IBM Security Access Manager
Version 7.0

# *Plug-in for Web Servers Administration Guide*

SC23-6507-02

IBM Security Access Manager
Version 7.0

# *Plug-in for Web Servers Administration Guide*

SC23-6507-02

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 297.

**Edition notice**

**Note: This edition applies to version 7, release 0, modification 0 of IBM Security Access Manager (product number 5724-C87) and to all subsequent releases and modifications until otherwise indicated in new editions.**

# Contents

# Figures

# Tables

# About this publication

IBM Security Access Manager for Web, formerly called IBM Tivoli Access Manager for e-business, is a user authentication, authorization, and web single sign-on solution for enforcing security policies over a wide range of web and application resources.

IBM Security Access Manager Plug-in for Web Servers manages the security of your Web-based resources by acting as the gateway between your clients and secure Web space. The plug-in implements the security policies that protect your Web object space. The plug-in can provide single sign-on, support, Web servers running as virtual hosts, and incorporate Web application server resources into its security policy.

**Note:** For details on: supported platforms, disk and memory requirements, software prerequisites and installation instructions for the plug-in, refer to the *IBM Security Access Manager for Web: Installation Guide.*

The *IBM Security Access Manager: Plug-in for Web Servers Administration Guide* provides administrative procedures and technical reference information for securing your Web domain using the Plug-in for Web Servers application.

## Intended audience

This guide is for system administrators responsible for the installation, deployment and administration of Access Manager Plug-in for Web Servers.

Readers should be familiar with the following:

- PC and UNIX operating systems.
- Database architecture and concepts.
- Security management.
- Internet protocols, including HTTP, HTTPS and TCP/IP.
- 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.

## Access to publications and terminology

This section provides:

- A list of publications in the “IBM Security Access Manager for Web library” on page xii.
- Links to “Online publications” on page xiii.
- A link to the “IBM Terminology website” on page xiv.

## IBM Security Access Manager for Web library

The following documents are in the IBM Security Access Manager for Web library:

- *IBM Security Access Manager for Web Quick Start Guide*, GI11-9333-01

  Provides steps that summarize major installation and configuration tasks.
- *IBM Security Web Gateway Appliance Quick Start Guide* – Hardware Offering

  Guides users through the process of connecting and completing the initial configuration of the WebSEAL Hardware Appliance, SC22-5434-00
- *IBM Security Web Gateway Appliance Quick Start Guide* – Virtual Offering

  Guides users through the process of connecting and completing the initial configuration of the WebSEAL Virtual Appliance.
- *IBM Security Access Manager for Web Installation Guide*, GC23-6502-02

  Explains how to install and configure Security Access Manager.
- *IBM Security Access Manager for Web Upgrade Guide*, SC23-6503-02

  Provides information for users to upgrade from version 6.0, or 6.1.x to version 7.0.
- *IBM Security Access Manager for Web Administration Guide*, SC23-6504-02

  Describes the concepts and procedures for using Security Access Manager. Provides instructions for performing tasks from the Web Portal Manager interface and by using the **pdadmin** utility.
- *IBM Security Access Manager for Web WebSEAL Administration Guide*, SC23-6505-02

  Provides background material, administrative procedures, and reference information for using WebSEAL to manage the resources of your secure Web domain.
- *IBM Security Access Manager for Web Plug-in for Web Servers Administration Guide*, SC23-6507-02

  Provides procedures and reference information for securing your Web domain by using a Web server plug-in.
- *IBM Security Access Manager for Web Shared Session Management Administration Guide*, SC23-6509-02

  Provides administrative considerations and operational instructions for the session management server.
- *IBM Security Access Manager for Web Shared Session Management Deployment Guide*, SC22-5431-00

  Provides deployment considerations for the session management server.
- *IBM Security Web Gateway Appliance Administration Guide*, SC22-5432-00

  Provides administrative procedures and technical reference information for the WebSEAL Appliance.
- *IBM Security Web Gateway Appliance Configuration Guide for Web Reverse Proxy*, SC22-5433-00

  Provides configuration procedures and technical reference information for the WebSEAL Appliance.
- *IBM Security Web Gateway Appliance Web Reverse Proxy Stanza Reference*, SC27-4442-00

  Provides a complete stanza reference for the IBM® Security Web Gateway Appliance Web Reverse Proxy.
- *IBM Security Access Manager for Web WebSEAL Configuration Stanza Reference*, SC27-4443-00

  Provides a complete stanza reference for WebSEAL.

- *IBM Global Security Kit: CapiCmd Users Guide*, SC22-5459-00

  Provides instructions on creating key databases, public-private key pairs, and certificate requests.
- *IBM Security Access Manager for Web Auditing Guide*, SC23-6511-02

  Provides information about configuring and managing audit events by using the native Security Access Manager approach and the Common Auditing and Reporting Service. You can also find information about installing and configuring the Common Auditing and Reporting Service. Use this service for generating and viewing operational reports.
- *IBM Security Access Manager for Web Command Reference*, SC23-6512-02

  Provides reference information about the commands, utilities, and scripts that are provided with Security Access Manager.
- *IBM Security Access Manager for Web Administration C API Developer Reference*, SC23-6513-02

  Provides reference information about using the C language implementation of the administration API to enable an application to perform Security Access Manager administration tasks.
- *IBM Security Access Manager for Web Administration Java Classes Developer Reference*, SC23-6514-02

  Provides reference information about using the Java™ language implementation of the administration API to enable an application to perform Security Access Manager administration tasks.
- *IBM Security Access Manager for Web Authorization C API Developer Reference*, SC23-6515-02

  Provides reference information about using the C language implementation of the authorization API to enable an application to use Security Access Manager security.
- *IBM Security Access Manager for Web Authorization Java Classes Developer Reference*, SC23-6516-02

  Provides reference information about using the Java language implementation of the authorization API to enable an application to use Security Access Manager security.
- *IBM Security Access Manager for Web Web Security Developer Reference*, SC23-6517-02

  Provides programming and reference information for developing authentication modules.
- *IBM Security Access Manager for Web Error Message Reference*, GI11-8157-02

  Provides explanations and corrective actions for the messages and return code.
- *IBM Security Access Manager for Web Troubleshooting Guide*, GC27-2717-01

  Provides problem determination information.
- *IBM Security Access Manager for Web Performance Tuning Guide*, SC23-6518-02

  Provides performance tuning information for an environment that consists of Security Access Manager with the IBM Tivoli Directory Server as the user registry.

## Online publications

IBM posts product publications when the product is released and when the publications are updated at the following locations:

**IBM Security Access Manager for Web Information Center**
The http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.isam.doc_70/welcome.html site displays the information center welcome page for this product.

**IBM Publications Center**
The http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss site offers customized search functions to help you find all the IBM publications that you need.

### IBM Terminology website

The IBM Terminology website consolidates terminology for product libraries in one location. You can access the Terminology website at http://www.ibm.com/software/globalization/terminology.

## Related publications

This section lists the IBM products that are related to and included with the Security Access Manager solution.

**Note:** The following middleware products are not packaged with IBM Security Web Gateway Appliance.

### IBM Global Security Kit

Security Access Manager provides data encryption by using Global Security Kit (GSKit) version 8.0.x. GSKit is included on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform.

GSKit version 8 includes the command-line tool for key management, GSKCapiCmd (**gsk8capicmd_64**).

GSKit version 8 no longer includes the key management utility, iKeyman (**gskikm.jar**). iKeyman is packaged with IBM Java version 6 or later and is now a pure Java application with no dependency on the native GSKit runtime. Do not move or remove the bundled *java*/jre/lib/gskikm.jar library.

The *IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6 and 7, iKeyman User's Guide for version 8.0* is available on the Security Access Manager Information Center. You can also find this document directly at:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/security/60/iKeyman.8.User.Guide.pdf

**Note:**

GSKit version 8 includes important changes made to the implementation of Transport Layer Security required to remediate security issues.

The GSKit version 8 changes comply with the Internet Engineering Task Force (IETF) Request for Comments (RFC) requirements. However, it is not compatible with earlier versions of GSKit. Any component that communicates with Security Access Manager that uses GSKit must be upgraded to use GSKit version 7.0.4.42, or 8.0.14.26 or later. Otherwise, communication problems might occur.

## IBM Tivoli Directory Server

IBM Tivoli Directory Server version 6.3 FP17 (6.3.0.17-ISS-ITDS-FP0017) is included on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform.

You can find more information about Tivoli Directory Server at:

http://www.ibm.com/software/tivoli/products/directory-server/

## IBM Tivoli Directory Integrator

IBM Tivoli Directory Integrator version 7.1.1 is included on the *IBM Tivoli Directory Integrator Identity Edition V 7.1.1 for Multiplatform* product image or DVD for your particular platform.

You can find more information about IBM Tivoli Directory Integrator at:

http://www.ibm.com/software/tivoli/products/directory-integrator/

## IBM DB2 Universal Database™

IBM DB2 Universal Database Enterprise Server Edition, version 9.7 FP4 is provided on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform. You can install DB2® with the Tivoli Directory Server software, or as a stand-alone product. DB2 is required when you use Tivoli Directory Server or z/OS® LDAP servers as the user registry for Security Access Manager. For z/OS LDAP servers, you must separately purchase DB2.

You can find more information about DB2 at:

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

## IBM WebSphere® products

The installation packages for WebSphere Application Server Network Deployment, version 8.0, and WebSphere eXtreme Scale, version 8.5.0.1, are included with Security Access Manager version 7.0. WebSphere eXtreme Scale is required only when you use the Session Management Server (SMS) component.

WebSphere Application Server enables the support of the following applications:

- Web Portal Manager interface, which administers Security Access Manager.
- Web Administration Tool, which administers Tivoli Directory Server.
- Common Auditing and Reporting Service, which processes and reports on audit events.
- Session Management Server, which manages shared session in a Web security server environment.
- Attribute Retrieval Service.

You can find more information about WebSphere Application Server at:

http://www.ibm.com/software/webservers/appserv/was/library/

## Accessibility

Accessibility features help users with 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 can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

Visit the IBM Accessibility Center for more information about IBM's commitment to accessibility.

## Technical training

For technical training information, see the following IBM Education website at http://www.ibm.com/software/tivoli/education.

## Support information

IBM Support provides assistance with code-related problems and routine, short duration installation or usage questions. You can directly access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html.

The *IBM Security Access Manager for Web Troubleshooting Guide* provides details about:

- What information to collect before you contact IBM Support.
- The various methods for contacting IBM Support.
- How to use IBM Support Assistant.
- Instructions and problem-determination resources to isolate and fix the problem yourself.

**Note:** The **Community and Support** tab on the product information center can provide more support resources.

# Chapter 1. Introducing Security Access Manager Plug-in for Web Servers

Security Access Manager Plug-in for Web Servers implements security policy for your protected Web space. Installed as part of the same process as your Web server, the plug-in acts as the security gateway between your clients and your protected Web space.

This introductory chapter gives an overview of Security Access Manager Plug-in for Web Servers technology, identifies the technical requirements for the product and provides an introduction to the processes for ensuring the security of your Web space using the plug-in.

**Note:** For details on supported platforms, disk and memory requirements, software prerequisites and installation instructions, see the *IBM Security Access Manager for Web: Installation Guide.* For details on upgrading Security Access Manager Plug-in for Web Servers to version 7.0 see the *IBM Security Access Manager for Web: Upgrade Guide*.

This introductory chapter contains the following topics:



## Security Access Manager Plug-in for Web Servers technology

The plug-in can be integrated with IBM Security Access Manager for Web to provide a complete security solution for your Web resources. The plug-in operates as part of the same process as your Web server, intercepting each request that arrives, determining if an authorization decision is required and providing a means for user authentication if necessary. The plug-in can provide single sign-on solutions and incorporate Web application resources into its security policy.

### Basic operational components and architecture

Security Access Manager Plug-in for Web Servers consists of two basic components; the plug-in and the authorization server. The plug-in accepts all requests to the Web server. Requests that have been identified through policy as not requiring authorization are passed directly to the Web server for processing. Other requests are passed to the authorization server.

The authorization server determines which virtual host the request is addressed to (if virtual hosts are present on the Web server) and determines if the request requires authorization. Requests that require authorization are processed by the authorization server.

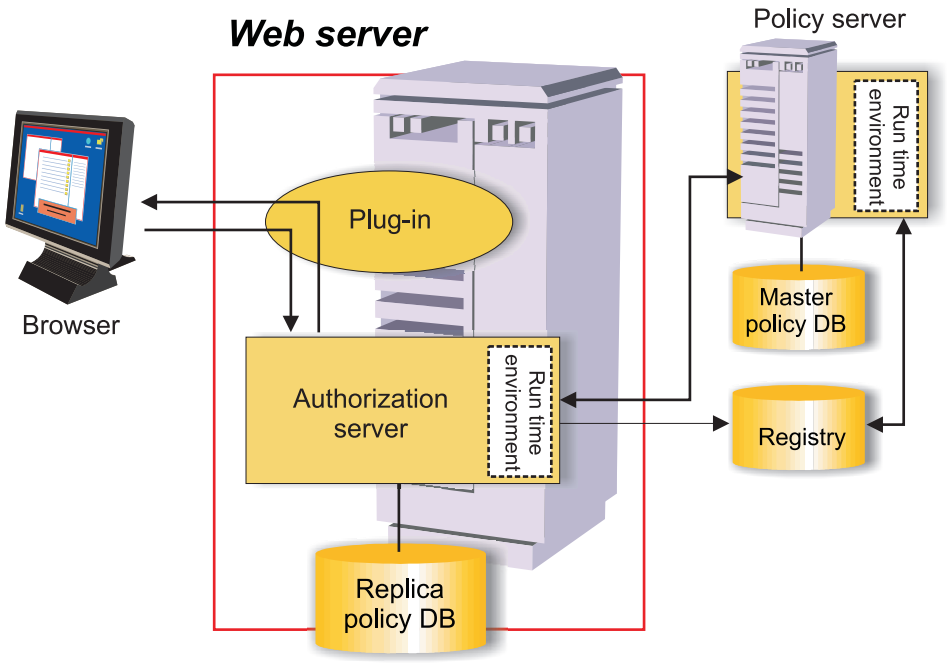


*Figure 1. Plug-in and Security Access Manager component interaction.*

## Plug-in features

Security Access Manager Plug-in for Web Servers provides the following features:

- It supports multiple authentication methods, including basic authentication, IP address (IPv4 and IPv6), token, certificates, and forms, among others.
- Accepts HTTP and HTTPS requests.
- Protects Web server resources by authenticating and authorizing user requests that are dependant on organizational policy.
- Supports the authentication and authorization of requests in a virtual host environment.
- Manages access control to the Web server space. Supported resources include URLs, URL-based regular expressions, CGI programs, HTML files, Java servlets, and Java class files.
- Caches session and credential information to eliminate repetitive queries to the user registry database during authorization checks.
- Provides single sign-on capabilities

### Security policy using the plug-in

A corporate security policy identifies the Web resources requiring protection and the level of protection required for each of those Web resources. Security 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 they specify the operations permitted on the object for each user type.

- 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.

- Authorization rules

  Authorization rules are conditions contained in an authorization policy that are used to make access decisions based on attributes such as user, application, and environment context.

- Extended attributes

  Extended attributes are additional values placed on an object, ACL, or POP that can influence an authorization decision.

To successfully implement security policy, you must logically organize the different content types and apply the appropriate ACL and POP policies. Access control management can be complex and is made easier by careful categorization of the content types. Comprehensive information on Security Access Manager including details of setting policy can be found in the *IBM Security Access Manager for Web: Base Administration Guide*.

For access to resources that do not require authentication, the plug-in supports a special cache called the unprotected resource cache. This cache identifies objects that can accept unauthenticated requests thus bypassing the need to engage the authorization server.

### Support for virtual hosts

A single Web server can appear as more than one host to the Internet by using virtual hosts. The Web servers supported by Security Access Manager Plug-in for Web Servers all provide virtual hosting capability.

Security Access Manager Plug-in for Web Servers provides the capability to implement security policy on a per virtual-host-basis.

## The request handling process

Security Access Manager Plug-in for Web Servers processes each Web request as it arrives at the Web server.

There are eight steps to the request handling process:

1. **Virtual host identification**

   Requests are examined to determine the destination virtual host for the request.

2. **Session identification**

   Once the virtual host has been determined the request is examined for existing authenticated session information. This information may be either a session cookie or SSL session ID. The information that is used for identifying the session is determined by the configured session modules or in replicated plug-in environments the session management server can be used.

3. **Authentication**

   If no existing session is identified, the request is examined for authentication information. This may be information such as a Basic Authentication user name

and password, a submission to a login form, or a client certificate. The information that is used for authenticating the client is determined by the configured authentication modules.

If valid authentication information exists in the request, a new authenticated user session is created. If no authentication information is present, the request is treated as unauthenticated.

If invalid authentication information is present in the request and the authentication method supports re-entry (for example, Basic Authentication) of authentication information then the user is challenged to provide their authentication again. If the authentication method does not support re-entry (for example, client certificate) an error is returned to the client. For more information about plug-in authentication, see "Plug-in authentication" on page 5.

4. **Pre-authorization**

   In some cases, initial request processing may be required before access to a requested resource is authorized. This processing is performed by configured pre-authorization modules. Pre-authorization modules provide functions that do not require authorization or they support capabilities that require access to the request prior to the authorization decision.

5. **Authorization**

   During the authorization, Security Access Manager policy is consulted using the credential information associated with the session to determine whether access should be granted to the requested resource and if so under what conditions.

6. **Authentication upgrade**

   In cases where the authentication level of the user is inappropriate for accessing a requested resource or a request is unauthenticated, the request is examined again for authentication information that would authenticate the user at the required authentication level.

   If no such information exists, and an authentication module that supports the ability to challenge the user for information at the required authentication level is configured, then the user is challenged to provide such information. If there is no means to upgrade the authenticated user session to a sufficient level to access the resource, access is denied.

7. **Post-authorization**

   Occasionally, after the authorization decision has occurred, certain processing may be required to:

   - Modify the Web request before it is handled by the Web server. For example, to insert a header,
   - Modify the Web response generated by the Web server. For example, to set a cookie,
   - Generate a complete response without the Web server handling the request. For example, processing redirects the user to a particular page after a successful logon.

   These operations occur after the result of the authorization process is known since the decision may affect how the request should be handled. These capabilities are provided by post-authorization modules.

8. **Response handling**

   Functions such as Forms Single Sign-on (FSSO) and External Authentication Interface (EAI) processing require the response generated by the Web server to be processed by the plug-in rather than being sent to the client. Generally, with

response handling, the plug-in processes responses from the Web server before passing an alternative response to the user. Modules that require this capability are called response modules.

## Plug-in authentication

Authentication is the method of identifying an individual process or entity attempting to log on to a secure domain. Authorization is the method of determining whether an authenticated user has the right to perform an operation on a specific resource. Authentication ensures that the individual is who they claim to be, but says nothing about the ability to perform operations on a resource.

Security Access Manager Plug-in for Web Servers enforces a high degree of security in a secure domain by requiring each client to provide proof of identity. Comprehensive network security can be provided by having the plug-in control the authentication and authorization of clients.

The following conditions apply to plug-in authentication:

- The plug-in supports a standard set of authentication methods. You can customize the plug-in to support other authentication methods.
- The plug-in process is independent of the authentication method.
- The plug-in requires only a client identity. From this identity, the plug-in obtains an authenticated (or unauthenticated) credential that can be used by the authorization server 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 plug-in authentication process results in the following actions:

1. Client authentication results in a client identity.

   Client authentication is only successful if the user has an account defined in the Security Access Manager user registry. Otherwise the user is designated as unauthenticated.
2. Security Access Manager Plug-in for Web Servers uses the client identity to acquire credentials for that client.

   The plug-in matches the authenticated client identity with a registered Security Access Manager user. The plug-in then obtains the appropriate user credentials. This is known as credential acquisition.

   Credentials include the user name and any groups where the user has membership. These credentials are available to the plug-in that permits or denies access to requested objects in the Security Access Manager protected object space.

   Credentials can be used by any Security Access Manager service that requires information about the client. Credentials allow Security Access Manager to securely perform a multitude of services such as authorization, auditing, and delegation.

   See Chapter 3, “Authentication and request processing,” on page 49 for further information about support for specific authentication methods.

## Credential acquisition

The primary goal of the authentication process is to acquire credential information describing the client user. The user credential is a key requirement for participating in the secure domain.

Security Access Manager distinguishes the authentication of the user from the acquisition of credentials. An identity of a user is always constant. However, credentials that 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.

The authentication process results in method-specific user identity information. This information is checked against user account information that resides in the Security Access Manager user registry (LDAP by default). Security Access Manager Plug-in for Web Servers maps the user name and group information to a common domain-wide representation and format known as the extended privilege attribute certificate (EPAC).

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.

The resulting credential, which represents privileges of a user in the secure domain, describes the user in a specific context and is valid only for the lifetime of that session.

Security Access Manager credentials contain the user identity and groups where this user has membership.

Credentials are used by any Security Access Manager service that requires information about the client. For example, the Security Access Manager authorization server uses credentials to determine whether a user is authorized to perform specific operations on a protected resource in the secure domain. Credentials are also used in other tasks such as logging and auditing.

EPACs contain the Unique Universal Identifiers (UUIDs) that Security Access Manager needs to work with access control lists (ACLs).

The following EPAC fields are appropriate to Security Access Manager:

*Table 1. Security Access Manager EPAC fields*

| Attribute | Description |
|---|---|
| **Secure Domain ID** | Principal's home secure domain identifier |
| **Principal UUID** | UUID of the principal |
| **Group UUIDs** | UUID(s) of groups to which the principal belongs |

Security Access Manager Plug-in for Web Servers can be configured to refresh credential information for a user while keeping their session current. This is useful functionality in cases when a user requires extra access to certain secure applications or when you want to restrict access of a user without having the user log out of their current session. For more information on configuring the plug-in for credential refresh see “Credential refresh” on page 43.

# Chapter 2. Configuration

This chapter describes general administration and configuration tasks you can perform to customize IBM Security Access Manager (Security Access Manager) Plug-in for Web Servers.

Topics in this chapter include:



## General plug-in information

The following sections describe general information about Security Access Manager Plug-in for Web Servers configuration:



### Root directory of the Security Access Manager Plug-in for Web Servers installation

The Security Access Manager Plug-in for Web Server's program files are installed in the following root directory:

**UNIX:**

/opt/pdwebpi/

**Windows:**

C:\Program Files\Tivoli\PDWebPI\

You can configure this path during a Windows installation of the plug-in. You cannot configure this path on UNIX installations. This guide uses the *install_path* variable to represent this root directory.

On UNIX installations, the following separate directory contains expandable files, such as audit and log files:

/var/pdwebpi/

Log files are written to the common Tivoli® Directory if this was selected during the configuration of the Security Access Manager runtime.

## The pdwebpi.conf configuration file

You can customize the operation of the plug-in by configuring the parameters located in the pdwebpi.conf configuration file. The file is located in the following directory:

**UNIX:**

*install_path*/etc/

**Windows:**

*install_path*\etc\

See Appendix C, "Plug-in configuration file reference," on page 211 for a description of the configurable parameters within the pdwebpi.conf configuration file.

**Note:** Anytime you make a change to the pdwebpi.conf file, you must manually restart Security Access Manager Plug-in for Web Servers so that the new changes are recognized. See "Starting and stopping Security Access Manager Plug-in for Web Servers" for information on starting and stopping the application.

## The pdwebpimgr.conf configuration file

UNIX installations of the plug-in include the configuration file pdwebpimgr.conf. This configuration file contains parameters that are used to automatically re-start the authorization daemon if it has failed.

The file is located in the directory:

*install_path*/etc/

There is no reason why you should need to change the parameters in this file.

## Starting and stopping Security Access Manager Plug-in for Web Servers

To start and stop the plug-in process, use the **pdwebpi_start** command on UNIX and use the Services Control Panel on Windows.

**UNIX:**

```
pdwebpi_start {start|stop|restart|status}
```

For example, to stop the plug-in and then restart it, use:

```
# pdwebpi_start restart
```

The **pdwebpi_start** command is located in the following directory:

*install_path*/sbin/

**Windows:**
Identify the plug-in process in the Services Control Panel and use the appropriate control buttons.

**Note: pdwebpi** is the authorization server process. On UNIX installations, the process **pdwebpimgrd** automatically restarts the authorization server if it fails. On Windows, the authorization server is automatically restarted by Windows services.

# HTTP error messages

Sometimes Security Access Manager Plug-in for Web Servers attempts to service a request and fails. There can be many causes for this failure. Two common causes of failure are:

- A file does not exist.
- Permission settings forbid access.

When a failure to service a request occurs, the plug-in returns an error code to the Web server, which interprets the error code and displays a corresponding error page.

## Customizing the display of IIS error messages

IIS provides the ability to customize the format and content of error pages displayed to clients. This is useful for displaying more detailed error information to clients. The plug-in can utilize this error customization facility within IIS.

Using the **use-error-pages** parameter within the **[iis]** stanza of the `pdwebpi.conf` configuration file you can choose whether IIS configured error pages or the standard error code pages are returned to the client browser. Set to *yes*, the **use-error-pages** parameter causes the plug-in to utilize any customized IIS error pages. Set to *no*, standard error pages are displayed for errors encountered by the Authorization Server. By default the **use-error-pages** parameter is set to *no*.

**Note:** Setting **use-error-pages** to *yes*, thus allowing the display of customized IIS error pages for Authorization Server errors, results in a significant degradation in plug-in performance.

# Macro support

The following macros are available for use in customizing HTML error pages. Macros dynamically substitute appropriate information that is available. Macros are available for use in forms or can be passed in URLs, or both. This ability is indicated in the right-hand columns of the table.

*Table 2. Supported Macro Substitutions*

| Macro | Description | Forms | URL |
|---|---|---|---|
| %user name% | The name of the logged-in user. | Y | Y |

*Table 2. Supported Macro Substitutions (continued)*

| Macro | Description | Forms | URL |
|---|---|---|---|
| %ERROR_CODE% | A numeric error code associated with an error. | Y | Y |
| %ERROR_TEXT% | Error text associated with an error. | Y | N |
| %METHOD% | The HTTP method used for the corresponding request. | N | Y |
| %URL% | The URL requested by the client. | Y | Y |
| %HOSTNAME% | Fully qualified hostname. | Y | Y |
| %PROTOCOL% | The protocol, either HTTP or HTTPS, used in the corresponding request. | N | Y |
| %HTTP_BASE% | Base HTTP URL of the server: http://*host*:*tcpport*/ | Y | N |
| %HTTPS_BASE% | Base HTTPS URL of the server: https://*host*:*sslport*/ | Y | N |
| %HTTP_BODY% | The body of the request, if one exists. | Y | N |
| %REFERER% | The value of the referer header from the request, or 'Unknown' if none. | Y | Y |
| %BACK_URL% | The value of the referer header from the request, or '/' if none. | Y | Y |
| %BACK_NAME% | The value 'BACK' if a referer header is present in the request, or 'HOME' if none. | Y | N |
| %POST_URL% | The configured POST URL for any of the Security Access Manager supplied forms. | Y | N |
| %COOKIES% | Any cookies that are found in the request. | Y | N |
| %CUSTOM% | A 'custom' macro that can be used by the different plug-in components to insert custom information. | Y | N |
| %AUTHNLEVEL% | The authentication level at which the credential is currently authenticated. | N | Y |
| %OLDSESSION% | This macro is available for forms based authentication mechanisms and redirect URI based mechanisms; that is, forms, token and external authentication interfaces. It is set to 1 if an old session cookie was detected and an empty string if no old session was detected. This is useful for providing users with an explanation that their session timed out. | Y | Y |

## Forms related macros

Security Access Manager Plug-in for Web Servers provides the following forms, located in the `/opt/pdwebpi/nls/html/lang/charset` directory:

- switchuser
- token
- forms login
- change password

These forms have been configured with the %POST_URL% macro. The %POST_URL% macro allows the plug-in to resend any POST data which might

have been included in the original request. Without the %HTTP_BODY% macro any POST data provided with the original request is lost once the plug-in has finished processing the form.

