
<tr>
<th>scheme = scheme_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of URL schemes that are not to be filtered by WebSEAL. A scheme is a protocol identifier.</td>
</tr>
</tbody>
</table>

This list is utilized when WebSEAL encounters a document containing a base URL. For example:

```html
<head>
  <base href="http://www.foo.com">
</head>
<a href="mailto:bee@bee.com">Send me mail</a>
```

WebSEAL identifies the scheme `mailto` because this scheme is included by default in the `[filter-schemes] stanza`. If `mailto` was not identified as a scheme, WebSEAL would interpret it as document and perform normal filtering. WebSEAL would then rewrite the link as:

```html
<a href="http://www.foo.com/mailto:bee@bee.com"
```

This would be incorrect.

WebSEAL provides a set of default schemes. The administrator can extend the list if additional protocols are used. Recommended practice is to not delete entries from this list.

Default list entries:

- `scheme = file`
- `scheme = ftp`
- `scheme = https`
- `scheme = mailto`
- `scheme = news`
- `scheme = telnet`
### MIME types and header filtering

- [filter-content-types]
- [filter-request-headers]

**[filter-content-types] stanza**

<table>
<thead>
<tr>
<th>type = type_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of entries that specify MIME types to be filtered by WebSEAL when received from junctioned servers.</td>
</tr>
<tr>
<td>Administrators can add additional MIME types that refer to a document that contains HTML or HTML-like content.</td>
</tr>
<tr>
<td>This list of stanza entries is required. Do not remove the default entries.</td>
</tr>
<tr>
<td>Default list entries:</td>
</tr>
<tr>
<td>type = text/html</td>
</tr>
<tr>
<td>type = text/vnd.wap.wml</td>
</tr>
</tbody>
</table>

See "Standard URL filtering rules for WebSEAL" on page 288

**[filter-request-headers] stanza**

<table>
<thead>
<tr>
<th>header = header_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of HTTP headers that WebSEAL filters before sending the request to a junctioned server. A default list is built-in to WebSEAL. The default entries are not included in the configuration file.</td>
</tr>
<tr>
<td>Default list:</td>
</tr>
<tr>
<td>host connection proxy-connection expect te iv-ssl-jct iv-user iv_user iv-groups iv_groups iv-creds iv_creds iv_remote_address iv-remote-address</td>
</tr>
<tr>
<td>The addition of new entries in this stanza is optional. For example, an administrator could add the header accept-encoding. This would instruct WebSEAL to remove any accept-encoding headers from requests before forwarding the request to the junction. The removal of the accept-encoding header would cause the junction server to return the document in an unencoded form, allowing WebSEAL to filter the document if necessary.</td>
</tr>
<tr>
<td>New entries must consist of valid HTTP headers.</td>
</tr>
</tbody>
</table>
### Script filtering

- [script-filtering]
- [preserve-cookie-names]
- [illegal-url-substrings]

#### [script-filtering] stanza

```plaintext
script-filter = {yes|no}
```

Enables or disables script filtering support. A value of yes means enabled. A value of no means disabled. When enabled, WebSEAL can filter absolute URLs encountered in scripts such as JavaScript.

This stanza entry is optional, but is included by default. When it is not declared, the value for the script-filter functionality is no by default.

Default value: no

Example: script-filter = no

See “Modifying absolute URLs with script filtering” on page 290.

#### rewrite-absolute-with-absolute = {yes|no}

Enables WebSEAL to rewrite absolute URLs with new absolute URLs that contain the protocol, host, and port (optionally) that represent how the user accessed the WebSEAL server.

This stanza entry is optional.

There is no default value, but if the entry is not specified in this configuration file, WebSEAL assumes the value is no.

Example: rewrite-absolute-with-absolute = no

See “Modifying absolute URLs with script filtering” on page 290.

#### hostname-junction-cookie = {yes|no}

Enables WebSEAL to uniquely identify the cookie used for resolving unfiltered links. This is used when another WebSEAL server has created a junction to this WebSEAL server, using a WebSEAL to WebSEAL junction.

This stanza entry is optional, but it is included by default in the configuration file.

Default value: no

Example: hostname-junction-cookie = no

---

#### [preserve-cookie-names] stanza

```plaintext
name = cookie_name
```
List of specific cookie names that WebSEAL must not modify.

WebSEAL, by default, modifies the names of cookies returned in responses from junctions created with pdadmin using -j flag. WebSEAL also by default modifies the name of cookies listed in the junction mapping table (JMT). This default modification is done to prevent naming conflicts with cookies returned by other junctions.

When a front-end application depends on the names of specific cookies, the administrator can disable the modification of cookie names for those specific cookies. The administrator does this by listing the cookies in this stanza.

When entering a value for cookie_name, use ASCII characters.

There are no cookie names set by default.

Example:
Name = JSESSIONID

### [[illegal-url-substrings] stanza]

**substring = string**

WebSEAL blocks HTTP requests containing any of the substrings specified here.

This stanza entry is optional.

Default entry:
substring = <script
**Credential refresh**

### [credential-refresh-attributes] stanza

**authentication_level = {preserve|refresh}**

Specifies whether the authentication level for the user should be preserved or refreshed during a credential refresh. The authentication level can reflect the results of an authentication strength policy (step-up authentication). In most cases, this level should be preserved during a credential refresh.

This stanza entry is required.

Default value: preserve

Example: authentication_level = preserve

### [attribute_name_pattern] stanza

**attribute_name_pattern = {preserve|refresh}**

Specifies whether an attribute, or group of attributes that match a pattern, should be preserved or refreshed during a credential refresh.

This stanza entry is optional.

Default entry: tagvalue_* = preserve

Example: tagvalue_* = preserve

---

**Header names**

### [header-names] stanza

**server-name = {iv-server-name|no value}**

Specifies the name of the authorization API administration server used with the server task command to junctioned application. The name is passed in a header to the junctioned server.

To disable the header, omit any value from this key:

server-name =

This stanza entry is required.

Default entry: server-name = iv-server-name

Example: server-name = iv-server-name
## Global Sign-On cache

### [gso-cache] stanza

| gso-cache-enabled | {yes|no} |
|-------------------|---------|
| Enables or disables the Global Sign-on (GSO) cache. This stanza entry is required. Default value: no |
| Example: gso-cache-enabled = no |

### gso-cache-size = number of entries

Integer value indicating the number of entries allowed in the GSO cache. The value must be greater than or equal to zero. Zero means that there is no limit on the size of the GSO cache. This is not recommended.

WebSEAL does not impose a maximum value. Choose your maximum value to stay safely within the bounds of your available system memory. See the discussion of maximum values for integer types in "Guidelines for configuring stanzas" on page 378.

This stanza entry is required, but is ignored when GSO caching is disabled.

Default value: 1024

Example: gso-cache-size = 1024

### gso-cache-entry-lifetime = number of seconds

Integer value that specifies the lifetime, in seconds, of a GSO cache entry. The value must be greater than or equal to zero (0). A value of 0 means that entries are not removed from the GSO cache due to their entry lifetime being exceeded. However, they may still be removed due to either the gso-cache-size being exceeded or the gso-cache-entry-idle-timeout stanza entry being exceeded.

WebSEAL does not impose a maximum value. See the discussion of maximum values for integer types in "Guidelines for configuring stanzas" on page 378.

This stanza entry is required, but is ignored when GSO caching is disabled.

Default value: 900

Example: gso-cache-entry-lifetime = 900

### gso-cache-entry-idle-timeout = number of seconds

Examples: 900
| Integer value that specifies the timeout, in seconds, for cache entries that are idle. The value must be greater than or equal to zero (0). A value of 0 means that entries are not removed from the GSO cache due to inactivity. However, they may still be removed due to either the gso-cache-size being exceeded or the gso-cache-entry-lifetime stanza entry being exceeded.

WebSEAL does not impose a maximum value. See the discussion of maximum values for integer types in “Guidelines for configuring stanzas” on page 378.

This stanza entry is required, but is ignored when GSO caching is disabled.

Default value: 120

Example: gso-cache-entry-idle-timeout = 120 |
## Lightweight Third Party Authentication cache

### [ltpa-cache] stanza

**ltpa-cache-enabled** = {yes|no}

| Enables or disables the Lightweight Third Party Authentication cache. A value of yes enables caching. A value of no disables caching. |
| This stanza entry is required. |
| Default value: yes |
| Example: ltpa-cache-enabled = yes |

**ltpa-cache-size** = *number_of_entries*

| Integer value indicating the number of entries allowed in the LTPA cache. The value must be greater than or equal to zero. Zero means that there is no limit on the size of the LTPA cache. This is not recommended. |
| WebSEAL does not impose a maximum value. Choose your maximum value to stay safely within the bounds of your available system memory. See the discussion of maximum values for integer types in "Guidelines for configuring stanzas" on page 378. |
| This stanza entry is required, but is ignored when LTPA caching is disabled. |
| Default value: 4096 |
| Example: ltpa-cache-size = 4096 |

**ltpa-cache-entry-lifetime** = *number_of_seconds*

| Integer value that specifies the lifetime, in seconds, of a LTPA cache entry. The value must be greater than or equal to zero (0). A value of 0 means that entries are not removed from the LTPA cache due to their entry lifetime being exceeded. However, they may still be removed due to either the ltpa-cache-size being exceeded or the ltpa-cache-entry-idle-timeout stanza entry being exceeded. |
| WebSEAL does not impose a maximum value. See the discussion of maximum values for integer types in "Guidelines for configuring stanzas" on page 378. |
| This stanza entry is required, but is ignored when LTPA caching is disabled. |
| Default value: 3600 |
| Example: ltpa-cache-entry-lifetime = 3600 |

**ltpa-cache-entry-idle-timeout** = *number_of_seconds*
Integer value that specifies the timeout, in seconds, for cache entries that are idle. The value must be greater than or equal to zero (0). A value of 0 means that entries are not removed from the LTPA cache due to inactivity. However, they may still be removed due to either the ltpa-cache-size being exceeded or the ltpa-cache-entry-lifetime stanza entry being exceeded.