The default forms have also been configured to cache any necessary session data within the form itself. This session data includes the originally requested URL, the referer for the originally requested URL, and the body of the original request.

## Configuring the Authorization Server

Most authorization and authentication processing is handled by the Authorization Server.

The Authorization Server provides a pool of worker threads that:

- Accepts requests from the plug-in.
- Sends results of each request back to the plug-in.

The plug-in communicates with the Authorization Server through an IPC mechanism that is implemented by using shared memory. The `[proxy-if]` stanza in the `pdwebpi.conf` configuration file specifies configuration parameters that pertain to communication between the plug-in and the Authorization Server.

### Configuring Worker Threads

The **number-of-workers** and the **worker-size** parameters in the **[proxy-if]** stanza of the configuration file, specify values that can be tuned to provide optimal plug–in Authorization Server performance. Consider the quantity and type of traffic on your network when setting these values.

```
[proxy-if]
number-of-workers = 10
worker-size = 10000
cleanup-interval=300
```

The **number-of-workers** parameter specifies the number of concurrent incoming requests that can be serviced by the plug-in. Requests that arrive when all worker threads are busy are buffered until a worker thread becomes available.

This parameter simply specifies the number of threads made available to service a potentially unlimited work queue. The parameter should be increased according to the maximum number of requests you expect the Web server to accept simultaneously. On UNIX platforms the operating system may impose limits on this value.

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.

The **pdwebpi.threads** statistic can be monitored to determine thread activity. This statistic reports the total number of worker threads active at any instant of time.

If this statistic shows that the number of active threads is consistently at the number of configured threads, then better throughput may be obtained by increasing the number of worker threads. No benefit will be obtained if the CPU

utilization of the system is already at 100% since an increase in threads can only increase throughput if there is processing power available for use by the extra threads.

The **worker-size** parameter defines the amount of shared memory that is allocated to each worker thread. This shared memory is used to pass information between the Web server plug-in and the authorization server.

The size of this shared memory must be sufficient to transfer the request and response information, which is needed by the authorization server to perform its configured functions. The size needs to be sufficient to transfer any given HTTP request that the Web server receives to the authorization server, including the HTTP request headers and body.

Similarly, if you enable features that require configuration of a response module, then the shared memory must be large enough to contain any response that needs to be processed, including response headers. Some additional memory is consumed as overhead by the plug-in.

Allow an additional 50% of the expected maximum request size (or response size if response modules are configured) to provide for this overhead. If the **worker-size** setting you are using is not sufficient, your errors will be reported in the Web server plug-in error log or the authorization server log, depending on which component detected insufficient memory.

**Note:** On UNIX and Linux systems, the Security Access Manager Plug-in for Web Servers uses System V shared memory segments and semaphores to communicate between the plug-in and the authorization server. The parameters that may require tuning on Solaris, AIX® and Linux systems are described in the *IBM Security Access Manager for Web: Performance Tuning Guide*.

The **cleanup-interval** is the time in minutes between successive clean-ups of the Authorization Server's shared memory.

**Note:** Change the **cleanup-interval** and **worker-size** parameters only to troubleshoot performance problems.

## Setting the Maximum Session Lifetime for IPC requests

The **max-session-lifetime** parameter in the **[proxy-if]** stanza of the `pdwebpi.conf` configuration file sets the time in seconds that the plug-in will wait for a response from the Authorization Server before timing out. This parameter relates only to the short-lived "session" that is established between the plug-in and the Authorization Server for request handling. If such a timeout occurs, an error page is sent to the client. Such timeouts are highly unlikely.

```
[proxy-if]
max-session-lifetime = 300
```

**Note:** The **max-session-lifetime** parameter does not control the lifetime of authenticated sessions. Authenticated session lifetime is controlled by the **timeout** parameter in the **[sessions]** stanza.

## Configuring for virtual host servers

Virtual hosts are identified to Security Access Manager Plug-in for Web Servers by an arbitrary name that is set in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file.

The plug-in can apply distinct security policy according to two characteristics of a request:

- An ID for the virtual host to which the request was addressed.
- The protocol (http or https) over which the request arrived.

The virtual host ID is derived from the host Web server's configuration information and is Web-server specific. It is determined as follows:

**IHS and Apache** The configuration algorithm used to construct virtual host IDs is as follows:

1. If a **ServerName** directive exists inside a *<VirtualHost {hosta}:{port} {hostb}:{port}...>* block, that name is used for constructing the object space for every host in the virtual host list. No attempt is made to resolve the supplied *servername* into a fully qualified *hostname*.
2. If there is no **ServerName** directive inside the **VirtualHost** block and the host names in the list are not numeric IP addresses, an attempt is made to fully qualify each name and then create object spaces for each distinct host name.
3. If there is no **ServerName** directive inside the **VirtualHost** block and the host names in the list are numeric IP addresses, an attempt is made to resolve each IP address to a fully qualified host name.
4. If there is still no host name and there is a name specified in a global **ServerName** directive, that name is used (without resolving).
5. If there is no global **ServerName** directive, the fully qualified version of the host name of the system is used.

**IIS** The ID corresponds exactly to the Web site name as shown in the Internet Information Services management snap-in. For example, the default Web site created when IIS is configured is named "Default Web Site", this is the ID used by Security Access Manager Plug-in for Web Servers.

Security Access Manager Plug-in for Web Servers defines security policy in terms of virtual hosts. A Security Access Manager Plug-in for Web Servers virtual host is identified by a virtual host ID as defined above and the set of protocols (http, https or both) that it should protect.

The virtual host defines the set and precedence of authentication schemes, session identification schemes, and post-authorization handling that should be applied to requests to the Web server virtual host over one of the matching protocols. The virtual host also defines the mapping of URIs to Security Access Manager Protected Object Space names.

Security Access Manager Plug-in for Web Servers virtual hosts are defined in the **[pdweb-plugins]** stanza of the configuration file. They may be defined as either protected or unprotected.

An unprotected virtual host will have no Security Access Manager security policy applied to it. If a request is received that does not match one of the defined protected or unprotected virtual hosts, a warning message is generated in the

Authorization Server's log file indicating the virtual host ID and the protocol of the request and the request is granted access. This facilitates diagnosis of configuration problems.

Protected virtual hosts are defined by the **virtual-host** parameter of the **[pdweb-plugins]** stanza. Unprotected virtual hosts are defined by the **unprotected-virtual-host** parameter of the **[pdweb-plugins]** stanza. The virtual host name used, typically corresponds to the virtual host ID that this virtual host matches but this is not necessarily always the case. It is the virtual host names defined in the **[pdweb-plugins]** stanza that are used to define virtual host-specific security policy.

The security policy for a particular virtual host is defined by the configuration parameters specified in a stanza with the name of the virtual host. All of the parameters that may be defined in the virtual host stanza have appropriate default values so it is not necessary to have a stanza for each virtual host. It is necessary to have such a stanza only if the security policy for the virtual host differs from the default.

Two parameters of the virtual host are used to match an incoming request to the virtual host that defines the security policy that should be applied to the request. These parameters are **id** and **protocols**.

The **id** parameter is defined to be the virtual host ID that this virtual host will match. The default value for the **id** parameter is the virtual host name itself.

The **protocols** parameter defines the set of protocols that the virtual host will match. This value may be **http**, **https** or **both**. The default value is **both**.

The remaining parameters of the virtual host define the security policy that should be applied to requests matching this virtual host.

Virtual hosts are associated with a particular sub-branch of the protected object space. A request's URI is prefixed with this sub-branch to construct a protected object space name. This protected object space name is used for making authorization decisions. The **branch** configuration parameter defines the name of this protected object space.

```
[virtual_host_name]
branch = virtual_host_id
```

If the virtual host ID value has no leading backslash (/), the entry is prefixed with /PDWebPI/.

The **branch** parameter defaults to the value of the **id** parameter resulting in a default object name prefix of /PDWebPI/*virtual-host-id*.

**Virtual host branches explained**
During plug-in configuration an objectspace is created called /PDWebPI. Within this objectspace, entries are created for each of the virtual hosts protected by the plug-in.

The objectspace under a virtual host object is owned by the plug-in Authorization Server that performs authorization decisions for resources in the virtual host object space. By default, the branch of the objectspace used for a virtual host takes its name from the virtual host ID. If a different branch of the /PDWebPI objectspace is

to be used then the *branch* extension is used to specify this. Branches may be shared between virtual hosts. This may happen when virtual hosts are aliases of each other.

**Note:** When a branch is changed, an object needs to be created with the new name. Any ACLs attached under the old branch stay attached to the now non-existent objects.

Following is an example showing the configuration parameters required for a Web server that has four virtual hosts:

- ibm.com®
- lotus.com-HTTP
- lotus.com-HTTPS
- domino.com

The virtual hosts lotus.com-HTTP and lotus.com-HTTPS are really the same virtual host as they share the same branch; however they are distinguished by the type of access (HTTP or HTTPS). In this case, authentication configuration can be set differently depending on the type of access. domino.com is not protected by the plug-in and ibm.com is another virtual host on the same server.

```
[pdweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com-HTTPS
virtual-host = lotus.com-HTTP
unprotected-virtual-host = domino.com

web-server = iis

[lotus.com-HTTPS]
id = lotus.com
protocols = https
branch = lotus.com

[lotus.com-HTTP]
id = lotus.com
protocols = http
branch = lotus.com

[ibm.com]
id = ibm.com
protocols = http, https
branch = ibm.com
```

Be sure to restart the Web Server each time you make a change to the configuration for virtual hosts in the `pdwebpi.conf` configuration file.

Further configuration on a per virtual host basis is necessary to set the authentication parameters for each individual virtual host. See “Configuring authentication for virtual hosts” on page 51 for details on configuring authentication methods for virtual hosts.

## Web-server-specific configuration

Some of the actions of a plug-in are specific to the Web server and therefore require particular configuration, depending on the Web server type on which the plug-in is operating. Use the **web-server** parameter in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file to define your Web server type. Valid values are **ihs**, **iplanet**, **iis** or **apache**. For example:

```
[pdweb-plugins]
web-server = ihs
```

Web-server-specific configuration items exist in the **[iis]**, **[ihs]**, **[apache]** and **[iplanet]** stanzas of the `pdwebpi.conf` configuration file.

Some Web server configuration parameters can be set on a per branch basis by appending the full virtual host branch to the stanza. For example, **[iplanet:/PDWebPI/lotus.com]**. Parameters related to browsing the Web space can be configured this way.

The following table explains the configurable parameters for specific Web server types.

*Table 3. Web-server-specific configuration parameters*

| Parameter | Description |
|---|---|
| **[ihs]** | |
| **query-contents** | Specifies the query contents program to use for browsing the IBM HTTP Server Web space by the 'pdadmin> object list' command. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[ihs:** *branch***]** for example [ihs:/PDWebPI/lotus.com] |
| **query-log-file** | Location of log file for errors encountered by the query contents program. |
| **doc-root** | Specifies the documentation root that provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[ihs:***branch***]** stanza, for example [ihs:/PDWebPI/lotus.com] |
| **[apache]** | |
| **query-contents** | Specifies the query contents program to use for browsing the Apache Web space by the 'pdadmin> object list' command. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[apache:***branch***]** for example [apache:/PDWebPI/lotus.com] |
| **query-log-file** | Location of log file for errors encountered by the query contents program. |
| **doc-root** | Specifies the documentation root that provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[apache:***branch***]** stanza, for example [apache:/PDWebPI/lotus.com] |
| **[iis]** | |
| **query-contents** | Specifies the query contents program for browsing the IIS Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[iis:***branch***]**, for example, [iis:/PDWebPI/lotus.com] |
| **query-log-file** | Location of log file for errors encountered by the query contents program. |

*Table 3. Web-server-specific configuration parameters (continued)*

| Parameter | Description |
|---|---|
| **log-file** | Defines the log file for error and trace messages from the IIS plug-in, that are kept separate from the Authorization Server's log file in order to ensure consistency of the files. If specified as a relative path, the location is relative to the log sub-directory of the installation directory. If specified as an absolute path, the absolute path is used. |
| **use-error-pages** | In certain situations the authorization server will need to send an error code to the client; for example, in the case of a request for authorization information - 401.The IIS server can be configured to send back a specific body for these error codes.<br><br>This parameter controls whether the IIS configured pages for the error code are sent back to client, or whether some simple constructed pages are sent instead. System performance may be effected if the IIS error pages are chosen. |
| **iis5-always-in-data-stream** | For IIS 5 (Windows 2000), some applications also implemented as IIS filters generate additional Web server requests while performing their own processing.<br><br>In order for these requests to be intercepted, the Security Access Manager Plug-in for Microsoft IIS must be configured to always remain in the data stream.<br><br>For performance reasons, the Security Access Manager Plug-in will normally remove itself from the data stream when it has finished processing a request. To force the plug-in to remain in the data stream and ensure that requests generated by other filters during their processing are intercepted, you must enable this configuration parameter.<br><br>This parameter may NOT be overridden on a per-virtual host basis. Any changes to this parameter require IIS to be restarted before they will take effect.<br><br>By default, the Security Access Manager Plug-in removes itself from the data stream when it has finished processing a request.<br><br>This configuration parameter is specific to IIS 5 (Windows 2000) and is ignored for later versions. |

*Table 3. Web-server-specific configuration parameters (continued)*

| Parameter | Description |
|---|---|
| **authenticate-by-redirect** | This parameter controls whether or not redirects are used to trigger Web server authentication (**web-server-authn** authentication module) or client certificate authentication (**cert** authentication module).<br><br>By default redirects are used. This ensures that all application types that may be hosted by IIS can be protected by these IBM Security Access Manager for Web authentication modules.<br><br>If redirects are not used, the type of application that may not be protected by IBM Security Access Manager for Web are those that are implemented as IIS filters.<br><br>Since these authentication methods require interaction with the IBM Security Access Manager for Web IIS extension, an IIS filter application that handles the request (typically based on URL pattern matching) may completely handle the request prior to control being passed to the extension. To ensure that the extension gains control, the client is redirected to a URL handled only by the extension.<br><br>If this is not necessary for your application, a performance improvement can be gained by disabling `authenticate-by-redirect`. This applies particularly to applications where the initial (or all) requests are POST requests with a significant amount of data in the request body.<br><br>When redirects are used in this authentication process, the POST body data must be cached, which involves transfer of the data to and from the authorization server. Disabling redirects removes the need to cache this data, since user agents will typically not submit the POST data until any HTTP 401 authentication exchanges have completed.<br><br>This parameter may be qualified by the IIS Web site name. For example, the following specifies that for all Web sites except "Web Site 2" redirects will not be used:<br>`[iis]`<br>`authenticate-by-redirect = no`<br>`[iis:Web Site 2]`<br>`authenticate-by-redirect = yes`<br><br>Changes to this parameter require restarting IIS before taking effect.<br><br>IIS must be configured to provide an authentication mechanism when this parameter is disabled (that is, `authenticate-by-redirect = no`). For example, if the Web Plug-in is configured for **cert** authentication, then Directory Security for the IIS Web Site must also be configured for certificate authentication. |

*Table 3. Web-server-specific configuration parameters (continued)*

| Parameter | Description |
|---|---|
| **fallback-to-server-port** | When following redirects to non-standard ports, some browsers do not include the non-standard port in the Host header of the redirected request.<br><br>This causes any subsequent redirects to target the standard port (80 for HTTP, 443 for HTTPS) rather than the non-standard port. This may cause these requests to fail.<br><br>Normally, the information in the Host header should be used to target redirects back to this server. Setting this configuration parameter to *true* overrides this and, if the Host header does not include port information, will fallback to the server's listening port as the default (rather than standard port for the protocol).<br><br>Changes to this parameter will not take affect until the Web server has been restarted.<br><br>This is a global parameter and may not be set on a per-virtual host basis nor on a per-branch basis. |
| **[iplanet]** | |
| **query-contents** | Specifies the query contents program for browsing the Sun Java System Web Server Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[iplanet:***branch***]**, for example, [iplanet:/PDWebPI/lotus.com] |
| **query-log-file** | Location of log file for errors encountered by the query contents program. |
| **doc-root** | Specifies the documentation root that provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[iplanet:***branch***]** stanza, for example, [iplanet:/PDWebPI/lotus.com] |

In the example below, the virtual hosts ibm.com and lotus.com both have a corresponding stanza in the configuration file: `[iplanet:/PDWebPI/ibm.com]` and `[iplanet:/PDWebPI/lotus.com]` where specific configuration parameters are defined.

```
[pdweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com
web-server = iplanet

[iplanet]
query-contents = /opt/pdweb/bin/wpi_iplanet_ls

[iplanet:/PDWebPI/ibm.com]
doc-root = /usr/local/ibm.com/doc/root

[iplanet:/PDWebPI/lotus.com]
doc-root = /usr/local/lotus.com/doc/root
```

## Web server considerations

### IIS

When configuring IIS security settings using the **Directory Security** tab in the Web server *Properties* dialog, it is important to remember that some of the configurable security settings are inheritable through the Web space hierarchy.

The plug-in dynamically creates "virtual" Web space objects to handle various functions. The security settings on these objects are often important. It is essential that the security properties on these objects are not changed.

After IIS security settings have been modified within the **Directory Security** tab of the *Properties* dialog, the *Inheritance Overrides* dialog is displayed. The *Inheritance Overrides* dialog lists *Child Nodes* that override the value you have just set. You have the option to choose which nodes should use the new value. *PDWebPI* nodes must **not** be selected in this dialog.

### Apache and IHS

At configuration time, the Web Plug-in sets configuration directives (**PDWebPIVHostId**) in the Apache `httpd.conf` file. These control the value that the virtual host uses to identify itself to the Web Plug-in. The default values set up by the Web Plug-in can be changed if they do not meet your requirements.

**Disabling Multiviews:**

When using Apache or IHS Web servers the MultiViews directive on the root directory should be disabled. Having the MultiViews directive enabled bypasses the authentication checking by Security Access Manager Plug-in for Web Servers thereby compromising Web server security.

The Multiviews directive is enabled on the document root directory by default in Apache.

**Configuration for PHP scripts:**

Security Access Manager Plug-in for Web Servers only works correctly when PHP scripts are handled internally to the Web server, using PHP support configured as a module implementation.

**Listing objects defined with Apache AliasMatch:**

For Web spaces protected by the plug-in, objects in the Apache `http.conf file` that use the **AliasMatch** directive will not appear in the output of a **pdadmin object list** command.

# Customizing object listings

Security Access Manager Plug-in for Web Servers supplies a binary file for each supported Web server used to determine the output for a **pdadmin** administration object list or an **object show** command.

The following table indicates the standard binary names, and their locations:

- IHS — *install_path*`/bin/wpi_ihs_ls`
- IIS — *install_path*`/bin/wpi_iis_ls.exe`

**Note:** These programs are executed by the Web Plug-in (not directly run by users) whenever a **pdadmin object list** or **object show** command is requested.
If you require object browsing capabilities that are not part of the standard functionality, you will need to develop your own customized binary to replace that supplied with the plug-in.

When developing a customized binary the following guidelines apply:

## Command Line Arguments

**IHS,Apache**

*directory virtual_host log_file* [-d]

Where:

| | |
|---|---|
| *directory* | The absolute path to the directory or file to list or show. |
| *virtual_host* | The virtual host for the directory or file. |
| *log_file* | The absolute path to a file which will contain any error information produced by the action. |
| **-d** | When the **-d** option is specified an object show is executed instead of an object list. |

**IIS**

[-log *log_file*] -path *directory* -vhost *virtual_host* [-d]

Where:

| | |
|---|---|
| *log_file* | The absolute path to a file which will contain any error information produced by the action. |
| *directory* | The absolute path to the directory or file to list or show. |
| *virtual_host* | The virtual host for the directory or file. |
| **-d** | When the **-d** option is specified an object show is executed instead of an object list. |

## Output

For each entry listed the format of the output will be:

```
<Object Type=[type] Description=[description] Attachable=[yes/no]> [name] </Object>
```

Where:

| | |
|---|---|
| type | A number which indicates the object type. Values include:<br>• 0 Unknown<br>• 1 Domain<br>• 2 File<br>• 3 Program<br>• 4 Directory<br>• 5 Junction<br>• 9 HTTP Server<br>• 10 Non-existent object<br>• 11 Container<br>• 12 Leaf<br>• 14 Application Container<br>• 15 Application Leaf |
| *description* | A textual description of the object. |
| attachable | Whether policy can be attached to the object. |
| *name* | The object name for the object. This object name should not contain any leading directory names. |

For example:

```
<Object Type=2 Description="File" Attachable="yes"> apache.gif </Object>
```

## Customizing message and error pages

The plug-in provides a number of default message pages that are returned when particular conditions occur. The content of these message pages can be changed to better suit your needs. You can also create your own error message pages, based on error codes returned by the plug-in.

Message pages are configured in the **[error-pages]** stanza of the `pdwebpi.conf` configuration file using the following format:

```
error code = error page returned
```

The default configuration is listed in the following table.

*Table 4. Default message pages returned for error codes*

| Error Code | Displayed page | Description |
|---|---|---|
| 0x35f02188 | `acct_locked.html` | Displayed when the user account is locked. |
| 0x35f0205d | `login_success.html` | Displayed when a user has successfully logged in, but no referring URL has been located. |
| 0x35f021be | `retry_limit_reached.html` | Displayed when a user has exceeded the allowable login attempts and their account is temporarily disabled. |
| 0x35f02421 | `session_limit_reached.html` | Displayed when a user has reached the concurrent session limit. |
| default | `azn_srv_error.html` | Displayed when an error condition has been detected that does not match any other configured error response. |

## Modifying existing error pages

By default, the HTML error pages are located in the following directory: *install_path*/nls/html/*lang*/*charset*.

Where:

- *lang* is taken from the NLS configuration. In a U.S. English installation, *lang* will be set to **C**.
- *charset* is the character set in which the page is encoded. The default is **utf-8**.

For details on plug-in language support refer to "Language support and character sets" on page 45.

Because error **0x35f021be** (retry_limit_reached.html) is a message returned before user authentication has occurred, any new URL of the message page needs to be either outside the Web server object space protected by the plug-in or have policy attached that permits unauthenticated access. If the error is configured as a macro file, this limitation does not apply.

When customizing the default message pages:

- Do not modify the hexadecimal error number. This is used by the plug-in to identify the error condition.
- The messages are displayed as HTML pages and so valid HTML tagging is required.
- Macros can be used to display dynamic information. Refer to "Macro support" on page 9 for a list of available macros.

## Creating new error pages

You can create new error message pages for errors returned by the plug-in.

### About this task

All plug-in errors are documented in the *IBM Security Access Manager for Web: Error Message Reference*.

### Procedure

1. Locate the error in the *IBM Security Access Manager for Web: Error Message Reference* making note of the error's hexadecimal code.
2. Create a new entry in the **[error-pages]** stanza of the pdwebpi.conf configuration file. The entry must specify the correct hexadecimal error code and assign it the HTML page you have created for display when the error is encountered. For example, to display a customized error page named example.html for an error which has an error code of 0x35f02040:

   ```
   [error-pages]
   0x35f02040 = example.html
   ```

   When only the filename of the message is given in the configuration, the plug-in attempts to locate the HTML message file on the directory, *install_path*/nls/html/*langcharset*.

   Error pages can be stored elsewhere if you specify in the configuration entry the absolute or server relative URL to the message file. For example:

   ```
   [error-pages]
   0x35f02040 = http://www.organization.com/TAM/errors/
          HTTP_method_unrecognized.html
   ```

For errors that occur before the client is authenticated, you will need to be sure the configured HTML message page is not part of the Web server object space protected by the plug-in or ensure the message file location has attached policy that permits unauthenticated access.

3. You can use macros to display dynamic information within the message. Refer to "Macro support" on page 9 for a list of available macros.

## Configuring switch user (SU) for administrators

The Plug-in for Web Servers switch user functionality allows specific administrators to assume the identity of a user who is a member of the Security Access Manager secure domain. The switch user implementation is similar to the **su** command in UNIX environments. In the plug-in environment, the administrator acquires the true credentials of the user and interacts with resources and back-end applications with the exact same abilities as the actual user.

Switch user is a useful help desk tool for troubleshooting and diagnosing problems. Switch user can also be used to test access of a user to resources and perform application integration testing.

The following items highlight the important features of switch user:

- Switch user does not require the password of the user.
- The administrator uses a credential representing the actual user.
- Switch user is restricted to members of a special administrator's group. An administrator cannot switch user to any other member of this group.
- Security Access Manager processes **sec_master**, and other selected users can be excluded from the switch user capabilities through membership in an exclusion group.
- A special HTML form is used to supply switch user information and activate a special authentication mechanism that returns the specified credential of a user without the requirement of a password.
- The administrator uses the **pkmslogout** utility to end a switch user session.

**Note:** When using Internet Explorer, cookie prompting must be switched off (the default browser setting). Otherwise, the cookie cache within the browser may become corrupted, causing the logout process to hang. To disable cookie prompting from within Internet Explorer, select **Internet Options** from the **Tools** menu, then click on the **Advanced** button under the **Privacy** tab and ensure the check box is not selected.

### Understanding the switch user process flow

The following sequence describes the switch user process flow:

1. The process begins with an authenticated administrator who is a member of the **su-admins** group.
2. The administrator connects to a pre-configured switch user HTML form. This form is accessible only to members of the **su-admins** group. If the user is not a member of the **su-admins** group, a "Not Found" page is returned.
3. The switch user form is completed and returned with the following information: user name (administrator is "switched to" this user), a destination URL, and an authentication method. This action results in a POST request being sent to **/pkmssu.form**.

4. Two checks are done before the switch is authorized.
   a. The plug-in checks if the "switched to" user is a member of the **su-admins** group. A user cannot "become" another user who is a member of the **su-admins** group.
   b. The plug-in checks if the "switched to" user is a member of the **su-excluded** group. No users are allowed to "become" a member of the **su-excluded** group. An error is returned if either of these two checks fail. All subsequent requests are made as though they were made by the "switched to" user.
5. The administrator remains as the "switched to" user until the standard Security Access Manager **/pkmslogout** utility is called at which time the "switched to" user is logged out and the administrator returns to their original session.

## Enabling switch user

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable switch user functionality, the **switch-user** module needs to be configured as a pre-authorization module. This allows the switch user function to access a user before authorization is performed.

```
[common-modules]
...
pre-authzn = switch-user
...
```

Ensure that the entry for switch user exists in the **[modules]** stanza of the `pdwebpi.conf` configuration file. For example:

```
[modules]
...
switch-user = pdwpi-su-module
...
```

## Configuring the switch user HTML form

The switch user form is defined in the **[switch-user]** stanza of the `pdwebpi.conf` configuration file.

- The **switch-user-form** parameter specifies the name of the file. By default the file name is `switchuser.html` and is located in the directory *`install_path`*`/nls/html/`*`lang`*`/`*`charset`*. On US English systems, the *lang* directory is called **C** and *charset* is **utf-8**.

  ```
  [switch-user]
  switch-user-form = switchuser.html
  ```

- The **switch-user-uri** parameter contains the URI which is used to invoke the switch user function. Note that the standard authorization policy is not applied to this URI. Group-based authorization checking is conducted instead of ACL checking.

  ```
  [switch-user]
  switch-user-uri = /switchuser.html
  ```

- The **switch-user-post-uri** parameter specifies the URI which the switch user form is submitted to:

  ```
  [switch-user]
  switch-user-post-uri = /pkmssu.form
  ```

The switch user form can be edited for customized appearance and functionality. The form contains requests for:

- **User name** (the administrator "switches to" this user)

This user cannot be a member of **su-excluded**, **securitygroup** and **su-admins**.

- **Destination URL**

  This page is displayed after a successful switch user operation. You can configure this as hidden input containing an appropriate home page or a successful switch user confirmation page.

- **Authentication method**

  The authentication method determines the type of information used to build the user credential. You can configure this field as hidden input. Refer to the notes below for a list of the valid authentication method parameters.

- The macro, %CUSTOM%, is included in the default form and can be used to automatically include in the form all configured switch-user authentication mechanisms.

Switch user form notes:

- The form is only available to members of the **su-admins** group. An ACL is not required on this file. The plug-in performs an internally hard-coded group membership check. The plug-in returns a 404 "Not Found" error when the group membership check fails.
- User name, destination URL, and authentication method are all required data.
- The required data can be built into the form as hidden fields.
- The plug-in 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.
- Valid values for the authentication method include:

  ```
  su-password
  su-token-card
  su-certificate
  su-http-request
  su-cdsso
  ```

  These authentication method parameters specify which authentication mechanism the plug-in is to use.
- Switch user form data is submitted to the **/pkmssu.form** action URL.

## Enabling and excluding users from switch user

Only administrators who are members of the **su-admins** group can use the switch user function and receive the switch user HTML form. Switch user functionality is enabled for any user who is a member of the **su-admins** group.

Administrators can switch user to any Security Access Manager account, except those belonging to certain groups. You can exclude other Security Access Manager users from switch user through membership in the **su-excluded** group. In addition, members of the Security Access Manager **securitygroup** group are excluded from switch user functionality. Typically, **sec_master** and the Security Access Manager processes are members of **securitygroup**.

During switch user, the plug-in performs checks on all three groups. You cannot "switch to" someone who is a member of the **su-admins**, **su-excluded**,or **securitygroup** groups.

## Configuring the switch user authentication mechanism

The switch user authentication mechanism, which is a built-in shared library, creates a credential which represents the **switched-to** user, which is based on the supplied user name and authentication method, without requiring a password as input.

Specify switch user authentication mechanisms in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file. The following authentication mechanisms are supported:

```
[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
```

Security Access Manager supplies a single switch user library that can be used to enable any of the authentication mechanisms in a default, non-customized environment. The switch user library differs from the standard authentication libraries. The library specifies an authentication mechanism that takes the user identity which is supplied in the switch user form. Then, it returns a valid credential for that user without requiring the user password for input.

The built-in switch user shared library that is provided with Security Access Manager is called:

**UNIX** `libsuauthn`

**Windows**
`suauthn`

The switch user feature also supports custom external authentication mechanisms. This support is important because a custom mechanism often supplies more information to the user credential.

Write a custom switch user mechanism that emulates the behavior of your existing mechanism and supports the requirement of returning a credential without requiring the user password for input.

Each configured switch user authentication library must be uniquely named, even when the default library, **libldapauthn**, is used for more than one authentication method.

In the following example for a Solaris platform, an existing environment has three authentication methods enabled:

1. Forms authentication by using the built-in **libldapauthn** library,
2. Certificates authentication by using the built-in **libsslauthn** library,
3. Token authentication by using a custom authentication mechanism that is developed against the external authorization C API.

The environment is now expanded to support switch user feature for any of those three authentication methods. Three more authentication parameters for switch user must be enabled in the `pdwebpi.conf` configuration file. In addition, a new custom library must be written to emulate the existing token authentication mechanism and support the requirements of switch user authentication:

```
[authentication-mechanisms ]
passwd-ldap =/opt/PolicyDirector/lib/libldapauthn.so
cert-ssl =/opt/pdwebrte/lib/libsslauthn.so
token-cdas =/opt/developer_libraries/lib/libcustom.so
su-password =/opt/pdwebrte/lib/libsuauthn.so
su-certificate =/opt/developer_libraries/lib/libsucert.so
su-token-card =/opt/developer_libraries/lib/libsucustom.so
```

where developer_libraries is the directory which contains the custom libraries that are created by the developer.

# Impacting other plug-in functionality

## Impact on session cache timeout configuration

The functionality of the configured plug-in session cache inactivity and lifetime timeout values are not affected by the switch user operation. The inactivity and lifetime timers are associated with the administrator's session cache entry and not the session data that changes during a switch user operation.

The session 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 session lifetime value is not extended by a switch user operation. It is possible for the lifetime timeout of the administrator session to expire during a switch user operation. If this timeout occurs, the sessions are deleted, logging off both the administrator and the 'switched-to' user. The administrator must reauthenticate and begin the switch user operation again.

## Incorporating Step-up authentication levels

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/pdwebrte/lib/libsuauthn.so&-l 1
su-certificate =/opt/developer_libraries/lib/libsucert.so&-l 0
su-token-card =/opt/developer_libraries/lib/libsucustom.so&-l 2
```

where *developer_libraries* is the directory containing the custom libraries created by the developer.

**Note:** For this version of Security Access Manager, the administrator must know the password of the user to successfully perform step-up authentication.

## Support for reauthentication

The plug-in 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:** For this version of Security Access Manager, the administrator must know the "switched-to" user's password to successfully reauthenticate.

### Support for 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 session* task and the *terminate all user sessions* tasks operate as in the following:

- The "switched-to" user session is terminated when the "switched-to" session ID or user session ID is specified.
- Both the administrator session and the "switched to" user session are terminated when the administrator session ID or the User Session ID are specified.

### Support for tag-value

The tag-value capability is recognized and supported by the switch user functionality.

### Auditing the administrator during switch user

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 ***tag_value_prefix*_su-admin**:

*tag_value_prefix*_su-admin = *su-admin-name*

Where *tag_value_prefix_* represents the **tag_value_prefix** parameter value configured in the **[pdwebpi-plugins]** stanza of the plug-in configuration file. This extended attribute is available to any auditing mechanism.

## Configuring failover for LDAP servers

Security Access Manager plug-in for web servers connects to any available LDAP server (master or replica, depending on priority) when it starts. If the LDAP master server is down for any reason, the plug-in needs to connect to an available LDAP replica server for any read operations. This configuration is the standard Security Access Manager LDAP replica configuration. For further details, see the *IBM Security Access Manager for Web: Base Administration Guide*.

IBM Directory (LDAP) supports the existence of one or more read-only replica LDAP servers. Sun Java System (formerly Sun ONE) Directory Server (LDAP) supports the existence of one or more read-only replica LDAP servers known as "consumers". You must add lines to the **[ldap]** stanza of the `pdwebpi.conf` configuration file to identify any replica servers available to the plug-in.

Security Access Manager supports a maximum of one host and nine LDAP replica servers listed in the `ldap.conf` file. If more than nine LDAP replica entries are listed, the Security Access Manager servers cannot start. Do not specify more than nine replica LDAP servers.

Use the following syntax for each replica:

```
replica =ldap_server,port,type,preference
```

where:

*ldap_server*
: The network name of the LDAP replica server.

*port* The port this server listens on. Generally, use `389` for unencrypted communications or `636` for communicating over SSL.

*type* The type of the replica server - either `read-only` or `read-write`. Normally, use `read-only`. A `read-write` type represents a master server.

*preference*
A number from 1 to 10 (10 is the highest preference). The server with the highest preference value is chosen for LDAP connections. See "Setting preference values for replica LDAP servers" in the *IBM Security Access Manager for Web: Base Administration Guide*.

Example:

```
replica =replica1.ldap.ibm.com,389,readonly,5
replica =replica2.ldap.ibm.com,389,readonly,5
```

## Supporting Platform for Privacy Preferences (P3P) headers

The Platform for Privacy Preferences Project (P3P), is a World Wide Web Consortium standard that provides a way to describe user privacy preferences and Web server privacy policy information in a uniform way. Using P3P, a user can configure privacy preferences that determine what information is divulged to a Web server and how that information is used.

With P3P, a Web server can specify what user privacy policy information it gathers and what it will use this information for. The privacy policy of a Web server is made available in a machine-readable format to clients that P3P enabled browsers can "read" and compare with the privacy preferences of the user. The user is alerted when the privacy policy of the Web server and the privacy configuration of the user do not match.

A common use for P3P is to enable browsers to make intelligent decisions about whether to accept cookies received from a Web server. Support for this is enabled by default within Internet Explorer 6.0. If Internet Explorer 6.0 receives a cookie from a site that doesn't send a P3P policy, or sends a policy that does not match the privacy preferences of the user, then the browser may decide to automatically block the cookie.

The plug-in relies on cookies to maintain session information as well as, for instance, to retain failover information. Internet Explorer, using its default settings blocks cookies and so will not store plug-in cookies, effectively limiting plug-in functionality. The plug-in provides P3P configuration options that specify a compact P3P policy statement sent with pages when a plug-in cookie is set.

The plug-in P3P configuration options allow you to create a compact P3P policy that matches the privacy policy of your organization. It is then up to the client to decide if they will allow Security Access Manager cookies to be set.

**Note:** You should not configure a P3P policy that does not match the privacy policy of your organization just to allow Internet Explorer to accept cookies. Before enabling P3P policies for plug-in cookies, ensure you are familiar with the P3P specification and understand exactly what it is you are claiming as the privacy policy of your organization.

## Configuring P3P headers

The plug-in provides configuration parameters that match the definition of the compact policy syntax of the W3C P3P requirements. These must be configured to allow plug-in cookies while still maintaining the integrity of your organization's privacy policy.

The first step in configuring P3P headers is to set the **send-p3p-header** parameter in the **[pdweb-plugins]** stanza in the `pdwebpi.conf` configuration file. This entry can be set on a per virtual host basis by defining it within a user defined **[***virtual_host_name***]** stanza. Set to *true* the **send-p3p-header** parameter specifies whether the plug-in adds a P3P header containing a compact policy statement to any HTTP response in which it has set cookies. The sending of a P3P policy is disabled by default.

If you have enabled the sending of P3P headers, the parameters in the **[p3p-header]** or **[p3p-header:***virtual_host***]** stanza must then be set. These parameters define the compact policy that applies to all HTTP cookies set.

The default settings in this stanza allow session cookies to be stored in an Internet Explorer 6 browser — even if they appear as third-party cookies.

*Table 5. [p3p-header] parameters*

| Parameter | Use |
|---|---|
| **p3p-element** | This parameter can be used to specify a reference to a full XML policy in addition to the compact policy configured using the other parameters in this stanza.<br><br>Uncommenting the line, `p3p-element = policyref="/w3c/p3p.xml"`, directs the browser to send the specified reference to the full XML policy.<br>**Note:** An ACL that allows unauthenticated access needs to be set on the `/w3c` directory to allow access to the policy. This is required as Internet Explorer does not send authentication information with the request that permits the viewing of policy. |
| **access** | Specifies the access the user has to the information contained within and linked to by the cookie. Possible values:<br>*none*<br>*all*<br>*nonident*<br>*contact-and-other*<br>*ident-contact*<br>*other-ident* |
| **disputes** | Specifies whether the full P3P policy contains some information regarding disputes over the information contained within the cookie. Valid values are *true* or *false*. This parameter is set to *false* by default. |
| **remedies** | Specifies the possible remedies for disputes. Possible values are:<br>*correct*<br>*money*<br>*law*<br><br>If not specified, no remedy information is included in the policy. |
| **non-identifiable** | When set to *true*, this parameter specifies that no information in the cookie, or information linked to by the cookie, personally identifies the user in any way. Valid values are *true* or *false*. This parameter is set to *false* by default. |

*Table 5. [p3p-header] parameters (continued)*

| Parameter | Use |
|---|---|
| **purpose** | Specifies the purpose of the information in the cookie and linked to by the cookie. Possible values are:<br>*current*<br>*admin*<br>*develop*<br>*tailoring*<br>*pseudo-analysis*<br>*pseudo-decision*<br>*individual-analysis*<br>*individual-decision*<br>*contact*<br>*historical*<br>*telemarketing*<br>*and other-purpose*.<br><br>For all values except *current*, an additional specifier may be configured. The possible values are:<br>*always*<br>*opt-in*<br>*opt-out*.<br><br>For purposes not specified, the default value is *always*. This value is specified after the purpose, separated by a colon, for example:<br>`purpose = contact:opt-in` |
| **recipient** | Specifies the recipients of the information in the cookie, and the information linked to by the cookie. Possible values are:<br>*ours*<br>*delivery*<br>*same*<br>*unrelated*<br>*public*<br>*other-recipient*. |
| **retention** | Specifies how long the information in the cookie or the information linked to by the cookie is to be retained.<br><br>Possible values are:<br>*no-retention*<br>*stated-purpose*<br>*legal-requirement*<br>*business-practices*<br>*indefinitely*. |

*Table 5. [p3p-header] parameters (continued)*

| Parameter | Use |
|---|---|
| **categories** | Specifies the type of information stored in the cookie or linked to by the cookie.<br><br>If the non-identifiable parameter is set to *true*, then no categories need be configured. Possible values are:<br>*physical*<br>*online*<br>*uniqueid*<br>*purchase*<br>*financial*<br>*computer*<br>*navigation*<br>*interactive*<br>*demographic*<br>*content*<br>*state*<br>*political*<br>*health*<br>*preference*<br>*location*<br>*government*<br>*other-category* |

Example P3P configuration:

```
[pdweb-plugins] or [virtual_host_name]
send-p3p-header = true
...
[p3p-header] or [p3p-header:virtual_host_name]
# p3p-element = policyref="/w3c/p3p.xml"
access = none
disputes = false
non-identifiable = false
purpose = current
purpose = other-purpose:opt-in
recipient = ours
retention = no-retention
categories = uniqueid
```

## Cross-site scripting protection

Cross-site scripting is a server-side vulnerability that is introduced by rendering client user input as HTML. Cross-site scripting attacks can expose sensitive information about the users of a website. To help mitigate the risk of cross-site scripting, a new feature was introduced in Microsoft Internet Explorer 6. This feature adds an attribute to cookies that prevents them from being accessed through client-side script. A cookie with this property set is called a **HttpOnly** cookie.

The optional **HttpOnly** flag reduces the risk of internal cookie information exposed to client-side script. When enabled, this attribute is generally applied to internal security cookies which includes the session, dsess, and failover cookies.

The **HttpOnly** flag can be added to Security Access Manager Plug-in for Web Servers internal cookies by setting a configuration file entry. If set, the **HttpOnly** flag is only recognized by Internet Explorer version 6 and above. Other Web browsers do not recognize this flag.

Setting the **HttpOnly** property does not prevent an attacker with access to the network channel from observing the cookie. You must use the Secure Sockets Layer (SSL) to prevent the cookie from being read in clear text through packet sniffing.

# Configuring plug-in auditing, logging, and tracing

Logging and auditing can provide information to help you identify any problems you might have with the plug-in. If you find you are having trouble and need a real-time view of error messages, then start the plug-in in the foreground using the **-foreground** option:

```
pdwebpi -foreground
```

**Note:** For installations on IIS, restart IIS before starting the plug-in in foreground mode to release any existing shared memory.

Status and error messages are logged in the file configured in the **log-file**, **logs** and **log-entries** parameters in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file.

Plug-in auditing configuration is performed using the parameters in the **[aznapi-configuration]** stanza of the `pdwebpi.conf` configuration file.

For more information on auditing, logging and tracing or Security Access Manager data, refer to the *IBM Security Access Manager for Web: Auditing Guide*.

## Using the Common Auditing and Reporting Service

The Common Auditing and Reporting Service is a tool for security products, to collect audit events and allow for user specific reporting on the collected audit events. The product is separate to Security Access Manager and therefore not a prerequisite, requiring a separate installation on the participating plug-in. Use of the Common Auditing and Reporting Service for the plug-in is enabled by default if IBM Security Access Manager for Web uses the service.

There are no plug-in specific extensions to the Common Auditing and Reporting Service interface.

Common Auditing and Reporting Service audit configuration is handled separately to the auditing of data for previous versions of Security Access Manager and can coexist with logging of such data.

For more information on the Common Auditing and Reporting Service, refer to the *IBM Security Access Manager for Web: Auditing Guide*.

## Audit records

The plug-in captures authentication, authorization, and general server status audit events. The standard authentication audit events do not allow the correlation of these events to a specific virtual host. For this reason, the plug-in implements its own audit event category to capture virtual-host specific authentication information.

Standard authorization audit events do capture plug-in relevant virtual host information by virtue of the protected object name being constructed with the

/PDWebPI/*virtual_host_name* prefix. However the plug-in may apply several different authorization rules and other logic before reaching a definitive authorization decision.

For this reason, the plug-in implements its own audit event category that captures the final authorization result. These application level authorization events are reported by the plug-in to the Common Auditing and Reporting Service as HTTP Resource Access Events, although the plug-in has no way of determining what the final access result, returned by the hosting web server after the plug-in completes its intervention, will be.

Other events have also been developed that trace user session terminations through various logout activities.

Plug-in-specific authentication audit events are recorded in virtual-host-specific audit event pools constructed as follows:

`wpi.`*`virtual_host_name`*`.authn.`*`authentication_module_name`*`.successful`

and

`wpi.`*`virtual_host_name`*`.authn.`*`authentication_module_name`*`.unsuccessful`

Plug-in-specific authentication audit events conform to the DTD definition described in the *IBM Security Access Manager for Web: Base Administration Guide*.

Elements of the XML style 'wpi' audit records are described in this table.

*Table 6. Authentication audit record field definitions.*

| XML Tag | Description |
|---|---|
| <event> | Encapsulates tag for the audit record. The element includes an attribute describing the doc type definition revision of the record. |
| <date> | Record of the date and time the event occurred. |
| <outcome> | The tag element includes a **status** parameter that identifies the Security Access Manager or plug-in error code. The element describes the broad outcome of the event. The possible values are:<br>• 0 = Success<br>• 1 = Failure<br>• 2 = Pending<br>• 3 = Unknown<br>Events with an outcome of success are written to the successful pool, all others are written to the unsuccessful pool. |
| <originator> | Header tag for the originator section of the audit record. The tag element includes the **blade** parameter that identifies the Security Access Manager blade responsible for the event. |
| <event_id> | 105 |
| <component> | The tag identifies the component that captured the audit record. The component is recorded in the form: `wpi.`*`virtual_host_name`*`.`*`type_of_event`*`.`*`module_name`*` .successful` or `unsuccessful` |

*Table 6. Authentication audit record field definitions. (continued)*

| XML Tag | Description |
|---|---|
| <action> | Identifies the authentication method attempted. Action codes and their corresponding authentication mechanisms are:<br>16961 - BA<br>17236 - Client side certificate<br>17731 - Ecsso<br>17999 - Failover cookie<br>17997 - Forms<br>18504 - Http Header<br>18768 - IP address<br>4806211 - IV header: PAC credential<br>4806229 - IV header:user name<br>4806220 - IV header:Distinguished name<br>300609 - IV header:IP address<br>21579 - Token |
| <location> | Defines the server name that initiated the event. |
| <accessor> | Header tag for the accessor section of the audit record. Tag element can include the name of the accessor. |
| <principal> | The principal tag includes the parameter **auth** that identifies the authenticating directory service. The tag defines the validated user name. |
| <target> | The target tag includes the parameter **resource** that can be one of the following values:<br>• 0 = authorization<br>• 1 = process<br>• 2 = TCB<br>• 3 = credential<br>• 4 = general<br>Authentication audit records always set this value to 3 - credential. |
| <object> | Holds audit data that has little meaning for the authentication process. |
| <authntype> | Common Auditing and Reporting Service compatible authentication method name. |
| <authnprovider> | Configured name of the authentication module. |
| <data> | Extra authentication failure information. For example, failure during an authentication attempt using HTTP header information will result in an audit log record recording the failed HTTP header in this field. |

Plug-in-specific authorization audit events are recorded in virtual-host-specific audit event pools constructed as follows:
`wpi.`*`virtual_host_name`*`.access.successfulor`
`wpi.`*`virtual_host_name`*`.access.unsuccessful.`

Plug-in-specific authorization audit events conform to the DTD definition described in the *IBM Security Access Manager for Web: Base Administration Guide*.

Elements of the XML style 'wpi' access audit records are described in the table below.

*Table 7. Authorization audit record field definitions.*

| XML Tag | Description |
|---|---|
| <event> | Encapsulates tag for the audit record. The element includes an attribute describing the doc type definition revision of the record. |

*Table 7. Authorization audit record field definitions. (continued)*

| XML Tag | Description |
|---|---|
| <date> | Record of the date and time the event occurred. |
| <outcome> | The tag element includes a **status** parameter that identifies the Security Access Manager or plug-in error code. The element describes the broad outcome of the event. The possible values are:<br>• 0 = Success<br>• 1 = Failure<br>• 2 = Pending<br>• 3 = Unknown<br>Events with an outcome of success are written to the successful pool, all others are written to the unsuccessful pool. |
| <originator> | Header tag for the originator section of the audit record. The tag element includes the **blade** parameter that identifies the Security Access Manager blade responsible for the event. |
| <event_id> | 109 |
| <component> | The tag identifies the component that captured the audit record. The component is recorded in the form: `wpi.`*`virtual_host_name`*`.access.successful` or `unsuccessful` |
| <action> | 2 |
| <location> | Defines the server name that initiated the event. |
| <accessor> | Header tag for the accessor section of the audit record. Tag element can include the name of the accessor. |
| <principal> | The principal tag includes the parameter **auth** that identifies the authenticating directory service. The tag defines the validated user name. |
| <target> | Generally audit access records always set this this value to 3 - credential. |
| <resource_access> | The URL requested and an <httpresponse> value. For the web plug-in the http response will not be set to the final response as that can only be obtained from the hosting web server after the plug-in passes on the authorized requests for handling.<br>It is possible the client could request a non-existent page, which might be permitted theoretical access by the authorization server but ultimately results in a 404 Not Found from the web server. |
| <data> | Not used. |

Plug-in-specific session termination audit events are recorded in virtual-host-specific audit event pools constructed as follows:
`wpi.`*`virtual_host_name`*`.authn.`*`authentication_module_name`*`.successful`

Plug-in-specific session termination audit events conform to the DTD definition described in the *IBM Security Access Manager for Web: Base Administration Guide*.

Elements of the XML style 'wpi' audit records are described in the table below.

*Table 8. 'wpi' audit record field definitions.*

| XML Tag | Description |
|---|---|
| <event> | Encapsulates tag for the audit record. The element includes an attribute describing the doc type definition revision of the record. |

*Table 8. 'wpi' audit record field definitions. (continued)*

| XML Tag | Description |
|---|---|
| <date> | Record of the date and time the event occurred. |
| <outcome> | 0 = Success |
| <originator> | Header tag for the originator section of the audit record. The tag element includes the **blade** parameter that identifies the Security Access Manager blade responsible for the event. |
| <event_id> | 103 |
| <component> | The tag identifies the component that captured the audit record. The component is recorded in the form: `wpi.`*`virtual_host_name.type_of_event.module_name`*`.successful` |
| <action> | 103 |
| <location> | Defines the server name that initiated the event. |
| <accessor> | Header tag for the accessor section of the audit record. Tag element can include the name of the accessor. |
| <principal> | The principal tag includes the parameter **auth** that identifies the authenticating directory service. The tag defines the validated user name. |
| <target> | The target tag includes the parameter resource that can be one of the following values:<br>• 0 = authorization<br>• 1 = process<br>• 2 = TCB<br>• 3 = credential<br>• 4 = general<br>Authentication audit records always set this value to 3 - credential. |
| <authntype> | Common Auditing and Reporting Service compatible authentication method name. |
| <terminateinfo> | Container for a <terminatereason> descriptive string. |
| <data> | Not used. |

## Auditing configuration

The following table displays the basic auditing configuration parameters and explains their function. The basic auditing configuration does not afford much flexibility in filtering the type of audit events to be captured. Greater control is possible using **logcfg** configurations to capture audit events. **logcfg** configuration is more fully described in the *IBM Security Access Manager for Web: Base Administration Guide*.

*Table 9. Basic auditing configuration parameter definitions*

| Parameter | Description |
|---|---|
| **logsize** | The size (in bytes) at which the log files rollover to a new file. If set to 0 the log files will not rollover. A negative number will roll the logs over daily regardless of size. |
| **logflush** | The interval in seconds at which the logs are flushed. Maximum of 6 hours and a default of 20 seconds. |
| **logaudit** | Enables or disables auditing. |

*Table 9. Basic auditing configuration parameter definitions (continued)*

| Parameter | Description |
|---|---|
| **auditlog** | Specifies the name of the audit file |
| **auditcfg** | Enables or disables authorization and/or authentication auditing. |

For example:

```
[aznapi-configuration]
logsize = 2000000
logflush = 20
logaudit = no
auditlog = audit.log
auditcfg = azn
#auditcfg = authn
auditcfg = wpi
```

**logcfg** auditing configuration overrides any basic auditing configuration. The following tables list the format of the **logcfg** command and the audit event categories that may be extracted from the web plug-in.

*Table 10. logcfg auditing configuration parameter definitions.*

| Parameter | Description |
|---|---|
| logcfg | Category (`stdout\|stderr\|file\|pipe\|remote`) parameters. Allows an event logger to be attached to a category of events. |

*Table 11. Audit event pools. These are values which may be specified for the category part of a logcfg configuration.*

| Category | Description |
|---|---|
| audit.authn | Authentication events logged by the authorization api. |
| audit.azn | Authorization events logged by the authorization api. |
| audit.wpi.*virtual_host._name*.authn.authentication *_module_name*.successful | Successful authentication events logged by the {*authentication_module_name*} authentication module for the {*virtual_host_name*} virtual host. |
| audit.wpi.*virtual_host._name*.authn.authentication *_module_name*.unsuccessful | Authentication attempts that failed, logged by the {*authentication_module_name*} authentication module for the {*virtual_host_name*} virtual host. |
| audit.wpi.*virtual_host._name*.access.successful | Access decision events logged by the {*virtual_host_name*} virtual host. The success event category records events where it was possible to make an access decision irrespective of whether that decision outcome was permitted or denied. |

*Table 11. Audit event pools. These are values which may be specified for the category part of a logcfg configuration. (continued)*

| Category | Description |
|---|---|
| audit.wpi.*virtual_host._name*.access.unsuccessful | Access decision events logged by the {*virtual_host_name*} virtual host. The unsuccessful event category records events where it was not possible for the plug-in to reach a definitive access decision. |

Event categories are inclusive of sub categories. Configuring the capture of the `audit.wpi.virtual_host_name` category of events will capture all of the events for which `audit.wpi.virtual_host_name` is the root portion of an events category. Hence, capturing the all encompassing 'audit' category will record all audit events.

## Tracing Plug-in actions

Security Access Manager Plug-in for Web Servers provides the ability to trace actions and store the results on file for the purposes of debugging. Primarily, tracing is an analysis and problem diagnosis tool used by application support to gain a complete view of actions causing errors. As a user, you might find some of the plug-in tracing facilities useful, although the majority will be of little benefit unless you are diagnosing complex problems.

It is possible to trace the HTTP messages at the plug-in. This can be very useful because it shows exactly what is received from the user and what is returned to the user, even if the communication is over HTTPS. Trace is enabled and disabled using the standard **pdadmin** tracing commands.

The inputs to and results of authorization decisions can be traced to facilitate diagnosis of authorization policy configuration problems. This tracing shows user credential information including names, UUIDs, session ID, and attributes. This tracing also shows the name of the Security Access Manager protected object being used to make the decision and the permissions required. The result of the decision is also shown along with any returned decision attributes.

### pdadmin trace commands

You can use **pdadmin** commands to specify tracing.

**Listing trace components:**

The **list** command produces a list of all the plug-in actions that can be traced.

Syntax:

```
pdadmin> server task PDWebPI-server-name trace list [component]
```

The majority of trace tasks listed are specific to Security Access Manager. Plug-in-specific trace items are prefixed with **pdwebpi**.

**Setting trace components:**

There are four main trace items that you might find useful for debugging:

- pdwebpi.request

- pdwebpi.plugin
- pdwebpi.azn
- pdwebpi.session

When **pdwebpi.request** is set to *two*, tracing occurs on every request that passes through the plug-in. When **pdwebpi.request** is set to *nine*, the request header is included in the trace.

**Note:** The **pdwebpi.request** component traces the entire HTTP request that the server receives, which might include sensitive information.

**pdwebpi.plugin** activates trace in the plug-in server. All messages are sent to the log file on the web server or, in the case of IIS, to a log different from that used for the authorization server.

**pdwebpi.azn** set to *one*, traces information about every authorization decision including protected object name, permission string, user name, session ID, HTTP method, HTTP URI and decision result. When **pdwebpi.azn** is set to *two*, tracing occurs for additional credential attribute information and both input and output decision attributes. When **pdwebpi.azn** is set to *five*, additional information is included about step-up and reauthentication processing.

**pdwebpi.session** set to *seven*, enables tracing of information about a session of a user.

Another useful trace item is **pdwebpi.proxy-cmd**. This trace item shows all information or commands that are passed back from the authorization server to the plug-in (for example, headers which are set).

The trace **set** command has the following syntax:

```
pdadmin> server task PDWebPI-server-name trace set component
 level [file path=file|other-log-agent-config]
```

Where *component* is the name of the trace component as shown by the list command. A trace is set for this component. *level* is the amount of detail gathered for the trace. The range is 1 to 9, with 1 being the least detailed and 9 being the most detailed.

The optional **file path** parameter specifies the location for trace output. Trace output by default is sent to the standard configured plug-in log file (except when using the component **pdwebpi.plugin**). For IIS installations, the name of the file to which the plug-in component trace is sent to is always configured using the **log-file** parameter under the **[iis]** stanza in the configuration file.

Output can be sent to the screen by the **pdadmin** daemon using the **-foreground** option. That is:

```
pdwebpi -foreground
```

**Showing trace components:**

To show trace components use the **show** command in the following form:

```
pdadmin> server task PDWebPI-server-name trace show [component]
```

## Cache database settings

Plug-in cache database configuration is performed using the parameters in the **[aznapi-configuration]** stanza of the `pdwebpi.conf` configuration file.

You can configure the plug-in 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 **disable**.