WebSEAL does not impose a maximum value. See the discussion of maximum values for integer types in "Guidelines for configuring stanzas" on page 378.

This stanza entry is required, but is ignored when LTPA caching is disabled.

Default value: 600

Example: gso-cache-entry-idle-timeout = 600
## Logging

<table>
<thead>
<tr>
<th>[logging] stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>server-log = <em>fully_qualified_path</em></td>
</tr>
<tr>
<td>Fully qualified path to the server error log file.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default location is <code>log/webseald.log</code>, located under the WebSEAL installation directory.</td>
</tr>
<tr>
<td>Example on UNIX:</td>
</tr>
<tr>
<td><code>server-log = /var/pdweb/log/msg_webseald.log</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>max-size = <em>number_of_bytes</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer value indicating the size limit of the log files. The size limit is also referred to as the <em>rollover threshold</em>. When the log file reaches this threshold, the original log file is renamed and a new log file with the original name will be created.</td>
</tr>
<tr>
<td>When the value is zero (0), no rollover log file is created.</td>
</tr>
<tr>
<td>When the value is a negative integer, the logs are rolled over daily, regardless of the size.</td>
</tr>
<tr>
<td>When the value is a positive integer, the value indicates the maximum size, in bytes, of the log file before the rollover occurs. The allowable range is from 1 byte to 2 megabytes</td>
</tr>
<tr>
<td>An integer value indicating the size limit of log files. This value applies to the request, referer, and agent logs.</td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td>Default value: 2000000</td>
</tr>
<tr>
<td>Example: <code>max-size = 2000000</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>flush-time = <em>number_of_seconds</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer value indicating the frequency, in seconds, to force a flush of log buffers.</td>
</tr>
<tr>
<td>The minimum value is 1 second.</td>
</tr>
<tr>
<td>The maximum value is 600 seconds.</td>
</tr>
<tr>
<td>This stanza entry is optional.</td>
</tr>
<tr>
<td>Default value: 20</td>
</tr>
<tr>
<td>Example: <code>flush-time = 20</code></td>
</tr>
</tbody>
</table>

| requests = {yes|no} |
|---------------------|
| This stanza entry is optional. |
Enables or disables the requests log. This log records standard logging of HTTP requests. The value yes enables requests logging. The value no disables requests logging.

This stanza entry is required.

Default value: yes

Example: requests = yes

For more information, see “Legacy auditing for HTTP” on page 105.

**requests-file = fully_qualified_path**

- Fully qualified path to the request log file.
- This stanza entry is required.
- The default location is `www-instance/log/request.log`, located under the WebSEAL installation directory.
- Example on UNIX:
  
  requests-file = /var/pdweb/www-web1/log/request.log

**referers = {yes|no}**

- Enables or disables the referers log. This log records the Referer: header of each HTTP request. The value yes enables referers logging. The value no disables referers logging.
- This stanza entry is required.
- Default value: yes
- Example: referers = yes
- For more information, see “Legacy auditing for HTTP” on page 105.

**referers-file = fully_qualified_path**

- Fully qualified path to the referers log file.
- This stanza entry is required.
- The default location is `www-instance/log/referer.log`, located under the WebSEAL installation directory.
- Example:
  
  referers-file = /var/pdweb/www-web1/log/referer.log

**agents = {yes|no}**

- Enables or disables the agents log. This log records the contents of the User-Agent: header of each HTTP request. The value yes enables agents logging. The value no disables agents logging.
- This stanza entry is required.
- Default value: yes
- Example: agents = yes
- For more information, see “Legacy auditing for HTTP” on page 105.

**agents-file = fully_qualified_path**
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agents-file</td>
<td>Fully qualified path to the agents log file. This stanza entry is required.</td>
</tr>
<tr>
<td></td>
<td>The default location is <code>www-instance/log/agent.log</code>, located under the WebSEAL installation directory. Example: agents-file = <code>/var/pdweb/www-web1/log/agent.log</code></td>
</tr>
<tr>
<td>gmt-time</td>
<td>Enables or disables logging requests using Greenwich Mean Time (GMT) instead of the local timezone. A value of <code>yes</code> means to use GMT. A value of <code>no</code> means to use the local timezone. This stanza entry is required. Default value: <code>no</code> Example: gmt-time = <code>no</code></td>
</tr>
</tbody>
</table>
## Auditing

**[aznapi-configuration] stanza**

<table>
<thead>
<tr>
<th>logclientid = webseald</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the daemon whose activities are audited through use of authorization API logging.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: webseald</td>
</tr>
<tr>
<td>Example: logclientid = webseald</td>
</tr>
</tbody>
</table>

**logsize = number_of_bytes**

| Integer value indicating the size limit of audit log files. The size limit is also referred to as the *rollover threshold*. When the audit log file reaches this threshold, the original audit log file is renamed and a new log file with the original name will be created. |
| When the value is zero (0), no rollover log file is created. |
| When the value is a negative integer, the logs are rolled over daily, regardless of the size. |
| When the value is a positive integer, the value indicates the maximum size, in bytes, of the audit log file before the rollover occurs. The allowable range is from 1 byte to 2 megabytes |
| This stanza entry is optional. |
| Default value: 2000000 |
| Example: max-size = 2000000 |

**logflush = number_of_seconds**

| Integer value indicating the frequency, in seconds, to force a flush of log buffers. |
| The minimum value is 1 second. |
| The maximum value is 600 seconds. |
| This stanza entry is optional. |
| Default value: 20 |
| Example: logflush = 20 |

**logaudit = (yes|true|no|false)**

| Enables or disables auditing. The values yes or true enable auditing. The values no or false disable auditing. |
| By default, when auditing is enabled using this key, and no configured audit events are specified by auditcfg, all auditable events are captured. |
| This stanza entry is required. |
| Default value: no |
| Example: logaudit = no |

**auditlog = fully_qualified_path**
Location of the audit trail file for WebSEAL.

The fully qualified path value represents an alphanumeric string.

This stanza entry is required when auditing is enabled.

The default value is set during WebSEAL configuration by appending `log/aznapi_webseald.log` to the path for the WebSEAL installation directory.

Example on UNIX:

```
auditlog = /var/pdweb/log/aznapi_webseald.log
```

### `auditcfg = {azn|authn|http}`

Indicates the components for which auditing of events is configured. To enable component specific audit records, add the appropriate definition.

Valid values are:

- **azn**: Capture authorization events.
- **authn**: Capture authentication events.
- **http**: Capture HTTP events. These correspond to the events logged by the request, referer, and agent logging clients.

This stanza entry is optional for WebSEAL. However, this stanza entry is required when auditing is enabled (`logaudit = yes`).

There is no default value for WebSEAL, because auditing is disabled by default.

Create a separate stanza entry for each component to be activated. The components are included in the default configuration file but are commented out. To activate a commented out entry, remove the pound sign (`#`) from the start of the entry.

Example:

```
auditcfg = azn
#auditcfg = authn
#auditcfg = http
```

### `logcfg = category:{stdout|stderr|file|pipe|remote}[ [parameter=value ] [ ,parameter=value]...]`

Appendix A. WebSEAL configuration file reference 475
Specifies event logging for the specified category.

For WebSEAL, the categories are:

- **azn**
  Authorization events
- **authn**
  Credentials acquisition authentication.
- **http**
  All HTTP logging information
- **http.clf**
  HTTP request information in common log format.
- **http.ref**
  HTTP Referer header information
- **http.agent**
  HTTP User_Agent header information
- **http.cof**
  NCSA combined format which captures HTTP request information (with time stamp) and appends the quoted referer and agent strings to the standard common log format.

Event logging supports a number of output destination types:

{stdout|stderr|file|pipe|remote}

WebSEAL auditing typically is configured to use the file type.

Each event logging type supports a number of optional parameter = value options.

For more information about output destination types and optional parameter = value settings, see the IBM Tivoli Access Manager Base Administration Guide.

This stanza entry is optional.

There is no default value.