```
[aznapi-configuration]
cache-refresh-interval = 60
```

The **db-file** parameter defines the full path to the ACL cache database. By default this parameter is not set.

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

The **listen-flags** parameter enables or disables the reception of policy cache update notifications. A "disable" value disables the notification listener. This parameter is set by the **svrsslcfg** utility.

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

## Plug-in statistics

Security Access Manager Plug-in for Web servers provides the ability to monitor and collect information about specific server activity. The gathered statistics information can be displayed and redirected to log files if required.

You can enable statistics reporting dynamically using the **pdadmin** command, **pdadmin stats on**, or statically with configuration parameters in the plug-in configuration file.

Full details of both these approaches, including statistics components available to the plug-in, are described in the Web plug-in section of the *IBM Security Access Manager for Web: Auditing Guide*.

## Configuring the authorization API service

The **[aznapi-entitlement-services]** stanza of the `pdwebpi.conf` configuration file assigns service IDs to services.

Each stanza entry defines different types of aznAPI service. See the *IBM Security Access Manager for Web: Administration C API Developer's Reference* for details.

Each entry is in the form:

*service_id* = *path_to_dll* [ & *params* ... ]

Service IDs are used by the aznAPI client to identify the services. You can specify parameters to pass to the service when it is initialized by the aznAPI. Parameters follow the & symbol in the entry.

## Credential refresh

The information gathered at the time of user authentication is cached in the credential for the duration of the session. See "Credential acquisition" on page 6 for more details on plug-in user credentials.

Unless configured for credential refresh, changes made to the source of the user credential information, such as adding or removing the user from a group, are not reflected in the session of the user until a new user session is created.

Some reasons why credential refresh is useful:

- You are able to refresh a credential of a user without requiring them to logout and re-login to an application. This improves the user experience.
- It provides an administrator with the ability to supply extra access to secure Web objects for a user during their current session.
- It improves security by allowing an administrator to restrict access permissions of a user during a current session if the administrator has reason to believe that the user is not behaving appropriately.

When refreshing credentials you may find it necessary to preserve some information about the user from the original credential. For example, a custom attribute may record the login time of a user which would be of value to keep unchanged when the credential is refreshed. The plug-in allows you to configure attributes to keep when a refresh is performed. These are configured based on the attribute name.

The **[cred-refresh]** or **[cred-refresh:***virtual-host***]** stanza specifies the attributes to preserve from the original credential and which attributes to refresh into the new credential when a credential refresh operation takes place.

The values are specified in the form **preserve =** *attribute_pattern* for attributes to preserve from the original. Values to refresh are specified in the form **refresh =** *attribute_pattern*. The standard plug-in pattern matching rules apply to the attribute patterns, with the exception that character comparisons are case insensitive. See Appendix F, "Special characters allowed in regular expressions," on page 295 for more information on the allowed matching rules.

The following rules apply to the attribute list:

- Rules that appear earlier in the stanza have precedence over those that appear later.
- Rules that do not match any attributes are refreshed by default.
- Certain attributes are always preserved, regardless of how the **[cred-refresh]** stanza is configured. The attributes are:
  - AZN_CRED_AUTHNMECH_INFO
  - AZN_CRED_BROWSER_INFO
  - AZN_CRED_IP_ADDRESS
  - AZN_CRED_PRINCIPAL_NAME
  - AZN_CRED_QOP_INFO
  - AZN_CRED_NETWORK_STR
  - AZN_CRED_NETWORK_BIN
  - AZN_CRED_IP_FAMILY

- Attributes marked by the aznAPI as read-only are always preserved, regardless of the contents of the **[cred-refresh]** stanza.
- If an attribute is not present in the original credential, it will not be preserved, regardless of the configuration of this stanza.

**Note:** A credential is only refreshed from the registry when the credential does not exist in the credential cache.

Once credential refresh functionality is configured you can use the **pdadmin** command line utility to refresh the credential of a particular user. The example below shows the commands for adding a user to a new group and refreshing the credential while the user is still logged on, effectively giving the user access permissions of the new group.

```
pdadmin> group modify group_name add user_name
pdadmin> server task server_name refresh all_sessions user_name
```

### Configuring credential refresh

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods.

Configure the **cred-refresh** module as a pre-authorization module to enable the credential refresh feature.

```
[common-modules]
...
pre-authzn = cred-refresh
...
```

Ensure that the entry for credential refresh exists in the **[modules]** stanza of the `pdwebpi.conf` configuration file:

```
[modules]
...
cred-refresh = pdwpi-cred-refresh-module
...
```

## Configuring HTTP request caching

The plug-in caches request data and uses this cached data to rebuild a request during a HTTP redirect, if a re-authentication requirement interrupts the completion of the request processing. This functionality benefits POST and PUT requests, because these requests types can include a variety of information fields.

When an authentication requirement interrupts a request, the plug-in 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 plug-in credentials/session cache.

Upon successful authentication (or re-authentication), the plug-in sends a HTTP redirect to the browser. The browser follows the redirect to the original URL contained in the redirect. The plug-in intercepts the redirect and rebuilds the request using the cached data. The rebuilt request is delivered to the URL destination.

### Configuring server-side caching parameters

The **max-cached-http-body** parameter in the **[pdweb-plugins]** stanza of the plug-in configuration file specifies the maximum amount of HTTP body data that is cached for any given request. When the amount of body data exceeds the configured maximum, all of the body data is discarded.

The **worker-size** parameter within the **[proxy-if]** stanza controls the amount of memory allocated for any given request. The max-cached-http-body size, at a minimum, should conform to the following algorithm:

**max-cached-http-body** * 4/3 * 2 + 3000 <= **worker-size** # #

This algorithm assumes that 3000 bytes is enough memory to hold the request less any POST data, and the returned form, less the cached POST data. If the size of the request plus the size of the returned form is likely to exceed 3000 bytes you should either increase the **worker-size** entry or decrease the **max-cached-http-body** value.

## FIPS cryptographic compliance

The plug-in operations affecting Federal Information Process Standards (FIPS) 140-2 compliance are:

- Session ID generation
- Encryption and decryption of the failover cookie
- Encryption and decryption of eCSSO tokens
- Encryption and decryption of CDSSO tokens
- Encryption and decryption of LTPA cookies

FIPS 140-2 compliant cryptographic operations within the plug-in are determined at RTE configuration time, using the default Security Access Manager setting within the `pd.conf` file. FIPS 140-2 compliance is disabled by default.

The optional `ssl-fips-enabled` setting within the **[dsess]** stanza of the configuration file determines whether or not Federal Information Process Standards (FIPS) mode is enabled on the session management server. If a FIPS setting other than that set for the policy server is required, this setting must be manually updated by the administrator. For more details, see Appendix C, "Plug-in configuration file reference," on page 211.

## Language support and character sets

Security Access Manager Plug-in for Web Servers can display Security Access Manager generated HTML pages in the preferred language of the customer. The language used for the display in HTML pages is taken from the standard *Accept-Language* header found within the HTTP request. Language values are represented by two characters. 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)

If the plug-in finds no appropriate language code in the HTTP request, it retries the language list without the qualifying dialect (e.g. es-MX is retried as es). If an appropriate language is still not found, the server uses English.

Only the Security Access Manager generated pages, contained within *install_path*/nls/html/*lang*/*charset*, are served in multiple languages. Examples of these pages include all Security Access Manager authentication forms and the Security Access Manager account management pages.

The languages specified within the *Accept Language* field in the HTTP header are mapped directly to directories found within the *install_path*/nls/html directory. A server can be modified to cater for language specifier variations by copying language directories. To modify a server, the actual language directories that should be copied are :

*am base install directory*/nls/msg/*lang*
*am webpi install directory*/nls/html/*lang*/*charset*
*am webpi install directory*/nls/msg/*lang*/*charset*

The following table lists the languages supported by the plug-in, with the associated sub-directory name:

*Table 12. Plug-in supported languages with supported directory.*

| Language | System Directory |
|---|---|
| English (default) | C |
| Czech | cs |
| German | de |
| Spanish | es |
| French | fr |
| Hungarian | hu |
| Italian | it |
| Japanese | ja |
| Korean | ko |
| Polish | pl |
| Portuguese, Brazil | pt_BR |
| Russian | ru |
| Chinese, China | zh_CN |
| Chinese, Taiwan | zh_TW |

Up to ten language specifications in the *Accept Language* field of the HTTP header are recognized.

Different character sets for a given language are located in a directory below that for language support. The character set directory used is selected based on the value of the **accept-charset** header received from the client. If no match is found (or if the header is not set) then the **utf-8** directory is used.

You can enable and disable the support for different languages and character sets based on the **accept-language** and **accept-charset** headers. The default settings for these parameters are configured in the **[pdweb-plugins]** stanza but can be overridden on a virtual host basis by defining them in a stanza with the name of the virtual host ID.

```
[pdweb-plugins] or [virtual-host]
...
use-accept-langauge-header = true
...
use-accept-charset-header = false
```

By default, the use of the accept-charset header is disabled.

The **use-accept-language-header** parameter enables or disables the use of the *accept-language* HTTP header when attempting to locate the language for the generated HTML response.

The **use-accept-charset-header** parameter enables or disables the use of the *accept-charset* HTTP header when attempting to locate the charset in which to decode elements of a HTTP request, or generate a HTML response. The default value (if not found within this configuration file) is *false*.

The *user-agent* header can be used as an alternative to the *accept-language* and *accept-charset* headers for selecting a language and character set. The *user-agent* header contains device-specific information that can be used to provide language and character information when the requirements of the device are known. The *user-agent* header is only used when either or both *accept-language* and *accept-charset* headers are not found, or if the use of those headers is disabled.

The default mapping from *user-agent* to language and character set is configured in the **[user-agent]** stanza and can be overridden on a per virtual-host basis. The stanza includes a list of patterns that are matched, in the order specified, to the contents of the *user-agent* header. For a list of the available wildcard characters refer to Appendix F, “Special characters allowed in regular expressions,” on page 295.

If a match is found then the directory for the corresponding language and charset is used. In addition to specifying a language and character set for a given *user-agent* pattern, it is also possible to specify a directory. In this case the directory name specified is used rather than the one for the charset when sending Security Access Manager pages. This directory must be located under the specified language directory.

# Chapter 3. Authentication and request processing

Authentication is the method of identifying an individual that is attempting to log on to a secure domain. Successful authentication results in a Security Access Manager identity that represents the user. The plug-in uses this identity to acquire credentials for the user which are then used to permit or deny access to protected resources depending on access control permissions, policy conditions, and authorization rules for the resource.

Security Access Manager Plug-in for Web Servers supports several authentication methods by default and can be customized to use others.

This chapter discusses how the plug-in maintains session state, handles the authentication process, and performs any post authorization processing required on authorized sessions.

This chapter includes the following topics:



## Configuring authentication

All available authentication methods with their associated shared library names are defined in the **[modules]** stanza of the `pdwebpi.conf` configuration file.

The **[modules]** stanza also lists the modules that are used for session identification and post-authorization handling. These modules are described later. The shared libraries must exist in the `pdwebpi/lib` directory. Shared library names are specified

without any operating-system-specific prefix (such as `lib`) and any operating-system-specific suffix (such as `dll`). For example:

```
BA = pdwpi-ba-module
```

In the preceding example, the BA module library is given as `pdwpi-ba-module`.

- On Windows, the plug-in looks for a file that is called `pdwpi-ba-module.dll`.
- On Solaris, the plug-in looks for a file called `libpdwpi-ba-module.so`.
- On AIX, the plug-in looks for a file called `libpdwpi-ba-module.a`.

**Note:** An alternative to the default search path for library files can be defined in the **[module-mgr]** stanza.

Each label that is defined in the **[modules]** stanza has a corresponding stanza of its own, for example **[BA]**, **[cert]**, and **[token]**. Specific configuration information for each authentication method is specified in these stanzas and applies to that authentication method independent of which virtual-host it is called from. If special configuration is required on a per virtual-host basis, then the default configuration can be overridden by using a stanza that qualifies the module label with a virtual-host label. For example:

```
[BA]
basic-auth-realm = "Access Manager"

[BA:ibm.com]
basic-auth-realm = "ibm.com"
```

In the example, users who access virtual host ibm.com by using Basic Authentication is subject to the configuration entries specified in the stanza **[BA:ibm.com]**.

A standard configuration of modules permits only one instance of a module library to be specified for an authentication method, for example:

```
[modules]
BA = pdwpi-ba-module
```

Some installations might require multiple instances of an authentication library to be specified. The requirement might be done when different behavior of a module is required for different authentication levels. The following example shows the configuration for two instances of the forms authentication module.

```
[modules]
BA = pdwpi-ba-module
forms-authn-level1 = pdwpi-forms-module
forms-authn-level2 = pdwpi-forms-module

[common-modules]
authentication = forms-authn-level1
authentication = forms-authn-level2
authentication = BA

[forms-authn-level1]
login-form = level1-form

[forms-authn-level2]
login-form = level2-form

[BA]
...
```

The last step in configuring authentication methods is to specify the authentication methods. The authentication methods are set in the **[common-modules]** stanza of the configuration file in their order of preference. For example:

```
[common-modules]
session = ssl-id
session = BA
session = session-cookie

authentication = cert
authentication = BA

post-authzn = ltpa
```

In the preceding example, the configuration settings ensure that:

- SSL session IDs are used to maintain session information as a first choice.
- BA headers, if available are used to maintain session information when an SSL session ID is not available.
- Session cookies are used as a last resort to maintain session information when neither SSL session IDs or BA headers are available.
- Certificates are used as the authentication method as a first choice.
- BA is used for authentication when a certificate is not available.
- LTPA cookies are to be added to the request as part of post-authorization processing.

## Configuring authentication for virtual hosts

Configuration of authentication methods can be achieved on a per virtual host basis by specifying the methods directly in each virtual host stanza. For example:

```
[pdweb-plugins]
virtual-host = ibm.com

[ibm.com]
....
session = ssl-id
session = BA
session = session-cookie

authentication = cert
authentication = BA

post-authzn = ltpa
```

An alternative way to specify the authentication methods for virtual hosts is to define a stanza for authentication method configuration. This allows multiple virtual-hosts to share a module configuration. The module configuration stanza is specified by the **modules** configuration entries in the virtual-host stanza. For example:

```
[pdweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com

[ibm.com]
modules = ibm-lotus-module-stanza

[lotus.com]
modules = ibm-lotus-module-stanza
```

```
[ibm-lotus-module-stanza]
authentication = BA
session = BA
post-authzn = ltpa
```

When separate stanzas for authentication method configuration on a per-virtual host basis are not defined in the configuration file, all virtual hosts use the entries configured in the **[common-modules]** stanza; that is, the default value for the **modules** stanza entry is *common modules*.

The following example sets up a virtual host called ibm.com that is configured to use SSL session IDs where it can, BA headers where it can't use an SSL ID and has BA headers, and uses session cookies as a last resort to maintain session information. It supports certificate authentication ahead of basic authentication and on successful authentication adds an LTPA cookie to the request to be handled by the Web server. The example only shows the entries defined here.

```
[pdweb-plugins]
virtual-host = ibm.com

[modules]
ssl-id = pdwpi-ssl-id
session-cookie = pdwpi-session-cookie
BA = pdwpi-ba
cert = pdwpi-cert
ltpa = pdwpi-ltpa

[ibm.com]
session = ssl-id
session = BA
session = session-cookie

authenitcation = cert

post-authzn = ltpa
```

Further configuration of authentication can be achieved on a per virtual host basis by creating virtual host specific authentication configuration stanzas. The example below shows the configuration for two virtual hosts: ibm.com and lotus.com. Each virtual host has module specific authentication configuration.

```
[pdweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com

[modules]
...

[ibm.com]
session = BA
session = session-cookie

authenitcation = BA
authentication = forms

[lotus.com]
session = session-cookie

authenitcation = BA
authentication = cert

[BA:ibm.com]
basic-auth-realm = "Access Manager - ibm.com"
```

```
[BA:lotus.com]
basic-auth-realm = "Access Manager - lotus.com"
```

## Configuring the order of authentication methods

Security Access Manager Plug-in for Web Servers supports a variety of authentication methods in a way that can be tailored for different requirements and for different security needs. The type of authentication methods you choose needs to be carefully considered and implemented in a way that is fail safe and achieves your security objectives. When configuring authentication methods, the order in which they appear in the configuration file is essential to the correct operation of your plug-in software.

The flow chart below shows the plug-in logic used to select an authentication module.

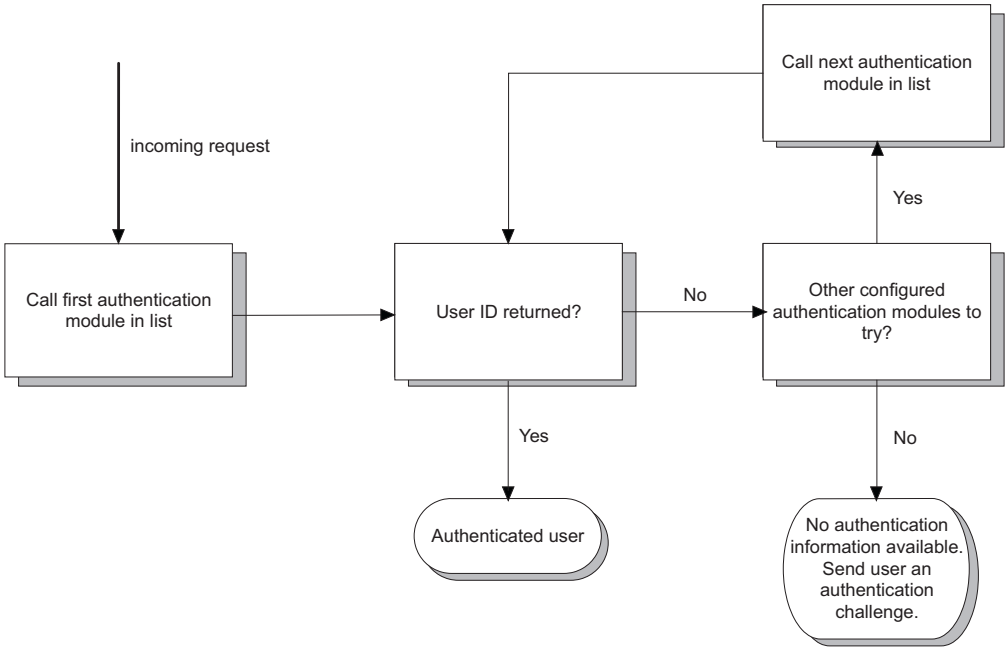


*Figure 2. Plug-in process flow for determining authentication module.*

The plug-in calls each authentication module in the order it was configured until one of the modules returns a credential for the user. If none of the configured authentication modules is able to generate a credential, an authentication challenge is sent to the user to prompt them to provide authentication information.

If an authentication challenge is required, then the first suitable authentication module from the configured list is called to generate the commands needed to produce the challenge. Not all authentication modules can generate a challenge.

For example, there is no challenge to request HTTP Headers — these are either present in the request or not. In addition, an authentication module might be unavailable because it is already being used to identify a proxy agent that is forwarding requests to the plug-in. The most common authentication mechanisms that can generate a challenge for the user are Basic Authentication (a BA challenge is sent to the user) and forms-based authentication (a logon form is sent to the

user). If no authentication method is available, the user cannot be authenticated and the plug-in returns a "Forbidden" page.

The flowchart in figure 2 shows the process for selecting an authentication method to send a challenge to the user.

Each configured authentication method is examined in the order in which it is configured until one is found that satisfies the required level of authentication. If a module is found that satisfies the authentication criteria, it is called to build the challenge that is sent to the user.

If none of the configured authentication methods is suitable, then no authentication is possible. The plug-in returns a "Forbidden" page to the user because the user does not have the permissions required to access the requested resource and there is no possibility to send them a challenge to authenticate at the required level.

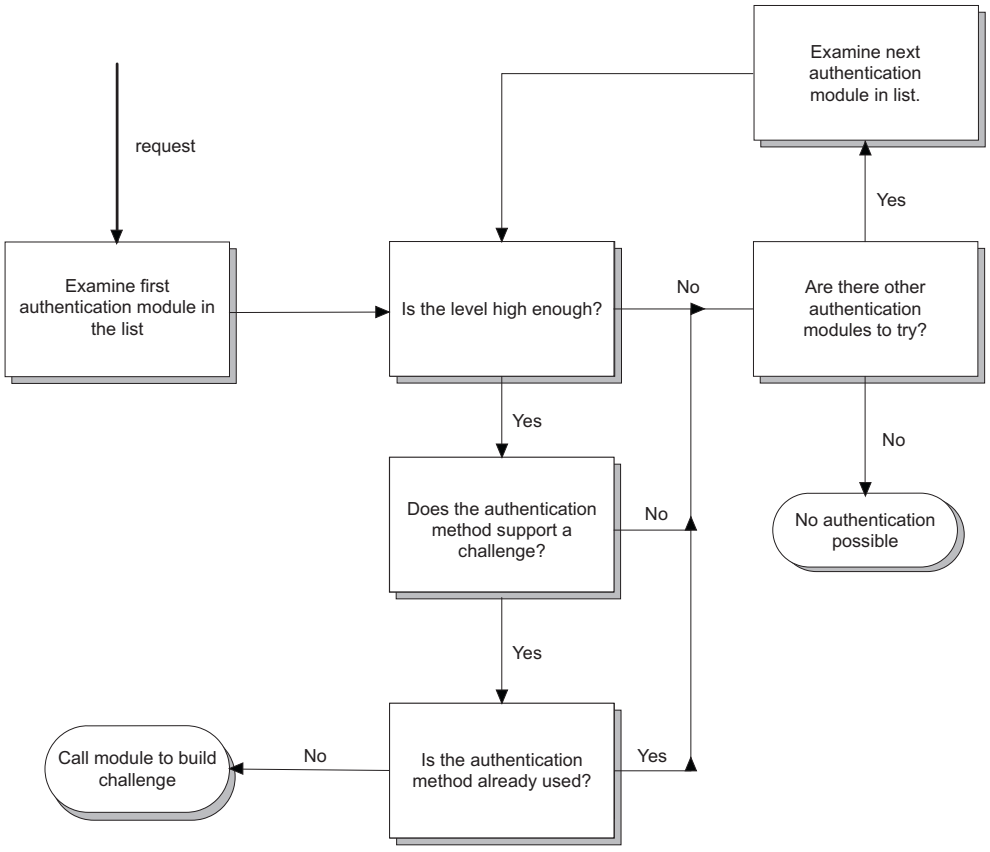


*Figure 3. Authentication challenge process logic.*

As discussed previously, the **[common-modules]** stanza of the `pdwebpi.conf` configuration file is where you specify the authentication methods you want to use. The **[authentication-levels]** stanza of the configuration file defines step-up authentication levels (see "Authentication-strength Protected Object Policy (Step-up)" on page 136) as well as the ordering of authentication methods configured in the **[common-modules]** stanza.

An authentication method defaults to a level of 1 when no entry for it is defined in the **[authentication-levels]** stanza. Authentication order is then determined as the

highest authentication level down to the lowest authentication level for the authentication methods defined in the **[authentication-levels]** stanza. If an authentication level is shared by several modules, the sub-order is then determined by the order in which the modules appear within the **[common-modules]** stanza.

To understand plug-in authentication it is useful to think of the plug-in asking two questions for each request it processes:

1. Can I authenticate this request using the configured method of authentication?
   If the answer to this question is no, then the plug-in asks the next question.
2. Can I generate an authentication request using the configured method of authentication?

Consider the following configuration.

```
[common-modules]
authentication = BA
```

For an incoming request, authentication of the user is required if the ACL does not permit unauthenticated users. The plug-in seeing BA as the only authentication method configured, asks; "can I authenticate this request using basic authentication?" If the request is new then the answer is no—the plug-in does not know of this user. The plug-in then asks; "can I generate an authentication request using basic authentication?" If basic authentication has been configured correctly, the answer is yes. The plug-in prompts the user for an ID and password.

This is a simple example of authentication using Basic Authentication. It is likely that you will want to configure more than one authentication method depending on the security requirements of your object space.

Following is a more detailed example of the logic that the plug-in uses to give priority to configured authentication methods.

The authentication logic discussed in the following paragraphs assumes that unauthenticated users are not permitted to access the resource and that the following configurations have been made to the `pdwebpi.conf` configuration file.

```
[common-modules]
authentication = BA
authentication = failover
authentication = forms

post-authzn = failover

[authentication-levels]
1 = BA
2 = failover
```

The preceding configuration specifies three authentication methods: BA, failover cookies, and forms. Failover cookies are used for post-authorization processing. The levels set in the **[authentication-levels]** stanza determine the order in which the authentication methods are called to authenticate requests. Forms authentication defaults to a level of 1 as no level has been defined for it in the **[authentication-levels]** stanza.

Using the above configuration, the plug-in, when receiving a request, looks for a failover cookie in the request header. The plug-in looks for a failover cookie before BA data because failover is specified at level 2 in the **[authentication-levels]** stanza. The **[authentication-levels]** stanza takes precedence over the order of the authentication modules definitions in the **[common-modules]** stanza.

The plug-in asks the question, "Can I authenticate this request using a failover cookie?" If the request has not been previously authenticated then the answer is no, as the plug-in will have not previously constructed a failover cookie for the request. The plug-in asks the second question, "Can I generate an authentication request using the failover cookie?" The answer is no, because the failover cookie module has no way of generating requests for authentication.

The plug-in moves to the next configured authentication method in the **[authentication-levels]** stanza, which in the example is BA. The plug-in asks the question; "can I authenticate this request using the BA header?" The answer is no, as the request has not been previously authenticated.

The plug-in then asks the question; "can I generate an authentication request using BA?" The answer is likely to be yes, and the user is prompted to enter a user ID and password. A successful authentication produces an authorized session and a failover cookie is inserted into the request header and used as the first method of authentication for subsequent requests during the same session.

Should the BA module be unable to generate a method for authenticating the user, the plug-in would default to the ordering of methods listed in the **[common-modules]** stanza of the configuration file. In the configuration example above, the plug-in would assign the priority of authentication methods thus:

level 1 = BA, forms
level 2 = failover cookie

If failover cookies and BA fail to provide a method for user authentication, the plug-in would authenticate using forms.

## Configuring post-authorization processing

Configured post-authorization modules are called after a request has been authorized. Post-authorization modules determine if any other action needs to be taken before a request is passed back to the plug-in for processing by the Web server. All configured post-authorization modules are called to determine if any need to take action on the request.

Post Authorization modules can be categorized as either:

- **Modifying Requests for SSO** — These post-authorization modules add information (cookies or headers) that are used by the Web application to identify the user without requiring a second authentication.
- **Modifying Responses** — These post-authorization modules modify responses usually by adding headers or cookies to it. For example, the failover module adds a failover cookie to responses.
- **Special Functions** — These post-authorization modules recognize the URI being requested as the trigger for some special function. A special function indicates that the request is handled by the plug-in. An example is eCSSO "vouch for" requests.

Post-authorization modules are called in the order they appear in the configuration file. Post-authorization modules specified "later" in the list have the ability to undo or overwrite changes made by prior post-authorization modules.

For example; the following configuration will result in different plug-in behavior depending on the order that *BA* and *forms* are specified in the **[common-modules]** stanza.

```
[common-modules]
...
post-authzn = BA
post-authzn = forms

[BA]
...
strip-hdr = always

[forms]
...
create-ba-hdr = yes
```

The configuration above is a simple example of how a great deal of flexibility can be achieved through the ordering of post-authorization modules and module configuration.

## Authentication configuration overview

As seen in the section "Configuring authentication" on page 49, it is the authentication modules that perform the process of extracting authentication information from requests. The actual authentication of requests is performed by authentication mechanisms which validate authentication information. The separation of roles between authentication modules and authentication mechanisms allows custom libraries, developed against the external authentication C API and written for WebSEAL, to be used with the plug-in.

The mechanisms for all authentication methods supported by Security Access Manager Plug-in for Web Servers are configured in the **[authentication-mechanisms]** stanzas of the `pdwebpi.conf` configuration file. Supported authentication method entries include:

- Local (built-in) authenticators

  Parameters for local authenticators specify the appropriate built-in shared library (UNIX) or DLL (Windows) files.
- Custom external authenticators

  The plug-in provides template server code that you can use to build and specify a custom authentication mechanism against the external authentication C API.

Unlike the configuration for the **[modules]** stanza, the full file name is required when configuring mechanisms in the **[authentication-mechanisms]** stanza. That is, include the file prefix and the operating-system-specific extension.

### Local authentication mechanisms

The following authentication mechanism entries specify local built-in authenticators:

*Table 13. Local Built-in Authenticators*

| Configuration Entry | Description |
|---|---|
| Forms and Basic Authentication | |
| **passwd-ldap** | Client access with registry user name and password. |
| Client-side Certificate Authentication | |

*Table 13. Local Built-in Authenticators (continued)*

| Configuration Entry | Description |
|---|---|
| **cert-ssl** | Client access using a client-side certificate over SSL. |
| HTTP Header, IP Address Authentication, IV Header with iv-remote-address activated. | |
| **http-request** | Client access through special HTTP header, IP address, or IV Header with iv-remote-address activated. |

Use the **[authentication-mechanisms]** stanza to configure the authentication method and the implementation in the following format:

*authentication_method_parameter = shared_library*

## External custom authentication mechanism entries

The following entries are available to specify custom shared libraries for external authentication mechanisms:

*Table 14. External Authentication Mechanism Entries*

| Configuration entries | Description |
|---|---|
| **passwd-cdas** | Client access with user name and password for a third-party registry. |
| **token-cdas** | Client access with registry user name and token passcode. |
| **cert-cdas** | Client access using a client-side certificate over SSL. |

In addition to the authentication libraries there are two other standard Security Access Manager libraries that can be used in the plug-in:

- **passwd-strength**

  This library checks new passwords entered on the password change form.
- **cred-ext-attrs**

  This library allows custom attributes (name/value pairs) to be specified for inclusion in the credential.

See the *IBM Security Access Manager for Web: Web Security Developer Reference* for details on building and configuring a custom shared library that implements an external authentication mechanism.

## Default configuration for plug-ins

By default the plug-in is set to authenticate clients using Basic Authentication (BA) user names and passwords.

The plug-in is normally enabled for both TCP and SSL access. Therefore, a typical configuration of the **[authentication-mechanisms]** stanza includes support for user name and password and support for client-side certificates over SSL.

The following example represents the typical configuration of the **[authentication-mechanisms]** stanza on Solaris:

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
cert-ssl = pdwpi-sslauthn.so
```

To configure other authentication methods, add the appropriate entry with its shared library.

## Configuring multiple authentication methods

Modify the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` 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:

1. All authentication methods can function independently from each other. It is possible to configure a shared library for each supported method.
2. The **cert-cdas** method overrides the **cert-ssl** method when both are configured. You must enable one of these to support client-side certificates.
3. Only one password type authenticator is actually used when more than one is configured. The plug-in uses the following order of priority to resolve multiple configured password authenticators:
   a. **passwd-cdas**
   b. **passwd-ldap**
4. 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 user name/password and HTTP header authentication. For this example, you would configure both the **passwd-cdas** > **http-request** entries with the same shared library. It is the responsibility of the developer to maintain session state and avoid conflicts between the two methods.

## Logout, change of password and help commands

Security 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. When using an authentication method that supplies authentication data with each request, the **pkmslogout** command clears the session cache although credential information is still contained in the request header. In this case, the user must close the browser to fully log out of the session.

The **pkmslogout** command is appropriate for authentication using token passcodes, Forms authentication, and certain implementations of HTTP header authentication.

Run the command as follows:

```
https://www.ibm.com/pkmslogout
```

The browser displays a logout form defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
logout-success = logout_success.html
```

The logout-success entry can specify either a pre-defined HTML file (contained within the base *install_path*`/nls/html/C` directory) or a URI. The specified URI could be either a relative URI or an absolute URI.

**Note:** When using Internet Explorer, cookie prompting must be switched off (the default browser setting). Otherwise, the cookie cache within the browser may become corrupted, causing the logout process to hang. To disable cookie prompting from within Internet Explorer, select **Internet Options** from the **Tools** menu, then click on the **Advanced** button under the **Privacy** tab and ensure the check box is not selected.

The **pkmslogout** command also allows the default HTML response page (such as `logout.html`) to be replaced by a custom response page. The custom response page is specified through a query string that can be appended to the **pkmslogout** URL as follows:

```
https://www.example.com/pkmslogout?filename=[custom_logout_file|URL]
```

where:

- *custom_logout_file* is the file name (without directory information) of the custom logout response page. This file must be located in the same `nls/html/lang/codeset` directory (such as `/opt/pdwebpi/nls/html/C/utf-8`) that contains the default HTML response forms. For example: `/pkmslogout?filename=logout.html`.
- *URL* is a valid URL to which the client will be redirected. For example: `/pkmslogout?filename=http://w3.ibm.com`.

## pkmspasswd

You can use this command to change your logon password when using Basic Authentication (BA) or Forms authentication. This command is appropriate over HTTP or HTTPS.

For example:

```
https://www.ibm.com/pkmspasswd
```

The browser displays a change of password form defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
password-change-form-uri = /pkmspasswd.form
password-change-uri = /pkmspasswd
password-change-success = password_change_success.html
password-change-failure = password_change_failure.html
```

You can modify the `password_change_success.html` and `password_change_failure.html` files to suit your requirements.

## pkmshelp

You can use this command to access help pages. This command is appropriate over HTTP or HTTPS.

The name and location of help pages are defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
help-uri = /pkmshelp
help-page = help.html
```

You can modify the `help.html` file to suit your requirements.

## Password change issue with Active Directory on Windows

The following problem occurs for password changes when using Active Directory as the Security Access Manager user registry and the Active Directory server is running on Windows. Depending on certain Active Directory policy settings, old passwords can still be used to log in to Security Access Manager after a password change has occurred. By default, both the old and the new passwords continue to work for approximately one hour after the password change. After one hour, the old password stops working.

Windows introduced this behavior into Active Directory. See the Microsoft KB article 906305 for information on what occurs and for instructions on disabling the behavior if necessary.

`http://support.microsoft.com/?id=906305`

# Configuring Basic Authentication

Basic Authentication (BA) is a standard method for authenticating clients by using a user name and password.

Basic Authentication is defined by the HTTP protocol and is implemented over HTTP and HTTPS.

## Enabling Basic Authentication

By default, the plug-in is configured for BA authentication. The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of BA for authenticating requests. That is:

```
[common-modules]
authentication = BA
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for basic authentication exists; that is:

```
[modules]
BA = pdwpi-ba-module
```

By default the BA authentication mechanism is given a level of 1 in the **[authentication levels]** stanza of the configuration file. This setting relates to the priority of authentication mechanisms for incoming requests.

## Configuring the Basic Authentication mechanism

The **passwd-ldap** entry specifies the shared library that is used to handle user name 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`.

You can configure the user name and password authentication mechanism by entering the **passwd-ldap** entry with the platform-specific name of the shared library file in the `[authentication-mechanisms]` stanza of the `pdwebpi.conf` configuration file – as indicated in the following list:

- **Solaris**

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

- **Windows**

```
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```

## Setting the realm name

The realm name is displayed on the dialog that prompts the user for a user name and password. The realm name is assigned to the **basic-auth-realm** entry in the **[BA]** stanza of the `pdwebpi.conf` configuration file.

```
[BA]
basic-auth-realm = realm_name
```

## Manipulating BA headers

You can configure the plug-in to supply protected applications with original or modified client identity information by controlling the contents of the BA headers that are sent to the Web server. The existing header that is sent from the client can be:

- Stripped of all requests.
- Stripped of unauthenticated requests.
- Passed unchanged for all requests.

For clients that do not provide a BA header or for existing client header information passed to the Web server, the header information can be:

- Set to a fixed user name and password.
- Have a fixed password sent (with the user name passed as the name of the authenticated user).
- Be set using information from Security Access Manager GSO lockbox.

To manipulate the BA headers of incoming requests, the plug-in must be configured to allow post-authorization processing using Basic Authentication. To do this, add the entry**post-authzn** and set it to the value *BA* in the **[common-modules]** stanza of the `pdwebpi.conf` configuration file. That is:

```
[common-modules]
post-authzn = BA
```

The **strip-hdr** entry instructs the plug-in to either:

*Table 15.* **strip-hdr** *instructions to plug-in*

| Value | Result |
|---|---|
| *ignore* | Leave the header as is. The plug-in passes the original client BA header to the resource without interference. This constitutes a direct login to the resource that is transparent to the plug-in in situations when you want to bypass plug-in authentication.<br><br>Setting this option can potentially allow unauthenticated users to send BA headers to the Web server. You should only use this option if you are sure you need it and you understand the security implications.<br><br>When BA authentication is configured at the plug-in and the protected resource attempts to authenticate the client with its own BA challenge, the credentials of a user will not be accepted by the protected resource. Other authentication mechanisms such as Forms, configured at the plug-in, will pass the original client BA header to the resource without interference. |
| *always* | Always remove the Basic Authentication header information from any client requests before forwarding the requests to the Web server. In this case, the plug-in becomes the single security provider.<br><br>If you need to supply the Web server with some client information, you can combine this option with IV header authentication to put Security Access Manager client identity information into HTTP header fields.<br>**Note:** If the protected server sends a BA challenge with this option enabled, the client sees an authentication pop-up window but cannot log in because their response is always removed. |
| *unauth* | The BA header received from the client is removed from all requests except those from users that have been authenticated by the plug-in using Basic Authentication. This permits authenticated users to send authenticated BA headers to the web server but prevents an unauthenticated user from doing so. |

The **add-hdr** entry in the **[BA]** stanza of the configuration file allows you to supply client identity information in HTTP Basic Authentication (BA) headers. Supplying client identity information in HTTP BA headers using the **add-hdr** parameter, occurs after any processing by the **strip-hdr** entry functionality. The **add-hdr** entry can be set as: *none*, *gso*, or *supply*.

- Set as *none*, a BA header is not added to the request.
- Set as *gso*, a GSO BA header is added to the request — refer to "Using global single sign-on (GSO)" on page 154 for detailed information on configuring plug-in GSO functionality.
- Set as *supply*, a static password and user name are added to the BA header. These static password and user names are defined in the **supply-password** and **supply-username** entries in the **[BA]** stanza of the configuration file.
- If the **supply-username** entry is not set, the user name in the BA header is created using the Security Access Manager authenticated user name. In this case the plug-in protected resource requires authentication from a Security Access Manager identity.

  When the **add-hdr** entry is set to *supply* and the **supply-password** and **supply-username** entries are set, the specified user name and password are used

for all requests. The use of a common user name and 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 the plug-in to access resources, this solution does not present any security problems. However, it is important to physically secure resources from other possible means of access. Since this scenario has no password-level security, plug-in protected resources must implicitly trust the plug-in to verify the legitimacy of the client. The registry must also recognize the Security Access Manager identity in order to accept it.

If **supply-username** is not set and the user is unauthenticated then no BA header is added to the request.

### Specify UTF-8 encoding of BA headers

By default BA headers use UTF-8 encoding. This can be overridden by setting the **use-utf8** stanza entry to *false*.

```
[BA]
use-utf8 = false
```

For more information on plug-in support for UTF-8 encoding, see "Language support and character sets" on page 45.

## Configuring authentication by using forms

Security Access Manager supports authentication by using forms as an alternative to the standard basic authentication mechanism.

With forms authentication, the user is prompted to enter authentication information into a configurable form rather than the standard basic authentication logon prompt.

When you use forms-based logon, the browser does not cache the user name and password information as it does with basic authentication.

### Enabling forms authentication

Forms must be configured as an authentication module as well as a pre-authorization module. The following configuration entries must exist in the **[common-modules]** stanza of the `pdwebpi.conf` configuration file:

```
[common-modules]
authentication = forms
pre-authzn = forms
```

When using forms authentication, the plug-in must also be configured to use forms for pre-authorization processing.

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for forms authentication exists; that is:

```
[modules]
forms = pdwpi-forms-module
```

## Configuring the forms authentication mechanism

The **passwd-ldap** entry specifies the shared library used to handle user name 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`.

The user name and password authentication mechanism is set by entering the **passwd-ldap** entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file, as in the following:

**Solaris:**

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

**Windows:**

```
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```

## Customizing HTML response forms

Forms authentication requires you to use a custom logon form. By default, the sample `login.html` form is located on the directory, *install_path*`/nls/html/`*lang*`/`*charset*`.`

Where *lang* is taken from the NLS configuration. On US English systems, the *lang* directory is called **C** and *charset* is **utf-8**.

The **login-form** entry in the **[forms]** stanza of the configuration file defines the file name of the form presented to the user during log on. The path of the file should be relative to the translated pdwebpi HTML directory (for example *pdwebpi*`/nls/html/`*lang*`/`*charset*).

```
[forms]
login-form = login.html
```

The configured form may also be a valid redirect URI that can pass variable macro data. For example:

```
[forms]
login-form = auth/login.html?user=%USER%
```

**Note:** Removal of the **wpi_url** field from the `login.html` form, will cause POST data, submitted with the login form, to be available in the HTTP request as POST data variables. This includes the user name and password used to login to Security Access Manager. Also, removing the **wpi_url** field from a form disables all POST data caching functionality and macros such as %POST_URL% will no longer be supported.

## Customizing the forms login URI

It is possible to have multiple instances of the forms login module used within a single virtual host. In such instances it is necessary to change the URI POSTed to

when the login form is submitted for each separate instance of the forms login module. The **login-uri** entry in the **[forms]** stanza controls this URI. If changed from the default, the form specified by the **login-form** entry (see "Customizing HTML response forms" on page 65) must be updated to reflect the change.

## Creating a BA Header

Forms authentication provides the ability to create a BA header based on the user name and password provided in the login form. Creation of the header provides an easy single sign-on mechanism that can be used when the back-end application requires basic authentication, and the user name and password match that used by Security Access Manager.

BA header creation is handled by the forms post-authorization module. Add an entry for forms post-authorization processing to the **[common-modules]** stanza of the plug-in configuration file:

```
[common-modules]
post-authzn = forms
```

The **create-ba-hdr** entry within the **[forms]** stanza of the configuration file enables or disables the creation of BA headers, for example:

```
[forms]
create-ba-hdr = yes
```

By default forms authentication does not create the BA header - **create-ba-hdr** is set to *no*. Regardless of how the entry is set, a BA header is not created when the user is not successfully authenticated and a header is not created when the password of the user has expired.

**Note:** If another module after the forms module in the **post-authzn** list overwrites the BA header (or removes it) then this function does not work. It is advisable to have the forms module specified as the last in the **post-authzn** list.

If UTF-8 encoding is required for BA headers, set the following configuration entry:

```
[forms]
use-utf8 = true
```

For more information on plug-in support for UTF-8 encoding, see "Language support and character sets" on page 45.

# Configuring certificate authentication

Security Access Manager Plug-in for Web Servers supports secure communication with clients by using client-side digital certificates over SSL. With this authentication method, certificate information such as the Distinguished Name or DN, is mapped to a Security Access Manager identity.

## Mutual authentication using certificates

Authentication using digital certificates takes place in two stages:

- The Web server where the plug-in is located identifies itself to SSL clients with its server-side certificate.

- The Web server uses its database of Certificate Authority (CA) root certificates to validate clients accessing the server with client-side certificates. The following process takes place:

1. An SSL client requests a connection with a Web server through the plug-in.
2. In response, the Web server sends its public key using a signed server-side certificate. This certificate has been previously signed by a trusted third-party certificate authority (CA).
3. The client checks whether the certificate's issuer is one that it can trust and accept. The client's browser usually contains a list of root certificates from trusted CAs. If the signature on the Web server's certificate matches one of these root certificates, then the server can be trusted.
4. If there is no match for the signature, the browser informs its user that this certificate was issued by an unknown certificate authority. It is then the responsibility of the user to accept or reject the certificate.
5. If the signature matches an entry in the browser's root certificate database, session keys are securely negotiated between the client and the Web server.

   The end result of this process is a secure channel over which the client can authenticate (for example, using a user name and password). After successful authentication, the client and server can continue to communicate securely over this channel.
6. The client sends its public key certificate through the plug-in to the Web server.
7. The Web server attempts to match the signature on the client certificate to a known CA using the Web server's certificate store.
8. If there is no match for the signature, an SSL error code is generated and sent to the client.
9. If there is a match for the signature, then the client can be trusted. Authentication of the client takes place, resulting in a Security Access Manager identity.
10. Session keys are securely negotiated between the client and the Web server. The end result of this process is a secure and trusted communication channel between the mutually authenticated client and server.

## Enabling certificate authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using certificates, assign *cert* to the **authentication** configuration entry:

```
[common-modules]
authentication = cert
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and the associated shared library name. Ensure that the entry for certificate authentication exists:

```
[modules]
cert = pdwpi-certificate-module
```

**Note:** For installations on IHS, you must configure the Web server to request certificates from clients.

## Configuring the certificate authentication mechanism

The **cert-ssl** entry specifies the shared library for mapping certificate authentication information.

On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-sslauthn`.

On Windows, the file that provides the built-in mapping function is a DLL called `sslauthn`.

You can configure the certificate authentication mechanism by entering the **cert-ssl** entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

- Solaris

```
[authentication-mechanisms]
cert-ssl= libpdwpi-sslauthn.so
```

- Windows

```
[authentication-mechanisms]
cert-ssl = pdwpi-sslauthn.dll
```

**Note:** The `pdwpi-sslauthn` authentication mechanism requires the subject DN in the certificate of the user to exactly match the registry DN of the user. If you need to use a more complicated mapping, a customized authentication mechanism must be developed. See the *IBM Security Access Manager for Web: Web Security Developer Reference* for instructions on building authentication mechanism modules. These instructions also apply to the Plug-in for Web Servers.

# Configuring authentication using RSA SecurID tokens

Security Access Manager Plug-in for Web Servers supports authentication using an RSA® SecurID® token passcode supplied by the client. To permit communications with remote RSA servers, the RSA SecurID client must be installed and configured on the plug-in enabled server for token authentication to work. The minimum supported SecurID client version is 6.0.

**Note:** The RSA ACE/Agent client is supported on all platforms except for Linux on System Z.

RSA's ACE/Servers authenticate several different tokens, including software tokens and handheld microprocessor controlled devices.

SecurID software tokens are binary programs that run on either workstations, smart-cards, or as a plug-in to a Web browser. SecurID software tokens can also 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 Security Access Manager plug-in for Web servers by entering the passcode into a login form.

The most typical form of SecurID Token is the handheld 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 PIN and tokencode of the user. A

tokencode is a changing number displayed on the key fob that is regenerated at one minute intervals. A user enters the PIN and tokencode to authenticate to the ACE/Server.

Security Access Manager Plug-in for Web Servers 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 token card 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. 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

PIN modes are defined by the method of creation, and by rules that specify entries for password creation and device type.

The plug-in 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

The plug-in 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

The following process occurs for the authentication of tokens in new PIN mode:

1. A user requests a protected Web object requiring token authentication.
2. The plug-in prompts the user for their user name and passcode.
3. The user enters their user name and tokencode and submits the form. When the user has no PIN, either because the token card is new or the administrator

reset the PIN, the tokencode is the same as the passcode. When the user has a PIN, but the token card is in new PIN mode, the user enters the PIN plus the tokencode.

4. The plug-in 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 plug-in which displays an error page to the client and the client prompted to re-authenticate.
   b. If the token was not in new PIN mode, the user is authenticated and given access to the requested protected Web object.
   c. If the token is in new PIN mode, the ACE/Server returns the NEW_PIN error code to the plug-in.
6. The plug-in presents to the user the password expired form.
7. The user enters the tokencode or passcode and the new PIN and posts it to the plug-in.
8. The plug-in checks to see if a password strength server is deployed.
   a. If a password strength server is not deployed, the plug-in continues to step 9.
   b. If a password strength server is deployed, the plug-in checks the new PIN. If the PIN is valid, the plug-in continues to step 9. If the PIN is not valid, he plug-in returns to step 6.
9. The plug-in 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, the plug-in 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

The plug-in password strength functionality is specific to particular authentication mechanisms. This support enables security architects to develop different password strength policies for different plug-in 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.

## Enabling token authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using tokens, assign *token* to the **authentication** parameter.

When authentication using tokens is enabled then tokens must also be configured for pre-authorization processing. In the **[common-modules]** stanza of the configuration file, construct a **pre-authzn** entry and assign it the value *token*. The **[common-modules]** stanza should include the following two entries:

```
[common-modules]
pre-authzn = token
authentication = token
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and the associated shared library name. Ensure that the entry for token authentication exists:

```
[modules]
token = pdwpi-token-module
```

## Configuring the token authentication mechanism

The **token-cdas** entry specifies the shared library for mapping token passcode authentication information.

- On UNIX the file that provides the built-in mapping function is a shared library called `libxtokenauthn`.
- On Windows the file that provides the built-in mapping function is a DLL called `xtokenauthn`.

The token shared library is installed as part of the Security Access Manager Web Security Runtime (PDWebRTE) package. The location of this shared library is:

**UNIX** `/opt/pdwebrte/lib`

**Windows**

```
c:\Program Files\Tivoli\PDWebRTE\bin
```

By default, this built-in shared library is hardcoded to map SecureID token passcode data. You can customize this file to authenticate other types of special token data and, optionally, map this data to a Security Access Manager identity. Refer to the *IBM Security Access Manager for Web: Web Security Developer Reference* for API resources.

You can configure the token authentication mechanism by entering the **token-cdas** entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

For example:

**Solaris:**

```
[authentication-mechanisms]
token-cdas = libxtokenauthn.so
```

**Windows:**

```
[authentication-mechanisms]
token-cdas = xtokenauthn.dll
```

## Customizing token response pages

The **token-login-form** entry in the **[token-card]** stanza of the configuration file defines the file name of the form presented to the user client during a token logon. The path of the file should be relative to the translated plug-in HTML directory (for example, *pdwebpi*`/nls/html/`*lang*`/`*charset*), where *lang* is taken from the NLS configuration. On US English systems, the *lang* directory is called **C** and *charset* is **utf-8**.

The **next-token-form** entry in the **[token-card]** stanza defines the form displayed to the user client to request the next token. The client is requested to enter another token when the server cannot successfully authenticate the user from the first. Inability to authenticate the user can be caused by a number of reasons. Most

commonly, though, the error occurs because the client and server clocks are not synchronized. When authentication cannot succeed using the first token, the page specified in the **next-token-form** entry is displayed to prompt for the next token.

The **token-card** stanza has the following format:

```
[token-card]
token-login-form = tokenlogin.html
next-token-form = nexttoken.html
```

# Configuring SPNEGO authentication

SPNEGO authentication provides a Single Sign-on (SSO) capability for Windows user accounts when protected objects are accessed using Internet Explorer. With SPNEGO authentication the plug-in performs the server-side of the negotiation, and Internet Explorer performs the client side.

When a user requests access to a secure Web server, Internet Explorer employs the Windows login credentials of the user to participate in a negotiation with the Web server to prove the authenticity of the user. Once the server has confirmed the identity of the user, they are granted access if:

- The user is a member of the domain.
- SPNEGO has been enabled in the Authorization Server.
- The Authorization Server permits access.

Users accessing resources protected by plug-in SPNEGO authentication who are not members of the domain or are using browsers other than Internet Explorer must authenticate using another method, for example Basic Authentication or forms.