Example entry for request.log (common log format) — entered as one long line:

```
logcfg = http.clf:file path=request_file, \ 
flush=time, rollover=max_size, \ 
log=clf,buffer_size=8192,queue_size=48
```

For more information, see “Event capturing and logging” on page 98.
### Policy database

**[aznapi-configuration] stanza**

<table>
<thead>
<tr>
<th>db-file = fully_qualified_path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully qualified path to the WebSEAL policy database cache file.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value is a path name that has <code>db/webseald.db</code> appended to the WebSEAL installation directory path.</td>
</tr>
<tr>
<td>Example on UNIX: db-file = <code>/var/pdweb/db/webseald.db</code></td>
</tr>
</tbody>
</table>

| cache-refresh-interval = {disable|default|number_of_seconds} |
|-------------------------------------------------------------|
| Poll interval between checks for updates to the master authorization server. |
| Valid values are: |
| **disable** The interval value in seconds is not set. |
| **default** When value is to default, an interval of 600 seconds is used. |
| **num_seconds** Integer value indicating the number of seconds between polls to the master authorization server to check for updates. |
| The minimum number of seconds is 0. There is no maximum value. See the discussion of maximum values for integer data types in "Guidelines for configuring stanzas" on page 378. |
| This stanza entry is optional. |
| The default value for WebSEAL is disable |
| Example: cache-refresh-interval = disable |

| listen-flags = {enable|disable} |
|---------------------------------|
| Enables or disables the reception by WebSEAL of policy cache update notifications from the master authorization server. A value of enable activates the notification listener. A value of disable deactivates the notification listener. |
| This stanza entry is required. |
| Default value: disable |
| Example: listen-flags = enable |

<table>
<thead>
<tr>
<th>mode = local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the authorization API client mode for the WebSEAL server. Only local mode is supported. This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration and should not be changed by the administrator.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value: local</td>
</tr>
<tr>
<td>Example: mode = local</td>
</tr>
</tbody>
</table>

| policy-cache-size = cache_size |

---

Appendix A. WebSEAL configuration file reference 477
The maximum size of the in-memory policy cache is configurable. The cache consists of policy and the relationships between policy and resources. The knowledge that a resource has no directly associated policy is also cached.

The maximum cache size should be relative to the number of policy objects defined and the number of resources protected and the available memory.

A reasonable algorithm to begin with is: (number of policy objects * 3) + (number of protected resources * 3)

This value controls how much information is cached. A larger cache will potentially improve the application performance but use additional memory as well.

Size is specified as the number of entries.

This stanza entry is optional.

Default value: none

Example: `policy-cache-size = 32768`

<table>
<thead>
<tr>
<th>azn-server-name = <code>webseal_server_name</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the WebSEAL server name for use when contacting the policy server as an authorization API client.</td>
</tr>
<tr>
<td>This stanza entry is set during WebSEAL configuration and is typically not changed by the administrator.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value consists of a combination of <code>webseald</code> with the hostname, separated by a hyphen (-):</td>
</tr>
<tr>
<td>Example: <code>azn-server-name = webseald-surf</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pd-user-name = <code>webseal_server_identity</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tivoli Access Manager identity of the WebSEAL server.</td>
</tr>
<tr>
<td>This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration and should not be changed by the administrator.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value consists of a combination of <code>webseald</code> with the hostname, separated by a forward slash (/):</td>
</tr>
<tr>
<td>Example: <code>pd-user-name = webseald/surf</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pd-user-pwd = <code>password</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Password for the authorization API client identity. This identity represents the WebSEAL server daemon.</td>
</tr>
<tr>
<td>This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration and should not be changed by the administrator.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value:</td>
</tr>
<tr>
<td>Example: <code>pd-user-pwd = ZsLuBKSo</code></td>
</tr>
</tbody>
</table>
## Entitlement services

<table>
<thead>
<tr>
<th>aznapi-entitlement-services stanza</th>
</tr>
</thead>
<tbody>
<tr>
<td>service-id = library_base_name</td>
</tr>
</tbody>
</table>

The Tivoli Access Manager authorization API provides a framework for adding entitlement services into the authorization decision making process. The authorization API obtains knowledge of active entitlement services by reading entries from stanza files, such as this one, and by reading initialization stanza entries that are sent to the API upon startup.

WebSEAL uses a built-in entitlement service that is supplied as a shared library. This configuration file entry provides a `service-id` of AZN_ENT_EXT_ATTR.

**Note:** The Authorization API uses the `service-id` to denote the presence of a service that is to be loaded at API initialization time. For more information, see *IBM Tivoli Access Manager for e-business Authorization C API Developer Reference*.

This configuration file entry also specifies the name of the shared library: azn_ent_ext_attr.

The file name of the azn_ent_ext_attr shared library, and its location within the file system, is specific to each operating system. For example, on Windows platforms, the names of shared libraries contain the suffix .dll. However, the base name for the library is common across operating systems. This value is specified in `library_base_name`.

WebSEAL reads the `library_base_name` and then uses an internal search algorithm to find the appropriate shared library by cycling through the known prefixes, suffixes, and file locations.

This stanza entry is required. Administrator should not change this entry.

For more information on entitlement services, see the *IBM Tivoli Access Manager for e-business Authorization C API Developer Reference*

WebSEAL contains one default entry for an entitlement service:

AZN_ENT_EXT_ATTR = azn_ent_ext_attr
## Policy server

- [policy-director]
- [manager]

### [policy-director] stanza

<table>
<thead>
<tr>
<th>config-file = /opt/PolicyDirector/etc/pd.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path name to the configuration file for the Tivoli Access Manager policy server for the domain in which the WebSEAL server is configured. This stanza entry is set by WebSEAL configuration and should not be modified.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>Default value of the configuration file path consists of the etc/pd.conf appended to the Tivoli Access Manager installation directory.</td>
</tr>
<tr>
<td>Example on UNIX:</td>
</tr>
<tr>
<td>config-file = /opt/PolicyDirector/etc/pd.conf</td>
</tr>
</tbody>
</table>

### [manager] stanza

<table>
<thead>
<tr>
<th>master-host = policy_server_hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname of the Tivoli Access Manager policy server for the domain to which this WebSEAL server belongs.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value. This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration. Administrators should not modify this stanza entry.</td>
</tr>
<tr>
<td>Example: master-host = server77</td>
</tr>
<tr>
<td>master-port = port_number</td>
</tr>
<tr>
<td>TCP port that the policy server host uses to communicate with WebSEAL.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value. This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration. Administrators should not modify this stanza entry.</td>
</tr>
<tr>
<td>Example: master-port = 7135</td>
</tr>
<tr>
<td>master-dn = LDAP_DN</td>
</tr>
<tr>
<td>Fully qualified DN of the policy server daemon for this Tivoli Access Manager secure domain.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>There is no default value. This stanza entry is set by <code>svrsslcfg</code> during WebSEAL configuration. Administrators should not modify this stanza entry.</td>
</tr>
<tr>
<td>Example:</td>
</tr>
<tr>
<td>master-dn = CN=ivmgrd/master,O=Policy Director,C=US</td>
</tr>
</tbody>
</table>
## Platform for Privacy Preferences (P3P)

- [server]
- [p3p-header]

### [server] stanza

| preserve-p3p-policy = {yes|no} |
|-------------------------------|
| Specifies whether to replace or preserve P3P headers from junctioned servers. The values yes means that headers are preserved. A value of no means that headers are replaced. |
| This stanza entry is required. |
| The default value is no. |
| Example: preserve-p3p-policy = no |

### [p3p-header] stanza

<table>
<thead>
<tr>
<th>p3p-element = policyref=location_of_policy_reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies elements to add to the P3P header in addition to the elements specified by the other configuration items in this stanza. Typically this is done by referring to the location of a full XML policy. The default entry points to a default policy reference located on the World Wide Web Consortium Web site.</td>
</tr>
<tr>
<td>This stanza entry is required.</td>
</tr>
<tr>
<td>The default value is policyref=&quot;/w3c/p3p.xml&quot;.</td>
</tr>
<tr>
<td>Example:</td>
</tr>
<tr>
<td>p3p-element = policyref=&quot;/w3c/p3p.xml&quot;</td>
</tr>
</tbody>
</table>

<p>| access = {none|all|nonident|contact-and-other|ident-contact|other-ident} |</p>
<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No access to identified data is given.</td>
</tr>
<tr>
<td>all</td>
<td>Access is given to all identified data</td>
</tr>
<tr>
<td>contact-and-other</td>
<td>Access is given to identified online and physical contact information as well as to certain other identified data.</td>
</tr>
<tr>
<td>ident-contact</td>
<td>Access is given to identified online and physical contact information. For example, users can access things such as a postal address.</td>
</tr>
<tr>
<td>nonident</td>
<td>Web site does not collect identified data.</td>
</tr>
<tr>
<td>other-ident</td>
<td>Access is given to certain other identified data. For example, users can access things such as their online account charges.</td>
</tr>
</tbody>
</table>

This stanza entry is required.

The default value is **none**.

Example:
```
access = none
```
<table>
<thead>
<tr>
<th>categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical</td>
</tr>
<tr>
<td>online</td>
</tr>
<tr>
<td>uniqueid</td>
</tr>
<tr>
<td>purchase</td>
</tr>
<tr>
<td>financial</td>
</tr>
<tr>
<td>computer</td>
</tr>
<tr>
<td>navigation</td>
</tr>
<tr>
<td>interactive</td>
</tr>
<tr>
<td>demographic</td>
</tr>
<tr>
<td>content</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>political</td>
</tr>
<tr>
<td>health</td>
</tr>
<tr>
<td>preference</td>
</tr>
<tr>
<td>location</td>
</tr>
<tr>
<td>government</td>
</tr>
<tr>
<td>other-category</td>
</tr>
</tbody>
</table>

Specifies the type of information stored in the cookie or linked to by the cookie. When the non-identifiable option is set to yes, then no categories need be configured.