**Note:** SPNEGO authentication module functionality can only operate correctly if the Web server is configured to allow anonymous access. In the case of IIS, *Integrated Login* must be unchecked, and *Anonymous Access* checked. In the case of other Web servers, the standard configuration should be used.

## Platform and user registry support

The SPNEGO authentication mechanism is available on all supported Web server/platform/user-registry combinations.

When Active Directory is not the Security Access Manager user registry, users must be replicated between the Active Directory registry and the Security Access Manager user registry.

## Limitations

The following plug-in features are not supported with SPNEGO authentication:

- POP or session timer based reauthentication of SPNEGO authenticated clients.
- Password change using pkmspasswd for user registries other than Active Directory.
- Mapping of a user name through an external authentication mechanism.
- SPNEGO clients cannot log out of the plug-in. Clients must log out from the workstation. Clients that access plug-in 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.

# Windows desktop single sign-on configuration

You must complete the following configuration tasks to implement Windows desktop single sign-on using SPNEGO authentication with the plug-in.

**Note:** Not all steps are required on each platform.

1. "Configuring the embedded Kerberos client (UNIX only)"
2. "Configuring the plug-in server into the Active Directory domain" on page 75
3. "Mapping a Kerberos principal to the Active Directory user" on page 75
4. "Verifying authentication of the Web server principal (UNIX only)" on page 77
5. "Using the keytab file to verify plug-in authentication (UNIX only)" on page 78
6. "Enabling SPNEGO authentication within the plug-in" on page 78
7. "Enabling SPNEGO authentication within the Web server" on page 79

## Configuring the embedded Kerberos client (UNIX only)

You must configure the Kerberos client that is embedded in Security Access Manager.

### About this task

To complete this configuration, you must create or modify the `krb5.conf` Kerberos configuration file in the following directory:

`/opt/PolicyDirector/etc/krb5.conf`

### Procedure

1. Copy `/opt/PolicyDirector/etc/krb5.conf.template` to `/opt/PolicyDirector/etc/krb5.conf`.
2. Open the `/opt/PolicyDirector/etc/krb5.conf` file to edit the entries to match your environment.
3. In the `[libdefaults]` stanza, set the following entries:

   **default_realm**
   Specify the Active Directory domain name in uppercase.

   **default_tkt_enctypes**
   Specify the cipher suite names supported by the Active Directory Server that you are using to encrypt the WebSEAL AD Kerberos user key.

   **Note:** Windows Server 2008 R2 disables DES ciphers by default. Available ciphers for Windows 2008 R2 are: `rc4-hmac`, `aes128-cts`, or `aes256-cts`.

**default_tgs_enctypes**
Specify the cipher suite names supported by the Active Directory Server that you are using to encrypt the WebSEAL AD Kerberos user key.

**Note:** Windows Server 2008 R2 disables DES ciphers by default. Available ciphers for Windows 2008 R2 are: `rc4-hmac`, `aes128-cts`, or `aes256-cts`.

For example:

```
[libdefaults]
default_realm = EXAMPLE.COM
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
```

The **am_kinit** application uses these settings to verify the Security Access Manager Kerberos client configuration. See "Verifying authentication of the Web server principal (UNIX only)" on page 77.

**Note:** Set the encryption cipher suites when you create the key tab file. See "Mapping a Kerberos principal to the Active Directory user" on page 75.

4. In the `[realms]` stanza, create an entry for the default realm that was specified by the `default_realm` entry of the `[libdefaults]` stanza. Set the following entries:

   **kdc** Specify the fully qualified host name of the Active Directory Key Distribution Center (KDC), which is the host name of the Domain Controller. For example `mykdc.example.com:88`.

   **default_domain**
   Specify the local DNS domain of the server running the plug-in. This value is the domain that client browsers use to access the plug-in for Windows single sign-on. For example, `example.com`.

   An example for the default realm of `EXAMPLE.COM` is:

   ```
   [realms]
   EXAMPLE.COM = {
           kdc = mykdc.example.com:88
           default_domain = example.com
   }
   ```

   **Note:** The **am_kinit** application uses the KDC value to verify the Security Access Manager Kerberos client configuration. See "Verifying authentication of the Web server principal (UNIX only)" on page 77.

5. Configure entries in the `[domain_realm]` stanza to map local domains and domain names to the Active Directory Kerberos realm.

   For example:

   ```
   [domain_realm]
   www.examplefancy.com = EXAMPLE.COM
    .examplefancy.com   EXAMPLE.COM
   ```

6. Optional: To use SPNEGO as the authentication type for each junction level and junction, configure the `auth-challenge-type` entry in the `[server:`*`jct_id`*`]` stanza, where *jct_id* is the junction point for a standard junction (including the leading "/" character) or the virtual host label for a virtual host junction.

   For example:

   ```
   [server:/test-jct]
   auth-challenge-type = spnego
   ```

## Configuring the plug-in server into the Active Directory domain

To participate in a Kerberos exchange with a browser, the plug-in server needs an identity in the Active Directory Kerberos domain. The browser can then employ the Windows login credential of the user to access the plug-in enhanced server.

See the Microsoft documentation for instructions on how to add an identity for the plug-in server host into an Active Directory domain.

**Note:**

- On Windows, run the default plug-in server (first server instance) as a domain user, instead of a local system user when contacting the Active Directory domain controller.
- On UNIX, the user name must match the host name of the plug-in server host. Do not use the full domain name. For example, for the system `diamond.example.com`, create a user **diamond**.
  - Do not require the user to change password at next login.
  - Do not set the password to expire.
  - The account must not be set to use DES ciphers.
  - If you intend to use AES encryption for tickets and keys placed in the keytab, modify the appropriate user account properties to allow it. The properties are:
    - `This account supports Kerberos AES 128 bit encryption`
    - `This account supports Kerberos AES 256 bit encryption`

## Mapping a Kerberos principal to the Active Directory user

The Internet Explorer client request to the Active Directory domain controller requests access to a Kerberos principal of name:

`HTTP/`*`DNS_name_of_plug-in_server@Active_Directory_domain_name`*

Map this name to the Active Directory user that represents the plug-in enhanced server instance, as created in "Configuring the plug-in server into the Active Directory domain."

This mapping requires the Windows **ktpass** utility. The **ktpass** utility might not be loaded on the Windows system by default. You can obtain it from the Windows Support Tools package on the Windows CDs.

**Windows:**

Register the service principal name for the plug-in server. On the Active Directory domain controller, run the **ktpass** command. For example, when the plug-in host is `diamond.example.com`, and the Active Directory domain is `IBM.COM` the command is:

```
ktpass -princ HTTP/diamond.example.com@IBM.COM -mapuser diamond
```

**UNIX:**

Complete the following steps:

**About this task**

For UNIX systems, in addition to creating the user, you must create a keytab file for use when verifying Kerberos tickets.

**Procedure**

1. On the Active Directory domain controller, run the **ktpass** command, entering the following syntax on one line:

   ```
   ktpass -princ HTTP/DNS_name_of_WebPI_server@ACTIVE_DIRECTORY_DOMAIN_NAME
   {-pass your_password | +rndPass} -mapuser WebPI_server_instance
   -out full_path_to_keytab_file -mapOp set -crypto cipher -ptype KRB5_NT_PRINCIPAL
   ```

   where:

   *DNS_name_of_WebPI_server*
   Specifies the name of the Web Plug-in server.

   *ACTIVE_DIRECTORY_DOMAIN_NAME*
   Specifies the Active Directory domain name, in all upper case.

   The domain name must map to the Active Directory user.

   *your_password*
   Specifies a password to set when the `-pass` parameter is used. The password specified here resets the password for the Active Directory user. A highly secure password, such as a randomly generated password, is preferred. If you use a known password, retain it for use in a later step to test your Kerberos configuration. You need this password to test authentication from a UNIX computer to the Active Directory Key Distribution Center.

   *WebPI_server_instance*
   Specifies the Web Plug-in server instance user identity.

   *full_path_to_keytab_file*
   Specifies the fully qualified path to the keytab file. The location of the keytab file is arbitrary.

   *cipher* Specifies the cipher to use.

   **Note:** Windows Server 2008 R2 and Windows 7 clients disable DES ciphers by default. Windows XP clients do not support AES and might limit the ability to use AES.

   For Windows Server 2003, the available cipher is `RC4-HMAC-NT`.

   For Windows Server 2008 R2, the following cipher options are available:

   **RC4-HMAC-NT**
   Specifies the Kerberos client cipher **rc4-hmac**.

   **AES256-SHA1**
   Specifies the Kerberos client cipher **aes256-cts**.

   **AES128-SHA1**
   Specifies the Kerberos client cipher **aes128-cts**.

   **ALL** Generates keys in the keytab file for all ciphers. Use this value when you require a mix of AES and RC4 clients.

   If needed, use the **am_ktutil** command to remove unused DES encrypted keys. If you use an AES value for the cipher, you must update the Active Directory user account settings appropriately. Use one or more of the following two properties:

   - `This account supports Kerberos AES 128 bit encryption`
   - `This account supports Kerberos AES 256 bit encryption`

2. Transfer the keytab files to the UNIX system using a secure transfer method. For example, a suggested location is:

   ```
   /var/pdwebpi/...
   ```

3. For best security practice, delete the keytab files from the Windows system.
4. Optional: If you are configuring SPNEGO for multiple virtual hosts, create a separate keytab file for each virtual host. Repeat the **ktpass** command for each principal in Active Directory. You must combine all keytabs into a single file using the **am_ktutil** program. Available commands include the following:

   **rkt** Reads in the keytabs.

   **list -e** Verifies that the principal names and ciphers are correct.

   **wkt** Writes out a new, combined keytab file.

   For example:

   ```
   # /usr/krb5/sbin/am_ktutil  ktutil:rkt eng_HTTP.keytab
   ktutil:  rkt www_HTTP.keytab
   ktutil:  rkt sales_HTTP.keytab
   ktutil:  list -e
   slot   KVNO   Principal
   ------ ------ ------------------------------------
   1      1 HTTP/eng.example.com@EXAMPLE.COM (arcfour-hmac)
   2      1 HTTP/sales.example.com@EXAMPLE.COM (arcfour-hmac)
   3      1 HTTP/www.example.com@EXAMPLE.COM  (arcfour-hmac)
   ktutil:  wkt spnego.keytab
   ktutil:  quit
   ```

   Use the combined keytab file as the keytab file for the plug-in server.

   **Note:**

   - If the –cipher ALL option was used when generating the keytab file, then you can use the **am_ktutil** program to remove redundant keys from the keytab.
   - Kerberos can have alternate names for a particular cipher. For example arcfour-hmac is an alias for **rc4-hmac** (the Windows RC4-HMAC-NT cipher).

5. On the UNIX system, assign ownership of the file to pdwebpi, and restrict permissions on the keytab file so that only the owner can access it. For example:

   ```
   # chown pdwebpi keytab_file
   # chgrp pdwebpi keytab_file
   # chmod 600 keytab_file
   ```

6. Repeat these steps for each plug-in instance on a UNIX server.

## Verifying authentication of the Web server principal (UNIX only)

Skip this task if you used the +rndPass option to generate the keytab files.

Use the **am_kinit** program to verify that the Kerberos principal for the plug-in enhanced server can authenticate. Use the password specified when you ran **ktpass** in the previous task:

```
# /usr/krb5/bin/am_kinit diamond@IBM.COM
 Password for diamond@IBM.COM: server_password
# am_klist
```

An output from **am_klist** showing the credentials for diamond@IBM.COM displays.

## Using the keytab file to verify plug-in authentication (UNIX only)

Verify that the plug-in can authenticate using the keytab file created in "Mapping a Kerberos principal to the Active Directory user" on page 75. Enter the following **am_kinit** command as one continuous line:

```
# am_kinit -k -t /var/pdwebpi/keytab-diamond/diamond_HTTP.keytab
HTTP/diamond.example.com@IBM.COM
# am_klist
```

Output from **am_klist** showing the credentials for HTTP/diamond.example.com@IBM.COM displays.

## Enabling SPNEGO authentication within the plug-in

Modify the Web Plug-in configuration file to enable SPNEGO.

### Procedure

1. In the [common-modules] stanza of the plug-in configuration file, pdwebpi.conf, assign the value, spnego, to the authentication entries.

   For example,

   ```
   [common-modules]
   authentication = spnego
   ```

2. In the [authentication-mechanisms] stanza, set the value of the kerberosv5 entry to the Security Translation Layer Interface (stli) shared library. Use the absolute path to the library. For example:

   **AIX:** `kerberosv5 = /opt/PolicyDirector/lib/libstliauthn.a`

   **Linux or Solaris**

   `kerberosv5 = /opt/PolicyDirector/lib/libstliauthn.so`

   **Windows:**

   `kerberosv5 = C:\PROGRA~1\Tivoli\POLICY~1\bin\stliauthn.dll`

3. In the [spnego] stanza, complete the following tasks:

   a. Set the spnego-krb-service-name entry to one of the following values:

      - HTTP
      - HTTP@ *fully_qualified_host_domain_name*

   b. On AIX, Linux, or Solaris systems, set the value of the spnego-krb5-keytab-file entry to the keytab file. Use the full path name of the keytab file that is to be used by the plug-in. On Windows, this option is ignored.

   c. If you want user IDs to include the domain name as a means of distinguishing users from multiple domains, set use-domain-qualified-name to yes.

      For example, if users with matching names could exist in separate domains, such as "joe@one.example.com" and "joe@two.example.com," set the use-domain-qualified-name entry to yes. The default value is no.

      **Note:** When use-domain-qualified-name is set to no, the Web Plug-in builds Security Access Manager credentials from the user ID short name only. For example, if SPNEGO authentication returns a user name of "name@one.example.com," the name is truncated to "name."

### Enabling SPNEGO authentication within the Web server

To enable SPNEGO for IIS, ensure the access policy of the Web server, which is set in the **Directory Security** tab, is set to `anonymous`. For other Web servers, the default configuration is acceptable. You must configure the Internet Explorer client to enable SPNEGO.

#### Procedure

1. If the plug-in is installed on UNIX, configure the *Local Intranet* zone to include the name of the UNIX server:
   a. Select **Tools** > **Internet Options**.
   b. From the **Security** tab select either **Local Intranet** > **Sites** > **Advanced** > **Trusted Sites** > **Sites**.
   c. Enter the UNIX server on which the plug-in is running.
2. Configure the integrated login behavior:
   a. Select **Tools** > **Internet Options**.
   b. From the **Security** tab click **Custom Level**.
   c. Scroll down to the *Logon* section in the *Security Settings* dialog, and select either *Automatic...* or *Prompt...* depending on the functionality you require.

      **Note:** If the **Trusted Sites** option was selected when configuring the *Local Intranet* zone, then the user will never be prompted to enter user name and password information.
3. If the client is version 6 of Internet Explorer, you must configure the Integrated Windows login. Complete the following steps:
   a. Select **Tools** > **Internet Options**.
   b. From the **Advanced** tab, check *Enable Integrated Windows Login*.
   c. Restart the browser for the change to take effect.

## Troubleshooting for SPNEGO, Windows desktop single sign-on, and Kerberos

See the section describing solutions to common Web security SPNEGO problems in the *IBM Security Access Manager for Web: Troubleshooting Guide*.

# Configuring NTLM authentication (IIS platforms only)

You might need to configure NT LAN Manager (NTLM) authentication.

### About this task

Previous versions of the Windows platform provided a rudimentary Single Sign-on (SSO) mechanism known as NT LAN Manager (NTLM) authentication. This method of authentication is based on hashing algorithms providing a similar level of security and operation as that of Basic Authentication. The plug-in supports NTLM authentication to facilitate backwards compatibility between modern Windows platforms (XP, 2000) and older systems such as Windows NT. The plug-in supports NTLM on the Windows IIS platform only. UNIX platforms are not supported.

### Procedure

1. Assign the value, *ntlm*, to the **authentication** entry in the **[common-modules]** stanza of the plug-in configuration file, `pdwebpi.conf`.

```
[common-modules]
authentication = ntlm
```

2. Ensure the access policy of the IIS Web server is set to anonymous.
3. To configure Internet Explorer for participation in NTLM (and SPNEGO) exchanges:
   a. Configure the integrated login behavior:
      1) Select **Tools** > **Internet Options**.
      2) From the **Security** tab click **Custom Level**.
      3) Scroll down to the *Logon* section in the *Security Settings* dialog, and select either *Automatic...* or *Prompt...* depending on the functionality you require.

         **Note:** If the **Trusted Sites** option was selected in step 1 above, then the user will never be prompted to enter user name and password information.
   b. If the client is version 6 of Internet Explorer then you need to configure the Integrated Windows login. To do this:
      1) Select **Tools** > **Internet Options**
      2) From the **Advanced** tab, check *Enable Integrated Windows Login*.
      3) Restart the browser for the change to take effect.

   The **use-pre-windows-2000-logon-name** entry in the **[ntlm]** stanza of the plug-in configuration file can be used to configure either the Windows 2000 or the pre-Windows 2000 user name formats.

   By default, the **ntlm** module uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the *user name* portion of the *user name*@domain.com logon name.

   The **use-pre-windows-2000-logon-name** entry allows the pre-Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the *user name* portion of the DOMAIN*user name* logon name.

   This entry is ignored if Security Access Manager uses Active Directory as its user registry. With Active Directory, the Security Access Manager user name is always the *user name* portion of the *user name*@domain.com logon name.

## Configuring Web server authentication (IIS platforms only)

You might need to configure Web server authentication for IIS platforms.

### About this task

Some Web servers provide the ability to perform authentication natively. One example of this is the capability of IIS to perform Integrated Windows Login (SPNEGO, NTLM or BA). The plug-in can be configured to utilize this native Web server authentication trusting that the Web server has performed adequately secure authentication checks. Currently Web server authentication for the plug-in is only supported on IIS.

### Procedure

1. Assign the value, *web_svr_authn*, to the **authentication** entry in the **[common-modules]** stanza of the plug-in configuration file,`pdwebpi.conf`.

   ```
   [common-modules]
   authentication = web_svr_authn
   ```

2. Configure Internet Explorer for participation in NTLM (and SPNEGO) exchanges:

a. Configure the integrated login behavior:
   1) Select **Tools** > **Internet Options**.
   2) From the **Security** tab click **Custom Level**.
   3) Scroll down to the *Logon* section in the *Security Settings* dialog, and select either *Automatic...* or *Prompt...* depending on the functionality you require.

      **Note:** If the **Trusted Sites** option was selected in step 1 above, then the user will never be prompted to enter user name and password information.

b. If the client is version 6 of Internet Explorer, configure the Integrated Windows login. To do this:
   1) Select **Tools** > **Internet Options**.
   2) From the **Advanced** tab, check *Enable Integrated Windows Login*.
   3) Restart the computer for the change to take effect.

   The **web-server-authn** authentication module can be configured to use either the Windows 2000 or the pre-Windows 2000 user name formats by setting the **use-pre-windows-2000-logon-name** entry in the plug-in configuration file under the **[web-server-authn]** stanza.

   By default, the **web-server-authn** module uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the *user name* portion of the *user name*@domain.com logon name.

   The **use-pre-windows-2000-logon-name** entry allows the pre-Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the *user name* portion of the DOMAIN*user name* logon name.

   This entry is ignored if Security Access Manager uses Active Directory as its user registry. With Active Directory, the Security Access Manager user name is always the *user name* portion of the *user name*@domain.com logon name.

## Configuring failover authentication

Use of the IBM Security Access Manager session management server (SMS) is the preferred solution for implementing a failover solution for plug-in enabled servers as the SMS provides more secure failover functionality. The failover authentication configuration described in this section is for customers not wanting to use the SMS.

This section contains the following topics:



### Failover authentication concepts

Failover authentication provides the ability to preserve an authenticated session across replicated plug-in instances when the active plug-in instance becomes unavailable. A failover cookie preserves the user information and recreates the user credential when the original session becomes unavailable. This saves the client from needing to re-authenticate manually. The plug-ins on replicated servers share a common key that decrypts the credential information held in the cookie and establishes the new session.

The failover facility is typically used by clients connecting to a replicated front-end Web server through a load-balancing mechanism. It can be used over either HTTP or HTTPS, or both.

**Note:** The plug-in SMS functionality discussed in "The Session Management Server (SMS)" on page 116 also supports a failover mechanism for replicated Security Access Manager environments.

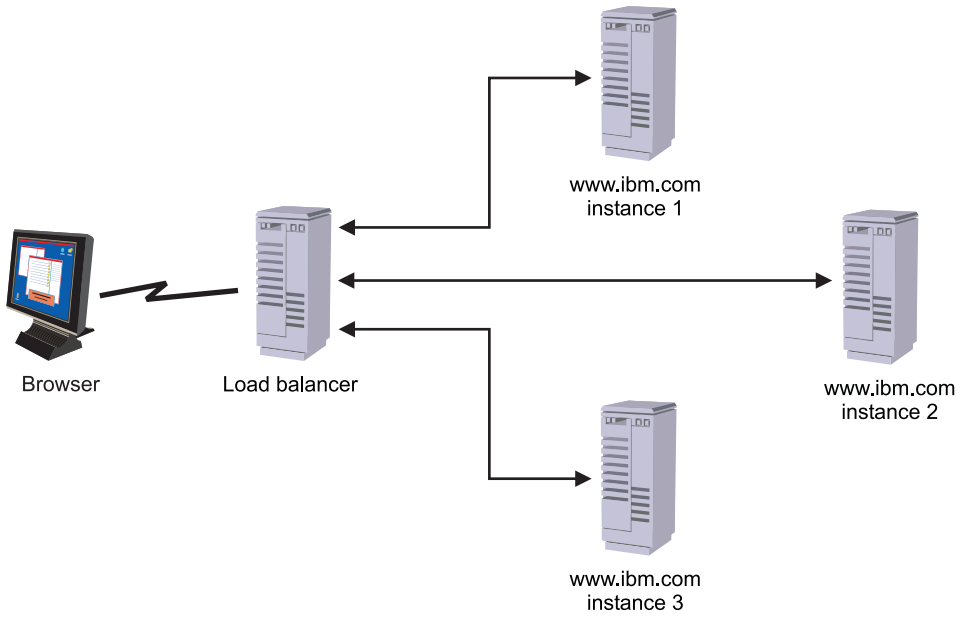


*Figure 4. Typical server architecture for failover cookies.*

In the above diagram, three identical instances of the same Web server are located behind a load balancing server that directs requests to one of the three servers depending on load and availability. Assume that each instance of `www.ibm.com` is configured to authenticate client access using failover cookies and also configured to use failover cookies for post-authorization processing.

A client accesses `www.ibm.com` and is directed to instance 1 of the server and authenticates successfully. The client's credential is encrypted and stored in a domain wide cookie that is stored at the client browser. If during the session instance 1 fails or demand becomes too great, the client is directed to either instance 2 or instance 3. The failover cookie stored in the client's browser is used to automatically re-authenticate the client. The original session start time is kept with the cookie so that the integrity of the session lifetime remains valid when an automatic redirection to a failover server takes place.

## Failover authentication library

A built-in failover authentication shared library is supplied for each of the supported authentication methods. Each failover shared library mimics the shared library for the corresponding authentication method and recovers any extended attributes that were originally placed in the credential of the user.

Failover authentication is available 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 sign-on (CDSSO)
- Kerberos authentication (SPNEGO)

The plug-in 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. Alternatively, you can supply a custom library that provides specific authentication capabilities required by your environment.

For example, when both forms authentication and failover authentication are enabled in a replicated plug-in environment, two separate libraries must be configured in the plug-in 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 the plug-in's built-in forms authentication library. The `failover-password` stanza entry specifies the plug-in's built-in failover authentication library.

## Addition of data to a failover cookie

The plug-in automatically adds specific data from the user session to each failover cookie. The plug-in can be configured to add additional information from the client data maintained in the credential cache. Also, the plug-in can be configured to add specific user-defined data. For example, user attributes obtained by a custom cross-domain authentication service can be added to the cookie.

By default the plug-in adds the following data to each cookie:

- **User name**

  This name corresponds to the name used to identify the user in the registry

  **Note:** When an authenticated user has used the plug-in's 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 the plug-in.
- **Cookie creation time**

  The system time when the cookie was created.

The plug-in 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 plug-in's step-up authentication level (also an integer value) that is assigned on the local plug-in server to the authentication method. Step-up authentication enables a user to authenticate to a different authenticate method without having to log out.

The plug-in defines additional user data that can be added to the cookie attribute list:

- **Session lifetime timestamp**

  When a user authenticates, the plug-in 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 session data of a user can remain in the session cache. When the current system time exceeds the timestamp value, the plug-in invalidates the entry of the user in the session cache (including the user credentials).

  The plug-in 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, plug-in 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 plug-in enhanced servers. If clock skew becomes great, sessions will expire at unintended times.

- **Session inactivity timestamp**

  The plug-in also tracks the amount of time that an entry of a user in the session cache of the plug-in has been inactive. When a user session is inactive for a period of time longer than the value set for session inactivity, the plug-in invalidates the session of the user.

  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 session cache of the plug-in. The system inactivity timeout maintained for the cache is calculated by combining two values:

  – The current system time
  – The maximum number of seconds that a session of a user 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 plug-in 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 the plug-in 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 plug-in 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).

## Extraction of data from a failover cookie

When a failover authentication event occurs, the plug-in receives a failover authentication cookie and by default extracts the following data from each cookie:

- User name
- Authentication method
- Cookie creation time

The plug-in first determines if the cookie is valid by subtracting the cookie creation time from the system time, and comparing this value against the plug-in 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, the plug-in uses the user name and authentication method to authenticate the user and build a user credential.

The plug-in next checks configuration settings to determine if additional cookie data should be extracted and evaluated. Note that the plug-in does not by default extract any other attributes from the failover authentication cookie. Each additional attribute to be extracted must be specified in the plug-in configuration file. Wildcard pattern matching can be used to obtain groups of attributes.

The plug-in can be configured to extract the following defined attributes:

- Authentication level

  When this value is extracted, the plug-in uses it to ensure that the user is authenticated with the authentication method necessary to maintain the specified authentication level.

  Note that the plug-in can obtain authentication levels from several different places:

  – Failover cookie
  – Failover authentication library
  – Cross-domain authentication service
  – Entitlement service

  The authentication level extracted from the failover cookie takes precedence over levels obtained from the other places.
- Session lifetime timestamp

  The plug-in 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, the plug-in 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

  The plug-in 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, the plug-in 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 plug-in 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. The plug-in adds the attributes to the user credential.

Attributes that are not specified in the plug-in 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.

### Domain-wide failover authentication

The plug-in supports an optional configuration that enables failover authentication cookies to be marked as available for use during failover authentication to any and all other plug-in enhanced servers or WebSEAL servers in the Security Access Manager domain.

When a client session goes through a failover authentication event to a replicated plug-in enhanced server, the client continues to access the same set of protected resources. When a client session goes through a failover authentication event to a plug-in enhanced server that is not replicated, it is possible that a different set of resources will be available to the client.

Such partitioning of resources within the Security Access Manager domain can be done for performance reasons and for administrative purposes. This is common in large Security Access Manager deployments.

Domain-wide failover authentication can be used to redirect a client to another server at a time when the client's requests have led it to request a resource that is not available through the local server. In this case, the client (browser) is redirected to another plug-in enhanced server.

The receiving plug-in can be configured to look for failover authentication cookies. The plug-in attempts to authenticate the client and recognizes the failover authentication cookie. By using the cookie, the plug-in 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.

## Failover authentication configuration

You can configure failover authentication.

### About this task

If you are not familiar with failover authentication concepts, review "Failover authentication concepts" on page 81.

### Procedure

1. Stop the plug-in server.
2. To enable failover authentication, complete each of the following tasks:
   a. "Enabling authentication using failover cookies" on page 87

b. “Specify the failover authentication library” on page 88
c. “Create an encryption key for cookie data” on page 88
d. “Specify the cookie lifetime” on page 89

3. Optionally, you can configure the plug-in 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 90
   b. “Add the session activity timestamp” on page 91
   c. “Add an interval for updating the activity timestamp” on page 91
4. Optionally, you can configure the plug-in to add extended attributes to the failover cookie. See “Add extended attributes” on page 92.
5. When you have configured the plug-in to add attributes to the failover cookie, you must configure the plug-in to extract the attributes when reading the cookie: See “Specify attributes for extraction” on page 92.
6. Optionally, you can enable failover authentication cookies for use on any plug-in enhanced server within the domain. If this is appropriate for your deployment, see “Enable domain-wide failover cookies” on page 93.
7. If you need to maintain backwards compatibility with failover authentication cookies generated by plug-in enhanced servers from versions prior to Version 6.0, complete the following instructions:
   a. “Specify UTF-8 encoding on cookie strings” on page 89
   b. “Require validation of a lifetime timestamp” on page 93
   c. “Require validation of an activity timestamp” on page 94
8. After completing all the instructions applicable to your deployment, restart the server.

## Enabling authentication using failover cookies

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. Failover cookies can be configured to perform authentication and post-authorization tasks.

Plug-ins configured for post-authorization processing using failover cookies, encrypt and store a credential as a failover cookie in the transaction response.

Plug-ins, configured to use failover cookies for performing authentication, re-authenticate clients using the encrypted credential from a failover cookie found in the transaction request.

To enable authentication and post-authorization using failover cookies, assign the reference 'failover' to the **authentication** and **post-authzn** parameters:

```
[common-modules]
authentication = failover
post-authzn = failover
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for failover authentication exists:

```
[modules]
failover = pdwpi-failovercookie-module
```

## Specify the failover authentication library

Edit the plug-in 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 plug-in 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
```

The plug-in supplies one standard failover shared library that functions for all the above authentication methods. Refer to the following table for the library names:

*Table 16. Failover authentication library file names*

| Operating system | Library file name |
|---|---|
| Solaris | `libfailoverauthn.so` |
| Linux | `libfailoverauthn.so` |
| AIX | `libfailoverauthn.a` |
| Windows | `failoverauthn.dll` |

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 library that implements a customized version of failover authentication for one or more authentication methods, insert the name of the custom external authentication mechanism as the value for the configuration file keyword. For example, if you developed a custom authentication mechanism 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 create the keys to secure the cookie data.

### About this task

This utility generates a symmetric key that encrypts and decrypts the data in the cookie.

**Attention:** If you do not configure the plug-in to encrypt failover authentication cookies, and you have enabled failover authentication, the plug-in will generate an error and refuse to start. Failover authentication cookies must be encrypted.

### Procedure

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/pdwebrte/bin/cdsso_key_gen` *`absolute_pathname_for_keyfile`*

   **Windows:**

   ```
   MSDOS> C:\Program Files\Tivoli\PDWebrte\bin\cdsso_key_gen
   absolute_pathname_for_keyfile
   ```

   You can give the key file any appropriate name, such as `/opt/pdwebrte/lib/wpi.key`.
2. Edit the plug-in 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 plug-in configuration file to supply the correct path name to `failover-cookies-keyfile` in the **[failover]** stanza.

## Specify the cookie lifetime

Edit the plug-in configuration file. Specify the valid lifetime for the failover cookie.

```
[failover]
failover-cookie-lifetime = 30
```

The default lifetime is 30 minutes.

## Specify UTF-8 encoding on cookie strings

Strings within the failover cookie use UTF-8 encoding by default. This can be changed using the following configuration entry:

```
[failover]
use-utf8 = true
```

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 plug-in enhanced server is using. By default, the plug-in supports UTF-8 encoding. When all servers in the plug-in deployment use UTF-8 encoding, leave this value at the default setting of `true`.

#### Backwards compatibility

Plug-in installations prior to Version 6.0 did not use UTF-8 encoding. Thus, cookies created by these servers do not have UTF-8 encoding on their strings. When a plug-in instance is operating with a plug-in from versions prior to Version 6.0, the plug-in should not use UTF-8 encoding.

For backwards compatibility, set `use-utf8` to `false`.

For more information on plug-in support for UTF-8 encoding, see “Language support and character sets” on page 45.

## Specify the authentication level

The plug-in provides a number of different ways to specify an authentication level. For failover cookies, there are two methods that can be used. One method sets the authentication level in the failover cookie. The other method sets the level when calling the failover authentication library.

When both methods are used, the authentication level in the failover cookie takes precedence over the level set when calling the library.

If neither of the methods are configured, the authentication level is set to the authentication level associated with the failover method by the **[authentication-levels]** stanza.

The two methods are:

- Specify authentication level in the failover authentication cookie.

  Add the authentication level to the plug-in configuration file. You must use the stanza entry keyword AUTHENTICATON_LEVEL:

  ```
  [failover-add-attributes] or [failover-add-attributes:virtual-host]
  AUTHENTICATION_LEVEL = add
  ```

  The actual value for AUTHENTICATION_LEVEL is an integer that the plug-in tracks internally. You do not need to specify the integer in this stanza.

  To retain the authentication level optionally loaded into the cookie by the originator, the value must also be configured to be preserved at the receiving end using the following entry:

  ```
  [failover-restore-attributes] or [failover-restore-attributes:virtual-host]
  AUTHENTICATION_LEVEL = preserve
  ```

- Specify authentication level when calling the failover authentication library.

  When configuring either the built-in plug-in failover authentication library or an external authentication mechanism library, you can optionally specify an authentication level to assign to the authenticated user. The authentication level is an integer that maps to a specific authentication method, as part of the plug-in's authentication strength feature.

  Add a command line argument to the configuration file entry for the appropriate failover authentication library. The syntax is:

  ```
  [authentication-mechanisms]
  failover_authentication_method = failover_authentication_libary& -l level_number
  ```

  The *level_number* must correspond to a valid integer, as specified in the [authentication-levels] stanza in the plug-in configuration file.

  For example, to activate failover authentication for forms authentication on a Solaris systems, and to assign to the user an authentication level of "3", create the following configuration file entry:

  ```
  [authentication-mechanisms]
  failover-password = libfailoverauthn.so& -l 3
  ```

## Add the session lifetime timestamp

The plug-in calculates the session lifetime timestamp by combining the following values:

- The current system time.
- The maximum lifetime in seconds that an entry is allowed to exist in the plug-in credential cache.

This maximum lifetime in seconds is specified in the plug-in configuration file `[session]` stanza:

```
[sessions]
timeout = 3600
```

To add this value to the failover authentication cookie, add the following entry to the plug-in 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

The plug-in 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 **[sessions]** stanza in the 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 plug-in configuration file:

  ```
  [failover]
  failover-update-cookie = -1
  ```

  The default value is `-1` seconds. A negative integer means the failover cookie is only updated when authentication occurs or when the credential is refreshed. 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 plug-in configuration file:

```
[failover-add-attributes]
session-activity-timestamp = add
```

**Note:** This attribute cannot be set by wildcard matching. The exact entry `session-activity-timestamp` must be entered.

## 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 = -1
```

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 83.

## Add extended attributes

The plug-in 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 plug-in configuration file. The syntax is:

```
[failover-add-attributes]
attribute_pattern = add
```

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:

```
[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 attributes for extraction

The plug-in 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 plug-in configuration file. The syntax is:

```
[failover-restore-attributes]
attribute_pattern ={preserve|refresh}
```

The value `preserve` tells the plug-in to extract the attribute and add it to the credential. Values set by this method overrule attributes of the same name that may have been set when the authentication mechanism created the new credential.

The value `refresh` tells the plug-in to conditionally extract the attribute and add it to the credential only if an attribute of the same name was not added when the authentication mechanism created the new credential.

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 plug-in within the same domain as the plug-in 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 = false
```

To enable this feature, set `enable-failover-cookie-for-domain` to *true*:

```
[failover]
enable-failover-cookie-for-domain = true
```

For information on the effects of enabling this stanza entry, see "Domain-wide failover authentication" on page 86.

## Require validation of a lifetime timestamp

The plug-in 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 = false
```

This stanza entry is used primarily for backwards compatibility.

**Attention:** For backwards compatibility with failover cookies created by plug-ins prior to Version 6.0, set this entry to *false*. Failover authentication cookies created by plug-ins prior to Version 6.0 do not contain this timestamp.

- When this value is *false*, and the session lifetime timestamp is missing from the failover cookie, the receiving server will view the cookie as *valid*.
- When this value is *true*, 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 *false* or *true*, 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

The plug-in 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 = false
```

This stanza entry is used primarily for backwards compatibility.

**Attention:** For backwards compatibility with failover cookies created by plug-ins prior to Version 6.0, set this entry to *false*. Failover authentication cookies created by plug-ins prior to Version 6.0 do not contain this timestamp.

- When this value is *false*, and the session activity timestamp is missing from the failover cookie, the receiving server will view the cookie as *valid*.
- When this value is *true*, 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 *false* or *true*, 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.

## Configuring IV header authentication

Security Access Manager supports authentication using internally generated header information supplied by a compatible client or a proxy agent. For historic reasons these are called IV (IntraVerse) headers. When the plug-in enhanced Web server receives requests from a trusted application such as WebSEAL or a multiplexing proxy agent, IV headers may be inserted into the requests relayed to the plug-in proxy server.

IV headers contain information that identify the originating client rather than the relaying server. The information in the headers is used to construct an originating client credential for authorization purposes. Similarly, if the plug-in enhanced Web server relays requests to another Security Access Manager server that recognizes IV headers, the plug-in proxy can insert IV headers to identify the originating client.

The plug-in can be configured to use IV headers for post-authorization processing or for authenticating requests. Configured for post-authorization processing, the plug-in, after a successful authentication, modifies a transaction request by inserting the client's true identity as IV headers. These headers may then be forwarded on to another server by the originating Web server.

If the plug-in is configured to use IV Headers to perform client authentication, the plug-in creates a client credential using the identity extracted from an IV header found in a transaction request. Because it is easy for clients to fake IV headers, such a credential is created only if the request is received via a trusted multiplexing proxy agent (MPA). See, "Supporting Multiplexing Proxy Agents (MPA)" on page 110.

For authentication, IV headers can be configured to accept one, some, or all of iv-user, iv-user-l, iv-creds, or iv-remote-address headers in the request as proof of authentication when received through a proxy. The iv-remote-address header is used to record the real remote address of the user.

Configured for post-authorization processing, IV headers are inserted with one, some or all of the iv-user, iv-user-l, iv-creds, iv-groups, and/or iv-remote-address, HTTP headers into the request.

*Table 17. IV header field descriptions*

| IV Header Field | Description |
|---|---|
| **iv-user** | The short name of the Access Manger user. Defaults to unauthenticated if the client is unauthenticated (unknown). |
| **iv-user-l** | The full domain name of the user (long form). For example, the registry distinguished name. |
| **iv-groups** | A list of the groups to which the user belongs. |
| **iv-creds** | Encoded opaque data structure representing the user's Security Access Manager credential. |
| **iv-remote-address** | The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT). |

**Note:** Access Manager only trusts headers received from trusted frontends. A frontend is considered trusted if it is recognized as a Multiplexing Proxy Agent (MPA). For details on configuring the plug-in for supporting MPA's refer to "Supporting Multiplexing Proxy Agents (MPA)" on page 110.

## Enabling authentication using IV headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using IV headers, assign the reference **iv-headers** to the **authentication** parameter:

```
[common-modules]
authentication = iv-headers
```

To enable IV headers for post-authorization processing, assign the value *iv-headers* to the **post-authzn** entry in the **[common-modules]** stanza of the `pdwebpi.conf` configuration file:

```
[common-modules]
post-authzn = iv-headers
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for IV header authentication exists:

```
[modules]
iv-headers = pdwpi-iv-headers-module
```

## Configuring IV header parameters

IV header authentication entries are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **accept** entry specifies the types of IV headers that are accepted for performing IV header authentication. By default the plug-in accepts all types of IV headers. The valid options are, all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
accept = iv-creds,iv-user
```

The **generate** entry specifies the type of IV headers to generate when forwarding proxied requests. By default the plug-in generates all types of IV headers when forwarding proxied requests. The valid options are: all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

## Specify UTF-8 encoding of IV headers

Edit the plug-in configuration file. Specify whether or not the plug-in should use UTF-8 encoding for IV headers.

```
[iv-headers]
use-utf8 = true
```

The default value is `true`.

For more information on plug-in support for UTF-8 encoding, see "Language support and character sets" on page 45.

## Configuring the IV header authentication mechanism for iv-remote-address

When using **iv-remote-address** in the IV Header you will need to specify the shared library for mapping HTTP authentication header information. The **http-request** authentication mechanism specifies the shared library for mapping HTTP authentication header information.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libhttpauth`.
- On Windows, the file that provides the built-in mapping function is a DLL called `httpauthn.dll`.

You can configure the HTTP header authentication mechanism by entering the http-request entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file, that is:

**Solaris:**

```
[authentication-mechanisms]
http-request = libhttpauth.so
```

**Windows:**

```
[authentication-mechanisms]
http-request = httpauthn.dll
```

# Configuring HTTP header authentication

Security Access Manager supports authentication through custom HTTP header information supplied by the client or a proxy agent.

This mechanism requires a mapping function (a shared library) that maps the trusted (pre-authenticated) header data to a Security Access Manager identity. The plug-in can take this identity and create a credential for the user.

The plug-in assumes that custom HTTP header data has been previously authenticated by a proxy agent. For this reason the module only works if the plug-in is located behind an authenticated Web proxy agent, and the **mpa-enabled** entry within the **[pdweb-plugins]** stanza is set to *true*.

By default, this shared library is built to map data from Entrust Proxy headers.

## Enabling authentication using HTTP headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using HTTP headers, assign the reference *http-hdr* to the **authentication** parameter; that is:

```
[common-modules]
authentication = http-hdr
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for HTTP header authentication exists:

```
[modules]
http-hdr = pdwpi-httphdr-module
```

## Specifying header types

You must specify all supported HTTP header types in the **[http-hdr]** stanza of the `pdwebpi.conf` configuration file.

```
[http-hdr]
header = header_type
```

A standard configuration of HTTP headers permits only one header to be specified, for example:

```
[modules]
http-hdr = pdwpi-httphdr-module
```

To specify multiple HTTP headers, multiple instances of the HTTP header module must be configured.

For example:

```
[modules]
entrust-client-header = pdwpi-httphdr-module
some-other-header= pdwpi-httphdr-module

[entrust-client-header]
```

```
header = entrust-client

[some-other-header]
header = some-other
```

## Configuring the HTTP header authentication mechanism

The **http-request** entry specifies the shared library for mapping HTTP authentication header information.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-http-cdas`.
- On Windows, the file that provides the built-in mapping function is a DLL called `pdwpi-http-cdas`.

By default, this built-in shared library is hard-coded to map Entrust Proxy header data to a valid Security Access Manager identity. You must customize this file to authenticate other types of special header data and, optionally, map this data to a Security Access Manager identity. See the *IBM Security Access Manager for Web: Web Security Developer Reference* for API resources.

You can configure the HTTP header authentication mechanism by entering the http-request entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

For example:

**Solaris:**

```
[authentication-mechanisms]
http-request = libpdwpi-http-cdas.so
```

**Windows:**

```
[authentication-mechanisms]
http-request = pdwpi-http-cdas.dll
```

## Cookie authentication

Cookie authentication provides the ability to authenticate and maintain session information using HTTP Cookies. Cookie authentication is an extension of the existing HTTP header authentication.

The **auth-source** entry in the `[http-hdr]` stanza of the configuration file can be used to control whether the configured authentication data (header) is retrieved from a HTTP header or from a cookie. See "[http-hdr]" on page 252.

# Configuring IP address authentication

The IP Address of incoming requests can be used to both maintain session state and to authenticate client requests using values in the client address headers.

Configuring the plug-in to use the IP address for maintaining session state is invalid without also configuring it to use the IP address to authenticate the client request. However, usage of the IP address to authenticate users is valid if the plug-in does not use the IP address to track user sessions.

## Enabling authentication using the IP address

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using the IP address of the request initiator, assign the reference *ip-addr* to the **authentication** entry as in the following:

```
[common-modules]
authentication = ip-addr
```

To enable the use of the IP address to track user sessions, assign the reference *ip-addr* to the **session** entry as in the following:

```
[common-modules]
session = ip-addr
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for IP address authentication exists, as in the following:

```
[modules]
ip-addr = pdwpi-ipaddr-module
```

## Configuring the IP address authentication mechanism

The IP address authentication mechanism is the same as that for HTTP headers. The **http-request** entry specifies the shared library for the IP address authentication mechanism.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-http-cdas`.
- On Windows, the file that provides the built-in mapping function is a DLL called `pdwpi-http-cdas`.

You can configure the IP address authentication mechanism by entering the http-request entry with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

For example:

**Solaris:**

```
[authentication-mechanisms]
http-request = libpdwpi-http-cdas.so
```

**Windows:**

```
[authentication-mechanisms]
http-request = pdwpi-http-cdas.dll
```

# Configuring LTPA Authentication

The plug-in can use LTPA cookies to authenticate users. LTPA cookies can either be provided by Security Access Manager WebSEAL or by IBM WebSphere server.

## Enabling LTPA Authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of LTPA for authenticating requests.

```
[common-modules]
authentication = ltpa
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for LTPA authentication exists; that is:

```
[modules]
ltpa = pdwpi-ltpa-module
```

## Setting the Key Details

The actual LTPA cookie, which is received, is encrypted by the sender. The cookie must be decrypted before authentication can take place. The **[ltpa]** stanza in the `pdwebpi.conf` configuration file contains the key details required for the decryption process:

```
[ltpa]
ltpa-keyfile = full path of keyfile
ltpa-stash-file = password stash file location
ltpa-password = password in lieu of the stash file
ltpa-cookie-name = name of the cookie containing the LTPA token
ltpa-lifetime = lifetime in seconds of the LTPA cookie
ltpa-generate-unauth = enables or disables the creation of LTPA cookies
 for unauthenticated users
```

where:

- The **ltpa-keyfile** entry specifies the name of the keyfile supplied from the originating machine. The keyfile entry is required.
- The **ltpa-stash-file** entry specifies the name of the file that contains the password to the keyfile. This entry is optional, although if it does not exist, the **ltpa-password** entry must exist. This entry takes precedence over any specified **ltpa-password**.
- The **ltpa-password** entry is required only if the **ltpa-stash-file** entry does not exist. It should contain the clear text password to the specified keyfile.
- The **ltpa-cookie-name** sets the name of the cookie in which the LTPA token is stored. This entry is optional. If no value is defined, the default value `LtpaToken` is used.
- The **ltpa-lifetime** entry specifies the LTPA cookie lifetime in seconds. This entry is required.
- The **ltpa-generate-unauth** entry determines whether LTPA cookies are created for unauthenticated users. LTPA cookies are not useful for unauthenticated users and might result in unexpected behavior. This configuration option is only provided for backwards compatibility with Security Access Manager Plug-in for Web Servers versions 5.1 and earlier. These earlier versions incorrectly generated LTPA cookies for unauthenticated users. The default value is `false`.

## Configuring LTPA post-authorization processing

The LTPA module is configured for post-authorization processing as part of a single sign-on solution to a WebSphere application server. See “Single sign-on to WebSphere application server using LTPA cookies” on page 151 for configuration details.

## Handling LtpaToken2 cookies

The **[ltpa2]** module of the configuration file deals with the **LtpaToken2** cookie, which is used by IBMWebSphere Application Server.

For **[ltpa2]** configuration details, see "[ltpa2]" on page 262.

The **LtpaToken2** cookie is encoded using enhanced security over the **LtpaToken** cookie. To use the **LtpaToken** cookie see "[ltpa]" on page 261.

The **ltpa** and **ltpa2** modules are independent and may be used together.

The **ltpa2** stanza contains all of the details for the **LtpaToken2** cookie-based authentication, session and post authorization modules. As an authentication module, this module allows sign-on to IBM Security Access Manager Plug-in for Web Servers using an **LtpaToken2** cookie. As a session module, this modules allows session tracking using the value of the **LtpaToken** cookie. As a post authorization module, this module is designed to allow single sign-on capability with a WebSphere server.

# Configuring the redirection of users after logon

Using the **login-redirect** module you can configure the plug-in to redirect users to a specific URL after they have successfully been authenticated. This may be useful in cases where you want all users to be directed to a portal rather than the Web page they requested, or for presenting the user with a welcome page or the start page for an on-line application.

The plug-in logon redirect functionality works independently of the method used for authenticating users. A redirection does not occur for a step-up authentication or for re-authentication.

## Enabling user redirection

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods.

To redirect the user to a specific URI after their initial logon and authentication, assign the reference *login-redirect* to the **pre-authzn** parameter; as in the following:

```
[common-modules]
pre-authzn = login-redirect
```

**Note:** Place the **login-redirect** entry first in the pre-authorization module list, otherwise, another authentication module redirect may take precedence. It must be placed before the account management pre-authorization module.

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for **login-redirect** exists, as in the following:

```
[modules]
login-redirect = pdwpi-loginredirect-module
```

## Configuring user redirection parameters

User redirection entries are configured in the **[login-redirect]** stanza of the `pdwebpi.conf` configuration file.

```
[login-redirect]
redirect-uri = redirect uri
```

Use the **redirect-uri** entry to specify the URI you want users directed to following a successful logon. The specified URI can either be a relative URI or an absolute URI.

## Using an external authentication service

The plug-in can be configured to accept authentication information from a third-party application. This authentication information can be either in the form of a user name or a privilege attribute certificate. A privilege attribute certificate is a digital record of a user's credentials.

The plug-in passes requests for resources configured for external authentication directly to the configured external authentication provider. The client negotiates authentication with the third-party application without plug-in involvement. Once the client is authenticated, the plug-in intercepts the response and constructs the client credential using the authentication information contained in the intercepted request.

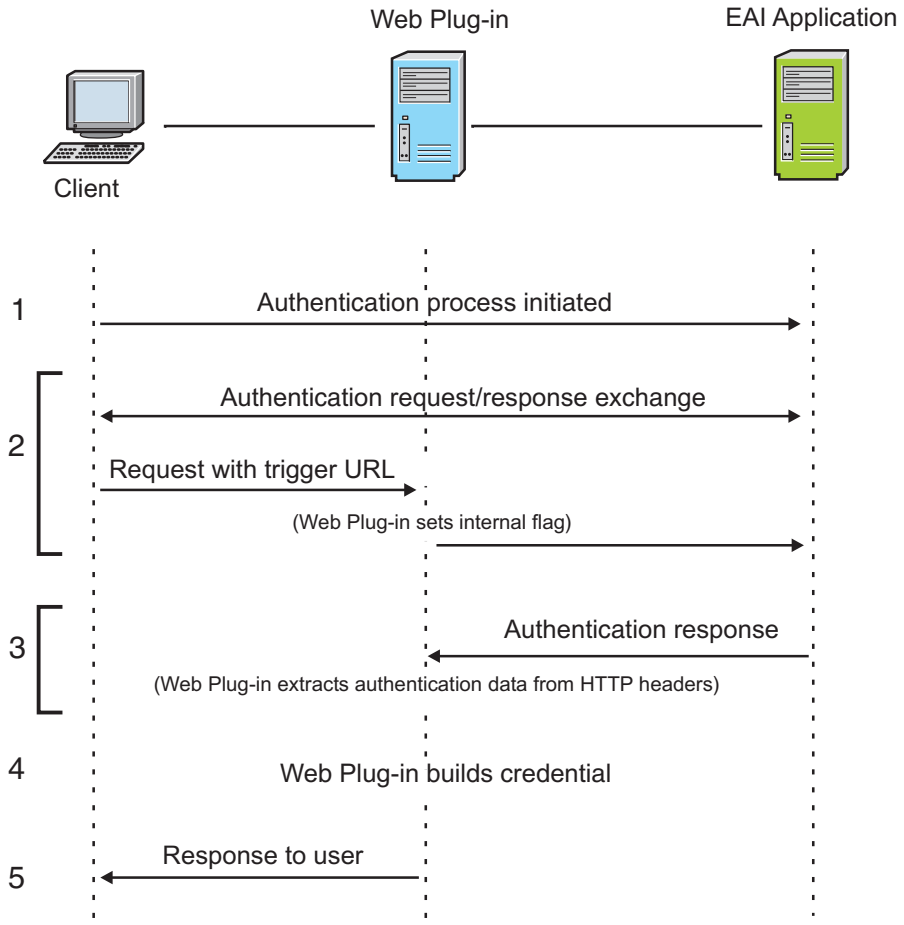


*Figure 5. Sample dataflow for Extended Authentication Interface*

The diagram shows a sample deployment using external authentication, wherein the following processes can occur:

- An unauthenticated client requests a protected resource.

- A redirect is returned to the configured redirect URL.
- The client authenticates to the External Authentication Server. These exchanges are unauthenticated, so the resource must be accessible (as a result of the policy you set) to such users. HTTPS is typically used for transmission integrity.
- At some point the client requests a special URI called a **trigger URI**.
- The exchanges are streamed through the Web Plug-in, which recognizes the URI and sets a flag indicating that the response to the request should be examined to see if it is an **authentication response** (that is, containing data relating to an authentication operation).
- If it is *not* an authentication response, the response is returned to the client.
- If it is an authentication response, the Web Plug-in generates a credential from the **authentication data** in the response.
- If a credential is *not* successfully generated, an AZN API error is returned to the client.
- If a credential is successfully generated from the authentication data, the client is directed to the originally requested resource.