Possible values are:

- **physical**
  Information that allows an individual to be contacted or located in the physical world. For example, telephone number or address.
- **online**
  Information that allows an individual to be contacted or located on the Internet.
- **uniqueid**
  Non-financial identifiers, excluding government-issued identifiers, issued for purposes of consistently identifying or recognizing the individual.
- **purchase**
  Information actively generated by the purchase of a product or service, including information about the method of payment.
- **financial**
  Information about an individual’s finances including account status and activity information such as account balance, payment or overdraft history, and information about an individual’s purchase or use of financial instruments including credit or debit card information.
- **computer**
  Information about the computer system that the individual is using to access the network. For example, IP number, domain name, browser type or operating system.
- **navigation**
  Data passively generated by browsing the Web site. For example, which pages are visited, and how long users stay on each page.
- **interactive**
  Data actively generated from or reflecting explicit interactions with a service provider through its site. For example, queries to a search engine, or logs of account activity.
- **demographic**
  Data about an individual’s characteristics. For example, gender, age, and income.
- **content**
  The words and expressions contained in the body of a communication. For example, the text of email, bulletin board postings, or chat room communications.
Mechanisms for maintaining a stateful session with a user or automatically recognizing users who have visited a particular site or accessed particular content previously. For example, HTTP cookies.

- **political**
  Membership in or affiliation with groups such as religious organizations, trade unions, professional associations and political parties.

- **health**
  Information about an individual’s physical or mental health, sexual orientation, use or inquiry into health care services or products, and purchase of health care services or products.

- **preference**
  Data about an individual’s likes and dislikes. For example, favorite color or musical tastes.

- **location**
  Information that can be used to identify an individual’s current physical location and track them as their location changes. For example, Global Positioning System position data.

- **government**
  Identifiers issued by a government for purposes of consistently identifying the individual.

- **other-category**
  Other types of data not captured by the above definitions.

This stanza entry is required.

The default value is uniqueid.

Example:
```
categories = uniqueid
```

```disputes = {yes|no}```

Specifies whether the full P3P policy contains some information regarding disputes over the information contained within the cookie. The value `yes` means that information about disputes is contained in the full P3P policy. The value `no` means that no information about disputes is contained in the policy.

This stanza entry is required.

The default value is `no`.

Example:
```
disputes = no
```
**remedies = {correct|money|law}**

Specifies the types of remedies in case a policy breach occurs. When this entry has no value, there is no remedy information in the P3P compact policy.

- **correct**
  Errors or wrongful actions arising in connection with the privacy policy will be remedied by the service.

- **money**
  If the service provider violates its privacy policy it will pay the individual an amount specified in the human readable privacy policy or the amount of damages.

- **law**
  Remedies for breaches of the policy statement will be determined based on the law referenced in the human readable description.

This stanza entry is required.

The default value is correct.

Example:
remedies = correct

**non-identifiable = {yes|no}**

Specifies that no information in the cookie, or linked to by the cookie, personally identifies the user.

- **yes**
  Data that is collected identifies the user.

- **no**
  No data is collected (including Web logs), or the information collected does not identify the user.

This stanza entry is required.

The default value is no.

Example:
non-identifiable = no

**purpose = {current|admin|develop|tailoring|pseudo-analysis|pseudo-decision|individual-analysis|individual-decision|contact|historical|telemarketing|other-purpose} [:[opt-in|opt-out|always]]**
Specifies the purpose of the information in the cookie and linked to by the cookie.

- **current**
  Information can be used by the service provider to complete the activity for which it was provided

- **admin**
  Information can be used for the technical support of the Web site and its computer system.

- **develop**
  Information can be used to enhance, evaluate, or otherwise review the site, service, product, or market.

- **tailoring**
  Information can be used to tailor or modify content or design of the site where the information is used only for a single visit to the site

- **pseudo-analysis**
  Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data (such as name, address, phone number, or email address) to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals for purpose of research, analysis and reporting, but it will not be used to attempt to identify specific individuals.

- **pseudo-decision**
  Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data (such as name, address, phone number, or email address) to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals to make a decision that directly affects that individual, but it will not be used to attempt to identify specific individuals.

- **individual-analysis**
  Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data for the purpose of research, analysis and reporting.

- **individual-decision**
  Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data to make a decision that directly affects that individual.

- **contact**
  Information can be used to contact the individual, through a communications channel other than voice telephone, for the promotion of a product or service.

- **historical**
  Information can be archived or stored for the purpose of preserving social history as governed by an existing law or policy.

- **telemarketing**
  Information can be used to contact the individual via a voice telephone call for promotion of a product or service.

- **other-purpose**
  Information may be used in other ways not captured by the above definitions.
For all values except current, an additional option can be specified. The possible values are:

- **always**
  Users cannot opt-in or opt-out of this use of their data

- **opt-in**
  Data may be used for this purpose only when the user affirmatively requests this use.

- **opt-out**
  Data may be used for this purpose unless the user requests that it not be used in this way.

When no additional option is specified, the default value is always.

This stanza entry is required.

The default values are current and other-purpose:opt-in.

Example:

```
purpose = current
purpose = other-purpose:opt-in
```

```
recipient = {ours|delivery|same|unrelated|public|other-recipient}
            [:[opt-in|opt-out|always]]
```
Specifies the recipients of the information in the cookie, and linked to by the cookie. Possible values are:

- **ours**
  Ourselves and/or entities acting as our agents, or entities for whom we are acting as an agent. An agent is a third party that processes data only on behalf of the service provider.

- **delivery**
  Legal entities performing delivery services that may use data for purposes other than completion of the stated purpose.

- **same**
  Legal entities following our practices. These are legal entities who use the data on their own behalf under equitable practices.

- **unrelated**
  Unrelated third parties. These are legal entities whose data usage practices are not known by the original service provider.

- **public**
  Public forums. These are public forums such as bulletin boards, public directories, or commercial CD-ROM directories.

- **other-recipient**
  Legal entities following different practices. These are legal entities that are constrained by and accountable to the original service provider, but may use the data in a way not specified in the service provider’s practices.

For all values an additional option can be specified. The possible values are:

- **always**
  Users cannot opt-in or opt-out of this use of their data

- **opt-in**
  Data may be used for this purpose only when the user affirmatively requests this use.

- **opt-out**
  Data may be used for this purpose unless the user requests that it not be used in this way.

When no additional option is specified, the default value is always.

This stanza entry is required.

The default value is ours.

Example:
```
recipient = ours
recipient = public:opt-in
```

```
retention = {no-retention|stated-purpose|legal-requirement|
             business-practices|indefinitely}
```
Specifies how long the information in the cookie or linked to by the cookie is retained. Possible values are:

- **no-retention**
  Information is not retained for more than a brief period of time necessary to make use of it during the course of a single online interaction.

- **stated-purpose**
  Information is retained to meet the stated purpose, and is to be discarded at the earliest time possible.

- **legal-requirement**
  Information is retained to meet a stated purpose, but the retention period is longer because of a legal requirement or liability.

- **business-practices**
  Information is retained under a service provider’s stated business practices.

- **indefinitely**
  Information is retained for an indeterminate period of time.

This stanza entry is required.

The default value is no-retention.

Example:

```
retention = no-retention
```
Appendix B. WebSEAL junction reference

The pdadmin utility provides an interactive command-line prompt from which you can perform WebSEAL junction tasks. This section describes the pdadmin server task command for creating WebSEAL junctions. Refer to the IBM Tivoli Access Manager for e-business Command Reference for complete syntax information for the pdadmin utility.

Note: You can also use the Tivoli Access Manager Web Portal Manager to manage junctions. For more information, see the Web Portal Manager online help screens.

Topic Index:
- “Using pdadmin to create junctions” on page 491
- “The junction commands” on page 492
- “Create a new junction for an initial server” on page 493
- “Add a server to an existing junction” on page 495

Using pdadmin to create junctions

Before using pdadmin, you must login to a secure domain as the sec_master administration user.

For example:

UNIX:

# pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
padmin>

Windows:

MSDOS> pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
padmin>

To create WebSEAL junctions, you use the pdadmin server task create command:

pdadmin> server task server_name-host_name create options

For example, if the configured name of a single WebSEAL server is default, the server_name is default-webseald followed by -host_name. For example, the server name would be:

default-webseald-cruz.dallas.ibm.com

If you install multiple WebSEAL server instances on the same machine, the server_name-host_name is the configured name of the WebSEAL server instance, followed by webseald, followed by the host name:

instance_name-webseald-host-name
For example, if the configured names for two additional WebSEAL instances are `webseal2` and `webseal3`, the server identifications appear as:

```
webseal2-webseald-cruz
webseal3-webseald-cruz
```

**Mandatory junction command options:**

The mandatory command options required to create a basic WebSEAL junction include:

- Host name of the back-end application server (`-h` option)
- Junction type — tcp, ssl, tcpproxy, sslproxy, local (`-t` option)
- Junction point (mount point)

```
pdadmin> server task instance_name-webseald-host-name create -t type
         -h host-name jct-point
```

### The junction commands

The following junction commands are available with `pdadmin server task`:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>Create a new junction for an initial server.</td>
</tr>
<tr>
<td>add</td>
<td>Add additional server(s) to an existing junction point.</td>
</tr>
<tr>
<td>remove</td>
<td>Remove a server from a junction point.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> remove <code>-i server-id junction-point</code></td>
</tr>
<tr>
<td></td>
<td>Use the <code>show</code> command to determine the ID of a particular server.</td>
</tr>
<tr>
<td>delete</td>
<td>Remove the junction point.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> delete junction-point</td>
</tr>
<tr>
<td>list</td>
<td>List all junction points on this server.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> list</td>
</tr>
<tr>
<td>show</td>
<td>Display the details of a junction.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> show junction-point</td>
</tr>
<tr>
<td>jmt load</td>
<td>The <code>jmt load</code> command provides WebSEAL with junction mapping table data</td>
</tr>
<tr>
<td>jmt clear</td>
<td>(jmt.conf) to handle processing of dynamically generated server-relative URLs.</td>
</tr>
<tr>
<td></td>
<td>The <code>jmt clear</code> command removes junction mapping table data from WebSEAL.</td>
</tr>
<tr>
<td>help</td>
<td>List junction commands.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> help</td>
</tr>
<tr>
<td>help command</td>
<td>Display detailed help on a specific junction commands.</td>
</tr>
<tr>
<td>exit</td>
<td>Exits the <code>pdadmin</code> utility.</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> exit</td>
</tr>
</tbody>
</table>

These commands, and associated options, are discussed in the following sections.
Create a new junction for an initial server

Operation: Creates a new junction point and junctions an initial server.

Syntax:
create -t type -h host-name [options] junction-point

<table>
<thead>
<tr>
<th>Junction Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t type</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h host-name</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual Authentication Over SSL</td>
</tr>
<tr>
<td>-K &quot;key-label&quot;</td>
</tr>
<tr>
<td>-B</td>
</tr>
<tr>
<td>-U &quot;username&quot;</td>
</tr>
<tr>
<td>-W &quot;password&quot;</td>
</tr>
<tr>
<td>-D &quot;DN&quot;</td>
</tr>
</tbody>
</table>

| Proxy junction options (requires -t tcpproxy or -t sslproxy) |
| -H host-name | The DNS host name or IP address of the proxy server. |
| -P port | The TCP port of the proxy server. |

| Supplying BA header information |
| -b BA-value | Defines how the WebSEAL server passes HTTP BA authentication information to the back-end server. One of: filter (default), ignore, supply, gso |

<p>| General TCP and SSL junction options |
| -c id-types | Insert Access Manager client identity in HTTP headers across the junction. The id-types argument can include any combination of the following Access Manager HTTP header types: iv-user, iv-user-l, iv-groups, iv-creds, all. |
| -i | WebSEAL server treats URLs as case insensitive. |
| -j | Supply junction identification in a cookie to handle script generated server-relative URLs. |
| -k | Send session cookie to back-end portal server. |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p port</td>
<td>TCP port of the back-end third-party server. Default is 80 for TCP junctions; 443 for SSL junctions.</td>
</tr>
<tr>
<td>-q location</td>
<td>Relative path for <code>query_contents</code> script. By default, Access Manager looks for <code>query_contents</code> in <code>/cgi_bin/</code>. If this directory is different or the <code>query_contents</code> file name is different, use this option to indicate to WebSEAL the new URL to the file. Required for back-end Windows servers.</td>
</tr>
<tr>
<td>-r</td>
<td>Insert incoming IP address in HTTP header across the junction.</td>
</tr>
<tr>
<td>-R</td>
<td>Allow denied requests and failure reason information from authorization rules to proceed across the junction.</td>
</tr>
<tr>
<td>-s</td>
<td>Specifies that the junction should support stateful applications. By default, junctions are not stateful.</td>
</tr>
<tr>
<td>-T resource/ resource-group</td>
<td>Name of GSO resource or resource group. Required for and used only with –b gso option.</td>
</tr>
<tr>
<td>-u UUID</td>
<td>Specifies the UUID of a back-end server connected to WebSEAL via a stateful junction (~s).</td>
</tr>
<tr>
<td>-v virtual-host-name[port]</td>
<td>Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. You use –v when the back-end junction server expects a host name header because you are junctioning to one virtual instance of that server. The default HTTP header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.</td>
</tr>
<tr>
<td>-w</td>
<td>Win32 filesystem support.</td>
</tr>
</tbody>
</table>

**Junction fairness**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l percent-value</td>
<td>Defines the soft limit for consumption of worker threads.</td>
</tr>
<tr>
<td>-L percent-value</td>
<td>Defines the hard limit for consumption of worker threads.</td>
</tr>
</tbody>
</table>

**LTPA junctions**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Enable and disable LTPA junctions.</td>
</tr>
<tr>
<td>-F &quot;keyfile&quot;</td>
<td>Location of key file used to encrypt LTPA cookie data.</td>
</tr>
<tr>
<td>-Z &quot;keyfile-password&quot;</td>
<td>Password for the key file</td>
</tr>
</tbody>
</table>

**WebSEAL-to-WebSEAL SSL junctions**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-C</td>
<td>Mutual authentication between a front-end WebSEAL server and a back-end WebSEAL server over SSL. Requires –t ssl or –t sslproxy type.</td>
</tr>
</tbody>
</table>

**Forms single sign-on**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-S config-file</td>
<td>Location of forms single sign-on configuration file.</td>
</tr>
</tbody>
</table>

**Local junction options (use with –t local)**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d dir</td>
<td>Local directory to junction. <strong>Required.</strong></td>
</tr>
</tbody>
</table>
Add a server to an existing junction

**Operation:** Adds a server to an existing junction point.

**Syntax:**
```
add -h host-name [options] junction-point
```

### Host Name

- **-h host-name**  
  **Required**  
  The DNS host name or IP address of the target back-end server.

### Options

<table>
<thead>
<tr>
<th>Mutual Authentication Over SSL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D &quot;DN&quot;</code></td>
</tr>
</tbody>
</table>

| Proxy junction options (required with `-t tcpproxy` and `-t sslproxy`) |
|-----------------------------|---------------------------------------------------------------------|
| `-H host-name`              | DNS host name or IP address of the proxy server.                    |
| `-P port`                   | The TCP port of the proxy server.                                   |

<table>
<thead>
<tr>
<th>General TCP and SSL junction options</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-i</code></td>
</tr>
<tr>
<td><code>-p port</code></td>
</tr>
<tr>
<td><code>-q url</code></td>
</tr>
<tr>
<td><code>-u UUID</code></td>
</tr>
<tr>
<td><code>-v virt-host-name</code></td>
</tr>
<tr>
<td><code>-w</code></td>
</tr>
</tbody>
</table>

### Junction Point

- Add server to this existing junction point.
Appendix C. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome
Minato-ku
Tokyo 106
Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy,
modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

### Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX
DB2
IBM
IBM logo
SecureWay
Tivoli
Tivoli logo
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.
Index

A
 absolute URL filtering, best practices 339
 accept-client-certs 151, 155, 415
 accept-language HTTP header 53
 access control list (ACL) 4
 access, p3p 481
 account expiry 142
 account management pages 88
 account-expiry-notification 142, 445
 account-locked 88, 443
 acct_locked.html 89
 ACL 4
 ACL policies
 defining 3
 explicit 5
 inherited 5
 valid characters for ACL names 111
 WebSEAL-specific 110
 acnt-mgt stanza 88, 212
 ad-server-config 400
 ADI 357
 ADI retrieval 357
 ADI from the WebSEAL client request 359
 aznapi-configuration stanza 360
 configuring dynamic ADI retrieval 364
 deploying the attribute retrieval service 365
 overview 358
 resource-manager-provided-adi 360
 retrieving ADI from the request header 360
 retrieving ADI from the request POST body 361
 retrieving ADI from the request query string 360
 retrieving ADI from the user credential 362
 supplying failure reason across junction 363
 agent.log 105
 event logging format 101
 example 107
 agents 105, 472
 agents-file 105, 472
 allow-backend-domain-cookies 308, 439
 amwebars stanza 365
 amwebars.conf 368
 amwebbackup.lst 92
 amwebcfg
 response file 23
 amwebcfg syntax 21
 AMWS_hd_prefix 359
 AMWS_pb_prefix 359
 AMWS_qs_prefix 359
 application support, back-end 338
 argument-stanza 329
 attribute retrieval service 364, 365
 attribute retrieval service reference 367
 audit data
 utfs 104
 audit trail files 98
 auditcfg 105, 475
 auditing 98
 auditlog 105, 474
 auth-headers stanza 417
 auth-using-compare 398

authentication
 authenticated access to resources 9
 automatic redirection 212
 basic authentication 145
 CDSSO 245
 certificate 149
 configuration overview 140
 configuring multiple methods 142
 conversion library 141
 default configuration 141
 e-community 256
 event logs 102
 forced redirection 212
 forms 147
 forms single sign-on 326
 goals 8
 HTTP header 156
 IP address 159
 local parameters 140
 login prompt 142
 multiplexing proxy agents (MPA) 183
 overview 8
 overview of process 130
 reauthentication 204, 208
 supported methods 130
 supported session data types 130
 switch user 188
 token 160
 unauthenticated access to resources 9
 authentication levels 119
 authentication methods, summary 130
 authentication strength policy 117
 authentication_level 466
 AUTHENTICATION_LEVEL 176
 authentication-levels stanza 119, 417
 authentication-mechanisms stanza 191
 authorization database replica location 40
 authorization database updates and polling 40
 authorization decision information (ADI) 357
 authorization rules evaluator 358
 authorization service 6
 authtoken-lifetime 251, 428
 automatic redirection, to home page 212
 azn_ent_amwebars library 365
 azn-entitlements-services stanza 365
 azn-server-name 478
 aznapi-configuration stanza 40, 100, 360, 365
 aznapi-entitlement-services stanza 479

B
 ba-auth 145, 414
 back-end application support 338
 backicon 76, 449
 backup 92
 backwards compatibility
 error pages 87
 failover authentication 172
 failover cookies 172
 token authentication 165
 base-crypto-library 33, 411
basic authentication
  configuring 145
basic-auth-realm 145, 414
basicauth-dummy-passwd 316, 457
best practices
  absolute URL filtering 339
  supplying HOST header information (-v) 339
BHA 32
bind-dn
  ldap 394
bind-id
  Active Directory 401
bind-pwd
  Active Directory 402
  ldap 394
BSAFE (RSA) 32

cache
  GSKit (SSL) 132
  WebSEAL credentials 132
cache-enabled 394
cache-group-expire-time 396
cache-group-membership 396
cache-group-size 395
cache-policy-expire-time 396
cache-policy-size 395
cache-refresh-interval 40, 477
cache-use-user-cache 396
cache-user-expire-time 395
cache-user-size 395
categories, p3p 483
CDMF library 245
CDSSO authentication 245
cdsso_key_gen 175, 250, 267
cdsso-argument 251, 428
cdsso-auth 249, 428
cdsso-create 249, 428
cdsso-incoming-attributes stanza 429
cdsso-peers stanza 250, 429
cdsso-token-attributes stanza 429
cert-cache-max-entries 153, 416
cert-cache-timeout 154, 416
cert-failure 88, 153, 445
cert-ldap 419
cert-ssl 152, 419
cert-stepup-http 154, 445
certfailure.html 89
certificate authentication 149
  configuration steps 151
  delayed 154
  delayed mode 149
  optional mode 149
Certificate SSL ID cache 153
  disabling 155
certificate-login 152, 444
certificates
  GSKit 228
  iKeyman 228
    key database file types 228
    managing 228
    prompt_as_needed 151
CGI programming
  supporting 336
  supporting WIN32 environment variables 337
cgi-environment-variables stanza 337, 447
cgi-timeout 30, 447
cgi-types stanza 76, 447
character encoding (URL query string), invalid 55
chunk-responses 386
ciphers
  list of supported 38
client-connect-timeout 30, 385
client-notify-tod 86, 445
client-side certificates
  overview 149
common log format (request.log) 107
compress-mime-types stanza 80, 452
compress-user-agents stanza 81, 452
compression
  HTTP data 80
config-file, policy server 480
configuration file
  for each instance 21
  configuring WebSEAL
    amwebcfg 21
    command line 21
  content-cache stanza 451
  content-caches stanza 77
  content-encodings stanza 454
  content-index-icons stanza 449
  Content-Length header 290
  content-mime-types stanza 453
  cookie name 464
cookie names, preserving across -j junctions 296
cookies
  domain 308
  hostname-junction-cookie parameter 286
  junction (IV_JCT) 292
  junction cookie, preventing naming conflicts 286
  preserve-cookie-names stanza 296
  preserving application cookie names 296
  session 135, 300
cred-attribute-entitlement-services 216
cred-ext-attrs 420
credential refresh
  configuration steps 224
  configuration syntax 222
  for specific user 225
  overview 220
  preserve and refresh 223
credential-refresh-attributes stanza 222, 466
credentials
  extended attributes (tag/value) 215, 344
  inserting user session id in HTTP header 344
  overview 10
  credentials cache 132
  configuring 133
  inactivity timeout 134
  lifetime timeout 134
  maximum entries 133
  overview and structure 10
CRL cache, configuring 231
  gsk-crl-cache-entry-lifetime 232
  gsk-crl-cache-size 232
CRL checking, configuring 231
crl-ldap-server 231, 409, 458
crl-ldap-server-port 231, 409, 458
crl-ldap-user 231, 410, 458
crl-ldap-user-password 231, 410, 458
cross-domain sign-on
  CDSSO 245
e-community 256
cross-site scripting
illegal-url-substrings stanza 56
preventing vulnerability 56
cryptographic hardware configuration
BHAPI 32
IBM 4758 32
IBM 4960 32
nP nForce 300 32
PKCS#11 32
pkcs11-driver-path 36
pkcs11-token-label 36
pkcs11-token-pwd 36
Rainbow CryptoSwift 32
secure key storage 32
SSL acceleration 32
token 34
webseal-cert-keyfile-label 37

data compression
gzip 80
HTTP 80
POP policy 82
user agent type 81
db-file 40, 477
decode-query 55, 388
default-policy-override-support 398
default-webseal ACL policy 110
deftype 453
delete-trash-dir 441
directory index icons 76
directory-index 75, 441
diricon 76, 449
disable-ncipher-bsafe 34, 37, 411
disable-rainbow-bsafe 34, 411
disable-ssl-v2 30, 407, 459
for junctions 281
disable-ssl-v3 30, 408, 459
for junctions 281
disable-tls-v1 30, 408, 459
for junctions 281
disputes, p3p 484
dnforpdl 402
doc-root 20, 74, 441
document caching 77
document-cache-control POP 78
flush caches 78
document root directory 19
change location 74
document types for URL filtering 79
document-cache-control POP extended attribute 78
domain cookies 308
domain, Active Directory 401
domino-server-config 403
double-byte-encoding 390
dynamic ADI retrieval 364
deploying the attribute retrieval service 365
dynamic URLs
dynurl-allow-large-posts 350
dynurl-map 347
example 353
GET and POST methods 350
mapping ACL objects 347
overview 347
placing limitations on POST requests 350
providing access control 347
dynamic URLs (continued)
request-body-max-read 330
resolving 349
summary and technical notes 351
updating, dynurl update 349
dynamic-adi-entitlements-services 365
dynurl update 349
dynurl-allow-large-posts 350, 388
dynurl-map 347, 387
dynurl.conf 347
E
e-community authentication 256
$vouch for$ request and reply 262
$vouch for$ token 263
conditions and requirements 263
configuring 264
domain keys 267
e-community cookie 261
encrypting $vouch for$ token 267
features 257
process flow 258
e-community cookie 261
e-community-domain-keys stanza 267, 433
e-community-name 266, 431
e-community-ss-o-auth 266, 431
e-community-token-auth stanza 267, 433
ecommunity
unauthenticated access 270
utf8 271
eccs-o-allow-unauth 270, 433
eccs-o-incoming-attributes stanza 434
eccs-o-token-attributes stanza 434
enable-failover-cookie-for-domain 179, 424
encoding, invalid character (in URL query string) 55
entrust-client 157
ENV 447
Eracon 32
error messages
HTTP 84
macro support for HTTP 86
serviceability 96
error page
certificate login 153
error-dir 87, 442
error.log 96
event log
authentication 102
event log output
HTTP 101
event logging
HTTP logging 100
logcfg 99
event pool
http 100
http:agent 100
http:clf 100
http:cof 100
http:ref 100
event pool
explicit ACL policy 5
extended attributes
credentials 215
document-cache-control (POP) 78
HTTP-Tag-Value 344
reauth (POP) 205

Index 503
failover cookie
  add authentication level  176
  adding data  168
  extended attributes  177
  extracting data  170
  session lifetime timestamp  168
failover cookies
  configure cookie lifetime  175
  configuring  166
  enable domain-wide cookies  179
  enabling  174
  encrypting/decryting cookie data  175
  session lifetime timestamp  176
  utf8 backwards compatibility  175
utf8 encoding  175
failover-add-attributes stanza  425
failover-auth  424
failover-cdso  174, 422
failover-certificate  174, 422
failover-cookie-lifetime  175, 424
failover-cookies-keyfile  175, 424
failover-http-request  174, 422
failover-kerberos/v5  422
failover-password  167, 174, 421
failover-require-activity-timestamp-validation  180, 425
failover-require-lifetime-timestamp-validation  179, 425
failover-restore-attributes stanza  178, 426
failover-token-card  174, 421
failover-update-cookie  176, 177, 425
failure reason  363
fatal.log  96
filter URLs
  Content-Length header  290
  processing URLs in requests  292
  script filtering for absolute URLs  290
  standard filtering rules  288
filter-content-types stanza  79, 463
filter-events stanza  461
filter-request-headers stanza  463
filter-schemes stanza  79, 462
filter-url stanza  289, 460
filtering
  absolute URLs  289
  best practices for absolute URLs  339
  Content-Length header
    X-Old-Content-Length  290
filter absolute URLs with script filtering  290
filter absolute URLs with script processing  292
filter absolute URLs with server-relative  289
filter absolute URLs with URL filtering rules  79
filter absolute URLs with WebSEAL static documents  288
filter absolute URLs with standard URL filtering rules  288
filter absolute URLs with standard MIME types  288
filter absolute URLs with text/html documents  288
filter absolute URLs with text/vnd.wap.wml  288
FIPS mode  33
fts-mode-processing  34, 411
flush caches  78
flush-time  107, 471
forced redirection, to home page  212
forms authentication  147
single sign-on solution  326
forms-auth  147, 414
forms-sso-login-pages stanza  329
front-end WebSEAL servers
  replicate  58
fsso.conf.template  328
G
GET method  350
global sign-on (GSO)  320
gmt-time  106, 473
gsk-crl-cache-entry-lifetime  232, 409
gsk-crl-cache-size  232, 409
GSKit  228
  CRL cache  231
  file types  228
GSKit (SSL) session cache  132
  configuring  133
GSO  320
  configure GSO cache  322
GSO cache, configure  322
gso-cache-enabled  322, 467
gso-cache-entry-idle-timeout  322, 467
gso-cache-entry-lifetime  467
gso-cache-lifetime  322
gso-cache-size  322, 467
gso-resource  329
gzip  80
H
header  157
help  88, 444
help.html  89
HOST header, best practices for junctions  339
hostname-junction-cookie  286, 464
hostname, Active Directory  400
HTML account management pages
  macro support  89
HTML custom pages  88
http  28, 386
HTTP
  data compression  80
  HTTP common log format  107
  HTTP error messages  84
  macro support  86
HTTP error pages
  creating new  86
  location  87
  time of day  86
http event pool  100
HTTP header
  accept-charset  54
  accept-language  53
  limiting size  299
  PD-USER-SESSION-ID  344
HTTP header authentication  156
HTTP logging
  default  105
  enabling and disabling  106
  using event logging  100
HTTP server identity, suppressing  69
HTTP TRACE, enabling  69
HTTP_IV_CREDS  298, 336, 338
HTTP_IV_GROUPS  298, 336, 338
HTTP_IV_REMOTE_ADDRESS  299
HTTP_IV_USER 298, 336, 338
HTTP_PD_USER_SESSION_ID 219
HTTP_PD-USER-SESSION-ID 345
http-headers-auth 156, 416
http-method-trace-enabled 69, 391
http-method-trace-enabled-remote 69, 391
http-port 28, 386
http-request 157, 419
HTTP-Tag-Value junction attribute 218, 344
http-timeout 456
http-timeout (junctions) 30
HTTP/1.1 responses 308
http.agent event pool 100
http.cf event pool 100
http.cf event pool (NCSA) 100
http.ref event pool 100
httpauthn 157
https 29, 386
https-port 29, 386
https-timeout 456
https-timeout (junctions) 30

IBM 4758 32
IBM 4960 32
ICC 33
iKeyman 230
configuration for cryptographic hardware 34
mutually authenticated SSL junctions 282
overview 230
SSL type junctions 280
WebSEAL test certificate 28
ikmuser.properties 34
ikmuser.sample 34
illegal-url-substrings stanza 56
inactive-timeout 134, 437
inherited ACL policy 5
instance name 16
interactive configuration 21
io-buffer-size 457
IP address authentication 159
ipaddr-auth 159, 417
ipauth 122
is-master-authn-server 269, 431
IV_JCT (junction cookie) 292
iv-creds 298, 338
iv-groups 298, 338
iv-remote-address 299
iv-user 298, 338

J
jnt load 293
jnt-map 293, 456
jnt.conf 293
junction cookie, preventing naming conflicts 286
junction cookies 292
junction fairness 42
junction mapping table 293
junction stanza 42
junction-db 276, 456
junctions
  -b filter 318
  -b gso 318
  -b ignore 318

K
Kerberos authentication 182, 234
kerberos5 242, 420
key database file types 228
key management
  configuring CRL checking 231
  configuring the CRL cache 231
  configuring WebSEAL key database parameters 229
iKeyman utility 230
key database file types 228
managing client-side certificates 228
key management (continued)
  managing server-side certificates 228
  overview of WebSEAL parameters 227
kinit 242
ktpass 238

L
lcp_bin 52
lcp_uri 52
LDAP data in HTTP headers 215
ldap-server-config 393
ldapauthn 145, 147
level, authentication 417
libfailoverauthnbm shared library 174
libhttputna 157
libldapauthn 145, 147
libsslauthn 152
libssauauthn 191
libtokenauthn 119
listen-flags 40, 477
listening port 17
local files
  executable UNIX files 77
log data
  utf8 encoding 108
logaudit 105, 474
logcfg 99, 100, 475
logclientid 474
logflush 105, 474
logging
  default HTTP 105
  HTTP (event logging) 100
logical network interface 17
login 88, 443
  prompt conditions 142
login prompt
  conditions 142
login_success.html 89
login-form-action 329
login-page 329
login-page-stanza 329
login-redirect-page 212, 446
login-success 88, 443
login.html 89, 148, 211
logout 88, 143, 443
logout.html 89
logsize 105, 474
Lotus Domino
  hostname 403
  keyfile 404
  KeyFile_DN 404
  KeyFile_PW 404
  LDAPPor 403
  NAB 405
  password 404
  PDM 405
  server 403
  UseSSL 404
LTPA (WebSphere) 324
  configure junction 324
  configure LTPA cache 325
LTPA cache, configure 325
ltpa-cache stanza 325
ltpa-cache-enabled 325, 469
ltpa-cache-entry-idle-timeout 325, 469
ltpa-cache-entry-lifetime 325, 469
ltpa-cache-size 325, 469

M
macro support
  HTML account management pages 89
  HTTP error messages 86
  USERNAME, for reauthentication forms 211
management objects 3
master-authn-server 269, 431
master-dn 480
master-host, policy server 480
master-http-port 269, 431
master-https-port 269, 432
max-client-read 202, 387
max-entries 133, 437
max-size 106, 471
max-webseal-header-size 299, 458
messages 96
  error.log 96
  fatal.log 96
  notice.log 96
  routing file 96
  warning.log 96
mgt-pages-root 88, 443
MIME types, specifying for URL filtering 79
mkkrb5clnt 240
mode 477
mpa 185, 418
MPA authentication 183
msg_webseald.log 96
multi-domain
  Active Directory 400
  multi-locale support 53
multiple WebSEAL instances
  syntax in pdadmin commands 279, 491
  multiplexing proxy agents (authentication) 183
  mutually authenticated junctions 282

N
name, cookie 464
nCipher nForce 300 32
NCSA combined format 100
net command (Windows) 72
next-token 88, 444
nexttoken.html 89
non-identifiable, p3p 485
notice.log 96

P
p3p
  access 62
  categories 63
  configuration 62
  custom compact policy 67
  default policy 61
  disputes 64
  full policy 67
  header preservation 60
  non-identifiable 64
  policy declaration 60
  policy overview 59
  purpose 65
p3p (continued)  
recipient 66  
remedies 64  
retention 67  
p3p-element 481  
passwd_exp.html 89  
passwd_rep.html 89  
passwd-cdas 419  
passwd-change 88, 443  
passwd-change-failure 88, 444  
passwd-change-success 88, 444  
passwd-expired 88, 443  
passwd-ldap 145, 147, 419  
passwd-strength 164, 420  
passwd-uraf 419  
passwd.html 89  
password strength  
token authentication 164  
password strength policy 114  
PD_PORTAL header 341  
pd-user-name 478  
pd-user-pwd 478  
PD-USER-SESSION-ID HTTP header 344  
pdadmin policy  
disable-time-interval 112  
max-login-failures 112  
max-password-repeated-chars 114  
min-password-alphas 114  
min-password-length 114  
min-password-non-alphas 114  
password-spaces 114  
pdadmin server task  
terminate all_sessions 345  
pdadmin server task (junctions) 279  
pdbackup 92  
pdbackup, archive extract 93  
pdbackup, restore 93  
pdconfig  
configuring WebSEAL 21  
pdweb 72  
pdweb command 72  
pdweb_start 72  
persistent-con-timeout 30, 385  
personalization service  
configuring WebSEAL 341  
example 341  
overview 341  
PIN mode 160  
ping-time 456  
ping-time (junctions) 30  
PKCS#11 32  
pkcs11-driver-path 36, 410  
pkcs11-token-label 37, 410  
pkcs11-token-pwd 37, 410  
pkcs11–driver-path 36  
pkcs11–token-label 36  
pkcs11–token-pwd 36  
pkmscdsso 251  
pkmslogout 143  
pkmspasswd 143, 214  
pkmsvouchfor 262, 269  
policy enforcer 5  
policy-cache-size 477  
polling 40  
polling authorization database 40  
POP 4  
authentication strength (step-up) 117  
POP (continued)  
   defining 3  
   document-cache-control extended attribute 78  
   reauth extended attribute 205  
POP policy  
   quality of protection 126  
   portal-map stanza 341  
   POST method 350  
   configuring limitations 350  
   post password change 144  
   post password change processing 214  
   POST request caching 200  
   post-pwd-change 214  
   post-pwdchg-process 422  
   pre-410-compatible-tokens 180, 252, 271  
   pre-410–compatible-tokens 390  
   pre-510-compatible-tokens 181, 253, 271  
   pre-510–compatible-tokens 391  
   prefer-readwrite-server 396  
   preserve-base-href 69, 391  
   preserve-cookie-names stanza 296, 464  
   preserve-p3p-policy 61, 481  
   problem determination  
   statistics 94  
   trace 94  
   process-root-filter stanza 392  
   process-root-requests 391  
   prompt_as_needed  
   certificate authentication 151  
   protected object 3  
   protected object policies 4  
   protected object space 3  
   management objects 3  
   protected object 3  
   system resource 3  
   user-defined objects 3  
   Web objects 3  
   WebSEAL server-name 74  
   purpose, p3p 485  
Q  
   quality of protection  
   default level 38  
   hosts 39  
   networks 39  
   POP policy 126  
   query_contents 309  
   customizing 311  
   installing 309  
   securing 312  
   query_contents.c 309  
   query_contents.cfg 309  
   query_contents.exe 309  
   query_contents.html 309  
   query_contents.sh 309  
R  
   Rainbow CryptoSwift 32  
   reauth POP extended attribute 205  
   reauth-extend-lifetime 206, 210, 423  
   reauth-for-inactive 209, 210, 423  
   reauth-reset-lifetime 205, 209, 423  
   reauthentication  
   based on security (POP) policy 204
reauthentication (continued)
  based on session inactivity 208
  customizing login forms 211
  preventing session cache entry removal 210
  reauth POP extended attribute 205
  reauth-extend-lifetime 206, 210
  reauth-for-inactive parameter 209
  reauth-reset-lifetime 205, 209, 210
  recipient, p3p 487
  redirect 446
  redirection, to home page 212
  referer.log 105
    event logging format 101
    example 108
  referers 105, 472
  referers-file 105, 472
  registry attribute entitlement service 215
  regular expressions
    for dynamic URLs 348
    for forms single sign-on 330
    list of 330
  related publications xxi
  remedies, p3p 485
  REMOTE_USER 336
  replica authorization database location 40
  replicating front-end WebSEAL servers 58
  request-body-max-read 387
  request caching 200
  request-body-max-read 202, 350
  request-max-cache 203, 387
  request.log 105
    event logging format 101
    example 107
  requests 105, 471
  requests-file 105, 472
  resend-webseal-cookies 136, 438
  resource manager 5
  resource-manager-provided-adi 360
  response file 23
  restore 92
  retention, p3p 488
  rewrite-absolute-with-absolute 290, 464
  routing file 96
  RSA 33
  RSA SecurID
    Linux for zSeries 162
    RSA SecurID client 160

S
  scalability 12
    replicated back-end servers 13
    replicated front-end servers 13
  script filtering (absolute URLs)
    rewrite-absolute-with-absolute parameter 290
    script-filter parameter 290
    script-filtering stanza 290
  script-filter 290, 464
  SecurID 160
  SecurID (RSA) 160
  security policy
    identifying content types 7
    levels of protection 7
    planning and implementing 6
  securitygroup 190
  server identity (HTTP), suppressing 69
  server instance
    adding a new 24
    hostname 16
    logical network interface 17
  server instance name 16
  server instance, configuration 16
  server instance, removing 26
  server-log 471
  server-name 58, 74, 224, 384, 466
  server-root 194, 384
  server-side application support 338
  server-side request caching 200
  service-url 365
  serviceability messages 96
    error.log 96
    fatal.log 96
    notice.log 96
    routing file 96
    warning.log 96
  session cache
    configuring 133
    GSKit (SSL) 132
    inactivity timeout 134
    lifetime timeout 134
    overview and structure 10
    WebSEAL credentials 132
  session cookies 135
    enable 136
  session data types 130
  session ID data types 138
  session ID management 343
  session state
    between client and back-end 343
    configuring GSKit SSL session ID cache 133
    configuring WebSEAL credentials cache 133
    enable session cookies 136
    failover cookies 166
    managing 132
    session cookies 135
    terminate all user sessions 345
    terminate single user session 345
    user session ID management 343
    valid session ID data types 138
  session-activity-timestamp 176, 426
  session-lifetime-timestamp 176, 426
  set-cookie header
    handling the domain attribute 308
    modify Name attribute in -j junctions 296
    modify Path attribute in -j junctions 295
  set-cookie header, as handled by -j junctions 295
  single sign-on
    -b filter 318
    -b gso 318
    -b ignore 318
    -b supply 316
    CDSSO 245
    concepts 316
    configure GSO cache 322
    e-community 256
    forms authentication 326
    global sign-on (GSO) 320
    LTPA (WebSphere) 324
    supply client identity in BA headers 316
    SPNEGO 182
    authentication compatibility 235
    configuration steps 237
SPNEGO (continued)

kinit 242
overview 234
supported platforms 235
spnego stanza 242
spnego-auth 242, 414
spnego-kerberos-service-name 415
SSL configuration
WebSEAL to LDAP 18
SSL connectivity 30
SSL session ID 136
disabling 153
ssl-authn-type 412
ssl-auto-refresh 411
ssl-enabled
ldap 397
ssl-id-sessions 136, 153, 438
ssl-keyfile 230, 406
ldap 397
ssl-keyfile-label 230, 407
ldap 397
ssl-keyfile-pwd 230, 407
ldap 398
ssl-keyfile-stash 230, 407
ssl-listening-port 40, 412
ssl-max-entries 133, 153, 408
ssl-pwd-life 412
ssl-qop-mgmt 38, 435
ssl-qop-mgmt-default stanza 38, 436
ssl-qop-mgmt-hosts stanza 39, 435
ssl-qop-mgmt-networks stanza 39, 435
ssl-v2-timeout 133, 408
ssl-v3-timeout 133
ssl-v3-timeout 408
sslauseth 152
sso-consume 249, 420
sso-create 249, 420
stanzas
uraf-ad in activedir.conf 400
uraf-domino 403
starting WebSEAL 72
stateful junctions 301, 302
statistics utility 94
step-up authentication 117
stepup-login 88, 120, 444
stepuplogin.html 89, 120
stopping WebSEAL 72
su-admin extended attribute 197
su-admins group 188, 190
su-cdesso 421
su-certificate 421
su-excluded group 190
su-http-request 421
su-kerberosv5 421
su-passwd 420
su-token-card 420
suauthn 191
substring 465
suppress-server-identity 69, 390
switch user 188
authentication mechanism 191
built-in shared library 191
custom shared library 197
enabling 190
excluding users 190
securitygroup 190
switch user (continued)
su-admin extended attribute 197
su-admins group 188, 190
su-excluded group 190
using 195
valid authentication methods 194
switch-user 88, 445
switchuser.html 89
system resource 3

T

tag value 215, 344
tagvalue_attributes 217
tagvalue_user_session_id 344
terminate all user sessions 345
terminate single user session 345
three strikes login policy 112
timeout 134, 205, 209, 210, 437
timeout parameters
GSKit SSL session cache 133
HTTP and HTTPS 30
WebSEAL credentials/session cache 133
TLS connectivity 30
token 34
token authentication 160
password strength 162
SecurID 160
token-auth 163, 415
token-cdas 163, 419
token-login 88, 444
tokenauthn 163
tokenlogin.html 89, 211
trace utility 94
Transport Layer Security (TLS) 30
type, MIME 79

U

unauthenticated users, controlling 126
unix-group 384
unix-pid-file 384
unix-user 384
unknownicon 76, 450
update notification listening 40
upgrade
utf-8 issues 47
uraf-ad stanza in activedir.conf 400
uraf-domino 403
URL
about absolute paths 288
about relative paths 288
about server-relative paths 288
filtering options 288
modifying URLs to back-end resources 287
single encoding 47
specifying document MIME types for filtering 79
understanding path types 288
using junction cookies 292
using junction mapping table 293
use-same-session 136, 137, 438
use-utf8 50, 51, 252, 271, 424, 428, 433
useEncryption, Active Directory 401
user identity
match with step-up 125

Index 509
User Registry Adapter Framework (URAF) 400
user session ID management 343
tagvalue_user_session_id 343
user-session-ids 343
user-and-group-in-same-suffix 399
user-defined objects 3
user-diz 441
user-session-ids 224, 343, 439
USERNAME macro, for reauthentication forms 211
utf-8
  authentication impacts 46
  CGI programs 46
  data conversion 45
  upgrade issues 47
utf8
  audit data 104
cdsso 252
  CGI programs 336
  ecsso 271
  HTML macros 90
  junction HTTP headers 305
  junctions 51
  log data 108
  multi-byte login 146
  overview 44
  serviceability messages 97
utf8_bin 52
utf8_uri 52
utf8-form-support-enabled 49, 389
utf8-qstring-support-enabled 50, 389
utf8-template-macros-enabled 441
utf8-url-support-enabled 48, 388
utf8-template-macros-enabled 90

V
verify-step-up-user 125, 418
VeriSign Profiling Service 364
vf-argument 268, 432
vf-token-lifetime 270, 432
vf-url 269, 432
vouch for request and reply 262

W
warning.log 96
Web objects 3
Web Portal Manager 5
  managing junctions 278
WebSEAL
  HTTP/1.1 responses 308
  in object space 74
  overview 1
  root directory of document tree 74
  server-name 74
  starting and stopping server 72
  statistics 94
  trace 94
WebSEAL credentials/session cache
  configuring 133
  overview 132
  overview and structure 10
WebSEAL data
  backing up 92
  restoring 92
WebSEAL junctions, See junctions 275

WebSEAL server
  display status 73
  restarting 73
  starting 72
  stopping 72
WebSEAL server certificate 36
webseal-cert-keyfile 229, 406
webseal-cert-keyfile-label 28, 37, 229, 307, 406
webseal-cert-keyfile-pwd 229, 406
webseal-cert-keyfile-stash 229, 406
webseal-mpa-servers group 184, 185
WebSphere
  deploying the attribute retrieval service 365
WebSphere LTPA 324
WIN32 environment variables, supporting 337
Windows desktop single sign-on 234
  configuration steps 237
worker threads
  global allocation 42
  junction fairness 42
  junctions 42
  managing 41
  per junction allocation 43
WebSEAL 41
worker-thread-hard-limit 42, 457
worker-thread-soft-limit 42, 457
worker-threads 41, 42, 385
WPM
  junction management 278

IBM Tivoli Access Manager for e-business: WebSEAL Administration Guide