## Enabling the external authentication interface

To enable the use of the external authentication interface configure the **ext-auth-int** module for both authentication, pre-authorization and response processing. That is:

```
[common-modules]
pre-authzn = ext-auth-int
authentication = ext-auth-int
response = ext-auth-int
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for **ext-auth-int** exists. That is:

```
[modules]
ext-auth-int = pdwpi-ext-auth-int-module
```

## Configuring the external authentication interface

Entries for external authentication configuration are contained in the **[ext-auth-int]** stanza of the plug-in configuration file.

The **auth-url** configuration entry identifies the URLs that should be ignored by the plug-in and passed directly to the external authentication service. These URLs should not be protected by Security Access Manager.

The entered values for the **auth-url** configuration entry can be either an absolute URL, or a server relative URL.

For example:

```
[ext-auth-int]
auth-url =/eai/app?orig-url=%URL%
```

The redirect URL is restricted to the length supported by the client server. The plug-in does not check the validity of the redirect URL against the length restrictions imposed by the client server. You should limit the data returned in the redirect URL to avoid errors.

Once a request has been identified as requiring authentication by an external application, the plug-in checks each request for the **trigger-url** configuration entry, which prompts the plug-in to create a client credential.

Configuring a matching string for the trigger URL limits the set of responses that the plug-in needs to examine for authentication data. The configured trigger URLs should be as specific as possible as this limits the number of requests and responses between the client and the external service that the plug-in examines for authentication data.

For example, a URL leading to a response from the external authentication server might look like:

```
http://plugin.IBM.com/eai/page.asp?url=/return_login_data.asp
```

A corresponding trigger URI could be configured as:

```
trigger-url = /eai/page.asp*login*
```

The configured trigger URL can contain the following special characters which are used when matching the request URL:

- ? : match any single character;
- * : match any number of characters;
- \ : turn off the special meaning of the character that follows;
- [ ] : match any one of the enclosed characters.

For example:

```
trigger-url =uri =/eai/page.asp*login*
trigger-url =uri =/cgi-bin/*
```

It is important that **trigger-url** information be concise. Responses which do not contain correct authentication information are streamed back to the client. Such additional inter-process communication between the Web server and the Plug-in will have a negative impact on performance.

It is preferable to configure a relative rather than absolute URL for the **trigger-url** entry. If you cannot avoid using an absolute URL then ensure that the absolute URL is not subject to any restrictive Security Access Manager authorization decisions.

More than one **trigger-url** can be specified. The plug-in pattern matches the trigger URL against the full URL, including the query portion.

Once the trigger URL containing the authentication information is identified, the plug-in examines the response header for configured headers. These are the headers containing authentication data used to authenticate the user. By default the configured header entries are:

```
redirect-url-hdr-name =am-eai-redir-url
pac-hdr-name =am-eai-pac
pac-svc-id-hdr-name =am-eai-pac-svc
user-id-hdr-name =am-eai-user-id
user-auth-level-hdr-name =am-eai-auth-level
user-qop-hdr-name =am-eai-qop
user-ext-attr-list-hdr-name =am-eai-xattrs
```

The following table provides more information on these header entries.

*Table 18. HTTP header authentication data*

| Authentication Type | Header | Mandatory? | Notes |
|---|---|---|---|
| Privilege Attribute Certificate (PAC) | `am-eai-pac` | Yes | The PAC is a digital document that contains authentication and authorization attributes and capabilities. This form of authentication will take precedence over the user ID form of authentication. For more information on the PAC, see the *IBM Security Access Manager for Web: Authorization C API Developers Reference*. |
| PAC Service ID | `am-eai-pac-svc` | No | The service ID that should be used to convert the PAC into a credential. If no service ID is specified the default PAC service will be used. |
| User ID | `am-eai-user-id` | Yes | The ID of the user to generate the credential for. |
| Authentication Level | `am-eai-auth-level` | No | The authentication level at which the generated credential will sit. If no value is specified a default value of 1 will be used. |
| Quality Of Protection (QOP) | `am-eai-qop` | No | The quality of protection that will be associated with the credential. If no value is specified it will default to *None*. |
| Extended Attribute List | `am-eai-xattrs` | No | A comma delimited list of HTTP header names that will be added to the credential as extended attributes. |
| Redirect URL | `am-eai-redir-url` | No | Specifies the URL that the client will be redirected to upon successful authentication. If no URL is specified, the cached URL will be used. If no cached URL is found, the URL first used to trigger the authorization will be used. |
| Session Lifetime | `am_eai_xattr_session_lifetime` | No | If this special extended attribute header is present, it will be added to the credential as an extended attribute. The session lifetime for the newly created session will be set to the value of this extended attribute, which over-writes the existing **max-sessions-lifetime** setting in the `[sessions]` stanza. |

If you want the redirect URL (provided in the authentication response from the EAI application) to take precedence over the originally requested URL in the post-authentication redirect, set the **use-redirect-url-first** value to *true*. The default value is *false*.

```
use-redirect-url-first = false
```

# Adding extended attributes for credentials

You can add extended attributes for credentials.

This section contains the following topics:



## Mechanisms for adding extended attributes to a credential

The Security Access Manager Plug-in for Web Servers authentication process accesses the Security 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.

The plug-in supports several different mechanisms (services) that allow administrators and application developers to extend the authentication process. When the plug-in conducts the authentication process, it checks to see if any external services have been implemented and configured. When they have, the plug-in 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:

- Credential attribute entitlement service

  This entitlement service is built-in to Security Access Manager by default. This service 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 credential 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 `pdwebpi.conf` configuration file. In Version 6.0, you should use the built-in entitlement service to obtain user registry data. For configuration information, see "Entitlement service configuration" on page 107
- 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. Security Access Manager supports this as part of the authorization API. For more information, see the *IBM Security Access Manager for Web: Authorization C API Developer Reference*.
- Credential extended attributes external authentication mechanism.

  The plug-in provides an external authentication API interface that can be used to develop external authentication services.

You can use the plug-in's external authentication API to develop your own external authentication service. This can be used when the need to obtain user authentication information extends beyond entitlements information.

A credential extended attributes mechanism typically used 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 Security Access Manager user ID. For more information, see the *IBM Security Access Manager for Web: Web Security Developer Reference*.

## Entitlement service configuration

To configure entitlement service, 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"

### 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 Security Access Manager configuration file. Typically, this is done in the plug-in configuration file.

Go to the Security 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

Take these steps to define your use of the entitlement service:

#### Procedure

1. Verify that the credential attributes entitlement service is configured. The following default entry should be present in the plug-in configuration file:
   `[aznapi-entitlement-services]AZN_ENT_EXT_ATTR = azn_ent_ext_attr`

   **Note:** The plug-in automatically takes the value `azn_ent_ext_attr` and finds the corresponding shared library. For example, on Solaris, `libazn_ent_ext_attr.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 entry **cred-attribute-entitlement-services**. You can choose a value, such as TAM_CRED_ATTRS_SVC. For example:`[aznapi-configuration ]cred-attribute-entitlement-services = TAM_CRED_ATTRS_SVC`

### 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 plug-in 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 Security Access Manager for Web: Authorization C API Developer Reference*.

Review the following example entry.

```
[TAM_CRED_ATTRS_SVC]
eperson = azn_cred_registry_id
group = cn=enterprise, o=ibm

[TAM_CRED_ATTRS_SVC:eperson]
tagvalue_credattrs_lastname = sn
tagvalue_credattrs_employeetype = employeetype
tagvalue_credattrs_address = homepostaladdress
tagvalue_credattrs_email = email

[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=ibm`

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
businesscategory=finance
```

Using the example configuration entries shown above, the attribute list returned would have the following entries:

| Attribute name | Attribute value |
|---|---|
| credattrs_lastname | Smith |
| credattrs_employeetype | bankteller |
| credattrs_address | 3004 Mission St Santa Cruz CA 95060 |
| credattrs_email | joeuser@bigco.com |
| credattrs_businesscategory | finance |

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 = employeetype
```

## Adding registry extended attributes to the HTTP header (tag value)

It is often useful to attach user-specific information from the registry (for example, telephone number, e-mail address) to the headers of HTTP authenticated requests.

### About this task

User-specific information allows multiple applications to access the attached information without having to constantly query the registry. The nature of this information is that it is relatively static and is never updated by any of the applications that use it. This data is placed into the user credential as part of the **ivauthn** authentication process. This information can also be attached to the users credential through a user-implemented external authentication mechanism.

The following process flow describes the sequence of events:

- User-defined supplemental data from any field of a user's registry account is added as extended attribute data in the user's Security Access Manager credential.
- When configured for tag value post-authorization processing, the plug-in extracts the extended attribute data and places it in the HTTP header of the request.
- The back-end application can extract the data from the header without requiring special code or the authorization API.

Follow the procedure to configure the plug-in for inserting registry extended attribute information into an HTTP header.

### Procedure

1. Configure tag value post-authorization in the Web Plug-in.

   Refer to "Enabling tag value processing" for details on how to do this.
2. Add the extended attributes to the /PDWebPI/*host* object in Access Manager.

   For example (entered as one line):

   ```
   pdadmin> object modify /PDWebPI/host
   set attribute HTTP-Tag-Value ldap-home-phone=homePhone
   ```

## Enabling tag value processing

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable processing using tag values, assign the reference *tag-value* to the **post-authzn** parameter:

```
[common-modules]
post-authzn = tag-value
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for tag value exists, as in the following:

```
[modules]
tag-value = pdwpi-tag-value-module
```

## Configuring tag value parameters

Tag value entries are configured in the **[tag-value]** stanza of the `pdwebpi.conf` configuration file.

```
[tag-value]
cache-definitions = yes
cache-refresh-interval = 60
```

The **cache-definitions** entry enables or disables caching of tag value definitions that are attached to the object space. The **cache-refresh-interval** defines the refresh interval in seconds for the cache of definitions.

If need be, you can configure a prefix to add to credential attribute names used for tag-value HTTP headers. This prefix is:

- Used as a search string by the tag-value module when searching credential attributes.
- Added to the session ID credential attribute.
- Added by the switch-user module to the credential attribute it uses to store the administrator's user name.

Specify the prefix using the **tag-value-prefix** entry in the **[pdweb-plugins]** stanza of the plug-in configuration file.

This entry can be overridden for a specific virtual host using a **[*virtual_host*]** stanza for that virtual host. The default behavior is to have no prefix.

# Supporting Multiplexing Proxy Agents (MPA)

Security Access Manager provides solutions for securing networks that use a Multiplexing Proxy Agent (MPA). Multiplexing Proxy Agents (MPA) are gateways that accommodate multiple client access.

Gateways establish a single authenticated channel to the secured Web server and "tunnel" all client requests and responses through this channel. To the plug-in, the information across this channel initially appears as multiple requests from one client. The plug-in must distinguish between the authentication of the MPA server and the additional authentication of each individual client.

A common example of such gateways are Wireless Access Protocol (WAP) gateways. Security Access Manager WebSEAL also acts as an MPA when configured with a junction to the host Web server to allow single sign-on between WebSEAL and the plug-in. To configure such a solution, the iv-header authentication module can be used. See Chapter 6, “Web single sign-on solutions,” on page 149 for more details on configuring for SSO.

## Valid session data types and authentication methods

Because IBM Security Access Manager Plug-in for Web Servers 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 different than the session data and authentication method used by the client. The table below lists the valid session types for the MPA and the client:

*Table 19. Valid session data types for MPA*

| Valid Session Types | |
|---|---|
| **MPA-to-plug-in** | **Client-to-plug-in** |
| SSL Session ID | |
| HTTP Header | HTTP Header |
| BA Header | BA Header |
| IP Address | |
| Cookie | Cookie |

- 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 a 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 use of SSL session IDs to maintain session state changes. Normally, having SSL session IDs configured to maintain session state, 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.

The authentication method used by the MPA-to-plug-in must be different than the authentication method used by the client-to-plug-in. The table below lists the valid authentication methods for the MPA and the client:

*Table 20. Valid MPA authentication types*

| Valid Authentication Types | |
|---|---|
| **MPA-to-plug-in** | **Client-to-plug-in** |
| Basic Authentication | Basic Authentication |
| Forms | Forms |
| Token | Token |
| HTTP Header | HTTP Header |
| Certificate | |
| IP Address | |

- 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. If MPA support is enabled, this restriction is removed. This allows the MPA to log on, for example, with Forms (or token) and clients to log on with Basic Authentication over the same transport.

## Authentication process flow for MPA and multiple clients

The following process flow occurs for MPA and multiple client authentication.

### Procedure

1. Make the following configuration changes:
   - Enable support for Multiplexing Proxy Agents in the configuration file.
   - Create a Security Access Manager account for the specific MPA gateway.
   - Grant Proxy ([PDWebPI]p) access for this account to the MPA Protected Object for the virtual host to which proxied requests will be directed. In the default configuration, this can be achieved by making the user a member of the **pdwebpi-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 the plug-in using the client request.
6. The MPA authenticates to the plug-in (using a method distinct from the client) and an identity is derived for the MPA (that already has a plug-in account).
7. The plug-in verifies the MPA's membership in the **pdwebpi-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 the plug-in needs to further identify the owner of the request. The MPA is able to distinguish the multiple clients for proper routing of logon prompts.
10. The client logs in and authenticates using a method distinct from the authentication type used for the MPA.
11. The plug-in builds a credential from the client authentication data.
12. The session data type used by each client must be distinct from the session data type used by the MPA.
13. The Authorization Server permits or denies access to protected objects based on the user credential and the object's ACL permissions.

## Enabling MPA authentication

The **mpa-enabled** entry in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file enables or disables MPA authentication. The valid settings are *true* and *false* for enabling and disabling MPA authentication respectively. MPA authentication is disabled by default. MPA authentication can be set for individual virtual hosts by specifying the **mpa-enabled** entry in the **[*virtual_host*]** stanza of the configuration file.

To identify a new session as being the primary session established by an MPA, an authorization decision is made, testing for the Proxy ([PDWebPI]p) permission on the MPA protected object. By default the MPA protected object is defined to be /PDWebPI.

To override this default setting, for example to define different sets of principals as representing MPAs for each virtual host, a value can be specified for the **mpa-protected-object** configuration parameter. This entry may be overridden for each virtual host by specifying a value for it in the **[*virtual_host*]** stanza of the configuration file.

For example, to enable MPA access for the ibm.com virtual host but not the lotus.com virtual host, use the following settings in the `pdwebpi.conf` configuration file:

```
[pdmweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com

[ibm.com]
mpa-enabled = yes
```

To define members of the ibm-mpa-servers group as being MPAs for requests to the ibm.com virtual host and lotus-mpa-servers group as being MPAs for requests to the lotus.com virtual host, use the following configuration:

```
[pdweb-plugins]
virtual-host = ibm.com
virtual-host = lotus.com

[ibm.com]
mpa-enabled = yes
mpa-protected-object = /PDWebPI/ibm.com

[lotus.com]
mpa-enabled = yes
mpa-protected-object = /PDWebPI/lotus.com
```

and define the following Security Access Manager policy:

```
pdadmin> acl create ibm-mpa
pdadmin> acl modify ibm-mpa set group ibm-mpa-servers T[PDWebPI]p
pdadmin> acl create lotus-mpa
pdadmin> acl modify lotus-mpa set group lotus-mpa-servers T[PDWebPI]p
pdadmin> acl attach /PDWebPI/ibm.com ibm-mpa
pdadmin> acl attach /PDWebPI/lotus.com lotus-mpa
```

The **mpa-protected-object** configuration entry specifies the object against which the authorization decision will be made.

## Create a user account for the MPA

Refer to the *IBM Security Access Manager for Web: Base Administration Guide* and the *IBM Security Access Manager for Web: Web Portal Manager Online Help* for information on creating user accounts.

## Add the MPA account to the pdwebpi-mpa-servers group

IBM Security Access Manager Plug-in for Web Servers creates a group for easily administering MPA servers. This group is called **pdwebpi-mpa-servers**. The default-pdwebpi ACL attached to `/PDWebPI` grants Proxy ([PDWebPI]p) permission to members of the **pdwebpi-mpa-servers** group.

When installed in a Security Access Manager secure domain that has at least one WebSEAL configured, the default-pdwebpi ACL is configured so that it also grants Proxy permission to members of the **webseal-servers** and **webseal-mpa-servers** groups.

You may choose your own groups and ACLs used to control identification of principals as Multiplexing Proxy Agents.

See the *IBM Security Access Manager for Web: Base Administration Guide* and the *IBM Security Access Manager for Web: Web Portal Manager Administration Guide* for information on managing groups.

## Extended CDAS User Mapping Rules

The cross-domain-authentication-server (CDAS) is used to authenticate a user and provide a Security Access Manager user identity. The latest client certificate user-mapping CDAS functionality in Security Access Manager gives you more flexibility when mapping attributes contained within the certificate to the Security Access Manager user identity.

The rules which govern the mapping of the client certificate are defined using XSL style notation, while XML is used for the data that forms an input to the rules. The Universal Management Infrastructure XML document model (or UMI XML model) defines the restrictions placed on the XSL/XML model, making the interface simple and yet functional for certificate purposes. The output from the XML and XSL input is a single string, which is used to determine the Security Access Manager user identity.

For full details of the extended CDAS User Mapping functionality, including valid string formats and configuration instructions, see the section titled *Client Certificate User Mapping* in the *IBM Security Access Manager: WebSEAL Administration Guide*.

**Note:** For the IBM Security Access Manager Plug-in for Web Servers, only the following subset of IBM Global Security Kit (GSKit) attributes can be specified in the certificate XML:

```
* cert_dn_printable
* cert_issuer_dn_printable
* cert_serial_number
```

For more information about GSKit, see the IBM *Secure Sockets Layer Introduction and iKeyman User's Guide*.

# Chapter 4. Managing session state

IBM Security Access Manager Plug-in for Web Servers uses session state information to identify the source of incoming requests. It uses the identity of the request source to maintain session state between client and server when the client performs numerous requests within one session.

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 the need for repeated authentication. The client can log on once and make numerous requests without performing a separate log on for each request.

The plug-in supports a variety of methods for maintaining session state with a client. For environments where there is more than one plug-in or WebSEAL protecting access to resources, a separate application is available in the Security Access Manager product suite called the IBM Security Access Manager session management server (SMS).

The SMS is an enterprise application that runs as a WebSphere Application Server service. The SMS manages sessions across clustered Web servers and allows for login tracking and single sign-on. The SMS permits an administrator to monitor and administer user sessions as required.

The SMS requires a separate installation and configuration and has associated deployment considerations that you should become familiar with before installing the product. The *IBM Security Access Manager: Shared Session Management Administration Guide* contains a discussion of SMS deployment considerations and provides configuration and administration information. Details on SMS installation can be found in the *IBM Security Access Manager for Web: Installation Guide*.

Configuration of the SMS is also required at the client - either the plug-in or WebSEAL, or both depending on your architecture. Details on configuring the plug-in to use the SMS are located in the first part of this chapter. Equivalent WebSEAL configuration information is included in the *IBM Security Access Manager: WebSEAL Administration Guide*.

For environments where there exists only one plug-in enabled Web server protecting your Web resources, the SMS will not provide any advantage and one of the following information types should be configured for managing session state:

- SSL session ID
- Basic Authentication
- Server-specific session cookie
- HTTP header data
- IP address
- LTPA cookies
- IV headers

Details on configuring the plug-in to use the above information types for session management can be found in “Managing Session State in non-clustered environments” on page 121.

Session management can be configured for individual virtual hosts. So it is possible that within one environment, some virtual hosts may use the SMS while others may use any of the information types in the above list.

This chapter includes the following topics:



## The Session Management Server (SMS)

The IBM Security Access Manager session management server (SMS) uses a single session ID across replicated servers to represent the user's session. Collections of replicated servers are known as replica sets. Replica sets may be grouped to form a session realm. The SMS can maintain sessions across replica sets within a session realm. A session realm may consist of replica sets of any kind; for example, plug-in replica sets, WebSEAL replica sets, or a combination of both.

The diagram below illustrates how replica sets and virtual hosts coexist in such a deployment.

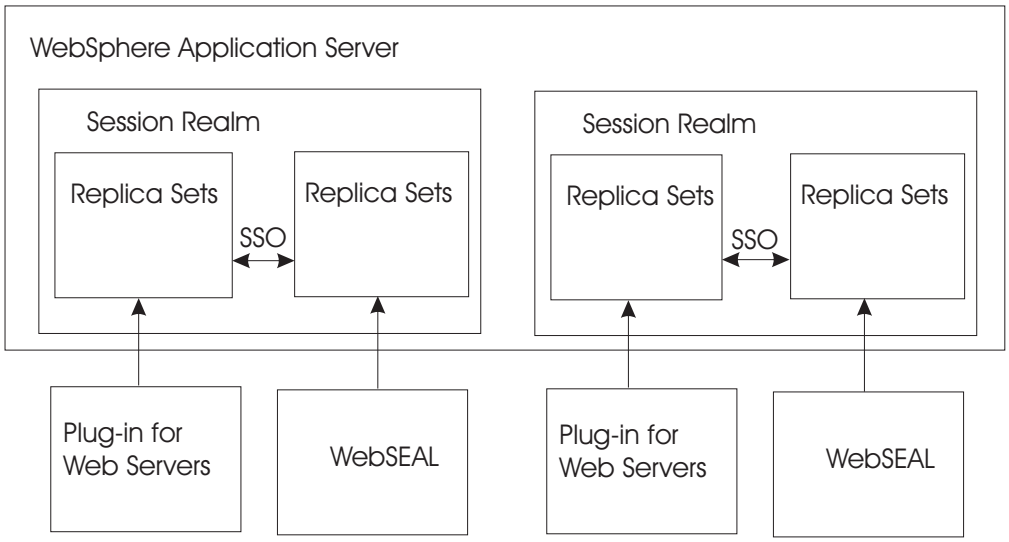


*Figure 6. Example deployment of replica sets*

When a request for a session is received at a server, that server queries the SMS for any session information it may have. If the SMS has the information, the session is reconstructed at the server; otherwise, the request is treated as unauthenticated.

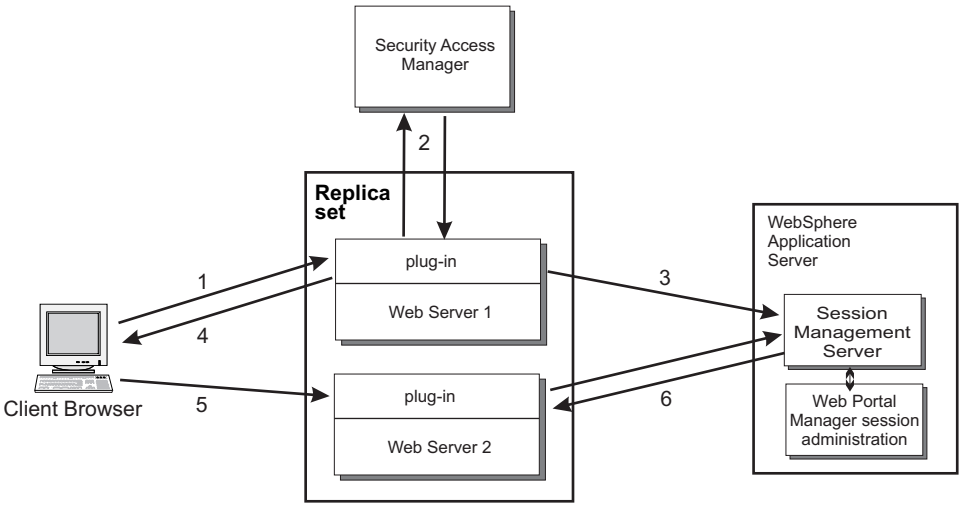


*Figure 7. Basic session flow using the SMS*

The diagram above shows a basic representation of the SMS and its associated session flows. The following processes occur in the example:

1. A user's request for a protected resource is intercepted by the plug-in. The plug-in challenges the user to authenticate.
2. The user's credentials are passed to Security Access Manager to authorize the request. Successful requests are returned to the plug-in. The plug-in creates a user session and associates it with the user's credential and stores it in the local session cache to promote performance.
3. The SMS is notified of the new session and associated credential.
4. The plug-in sends a session cookie to the user's browser.
5. The user accesses a new server in the replica set using the session cookie. This server may contain a new part of the same application or may be a copy of the original server that the user has been directed to for reasons of failover or load balancing.
6. Using the session cookie, server 2 consults the SMS to determine whether the user has already authenticated. The SMS replies with the user's cached credential which the server uses to trust the client.

After the initial logon, the processes in the above example all occur without the need for user intervention. In certain architectures, participating servers in a replica set might require a stronger level of authentication than the user originally logged in with. In such cases the user is prompted to re-authenticate to the higher level. This process is known as step-up and is described more fully in "Authentication-strength Protected Object Policy (Step-up)" on page 136.

If the example architecture in figure 5 did not use the SMS, then the re-direction to server 2 would require the user to re-authenticate. This is because, without the SMS, server 2 has no record of the user's original credentials.

## Configuring the plug-in to use the SMS

To use the SMS, the plug-in distributed session module, **dsess**, needs to be configured as a session module. The **dsess** module uses cookies to maintain session state information and it uses the SMS to distribute the session information throughout the participating servers in the session realm.

In the plug-in configuration file, add the following entry to the **[common-modules]** stanza:

```
session = dsess
```

The library name for the **dsess** module is configured by default in the **[modules]** stanza.

When configuring the plug-in to use the SMS, the **dsess** module is the only session module required. However, different virtual hosts can use different mechanisms for maintaining session state. When such a facility is required, the **[common-modules]** stanza should be configured for specific virtual hosts. For details on configuring for virtual hosts refer to, "Configuring authentication for virtual hosts" on page 51.

Configuration entries for the **dsess** module are located within the **dsess** stanza of the configuration file. First, you need to point the plug-in at the SMS. Within the **dsess** stanza, define a cluster name in the `dsess-cluster-name` entry. For example:

```
dsess-cluster-name = cluster1
```

You then need to define details for that cluster. This can be done in either the default **[dsess-cluster]** stanza, or in a qualified cluster stanza with an appropriate name (for example, using the `dsess-cluster-name` entry above, `[dsess-cluster:cluster1]`). If specific details for a cluster are not found in a qualified `[dsess-cluster:`*`cluster_name`*`]` stanza, then default details from the **[dsess-cluster]** stanza will be used.

For architectures where more than one SMS is installed in a failover configuration, multiple instances of this configuration entry should be created.

Uniquely identify each plug-in instance within a replica set using a unique name configured in the `dsess-replica-name` entry. For example:

```
dsess-replica-name = human_res_serv_3
```

If this entry is not set, the plug-in instance is identified to the SMS using its authorization server name.

Identify the replica set within which the plug-in functions using the `dsess-replica-set` entry. For example:

```
dsess-replica-set = human_res_replica_set
```

If this entry is not set, the value defaults to the name of the virtual host. The default value for this configuration entry should be correct for the majority of environments.

Replica set configuration entries also need to be set during SMS configuration.

## Configuring secure communication between the plug-in and SMS

Communication between the plug-in and SMS can be configured over SSL using the Security Access Manager server certificate. Other certificates can be used however the Security Access Manager server certificate is the default and easiest to configure. Generally, any certificate considered valid by the host WebSphere Application Server is acceptable as authentication.

When SSL is to be used for plug-in to SMS communications (that is, the URL configured in the `server` entry of the **[dsess-cluster]** stanza begins with HTTPS) the following configuration entries are required. These entries are also required

when a certificate other than the one used by the server is needed for communication with the policy database. Details of the default certificate can be found in the **[ssl]** stanza.

Use the `ssl-keyfile` configuration entry to specify the absolute path name of the key database file which houses the client certificate. For example:

```
ssl-keyfile = /var/pdwebpi/www/certs/sms.kdb
```

Using the `ssl-keyfile-stash` entry, specify the name of the password stash file for the key database file. For example:

```
ssl-keyfile-stash = /var/pdwebpi/www/certs/sms.sth
```

Using the `ssl-keyfile-label` entry, specify the label of the client certificate within the key database.

```
ssl-keyfile-label = Plug-in-SMS
```

In rare situations you may need your session cookies to operate across both HTTP and HTTPS connections. Having session cookies travel across insecure network connections is not advisable as a nefarious agent may be able to steal a session cookie and assume the identity of a valid user. However, the `dsess-use-same-session` entry is available for customers requiring such a function. Set this entry to *yes* if you require your session cookies to operate across both HTTP and HTTPS.

The `dsess-propagate-unauth` entry enables or disables the distribution of unauthenticated sessions across multiple servers. Storing a record of the unauthenticated session at the SMS by setting this entry to *true* means the unauthenticated session information is available to all replicated plug-in instances. Doing this can leave the SMS open to denial-of-service attacks. The SMS is a vital component of the architecture and denial of service attacks at this level will potentially have far greater consequences than at the level of the plug-in.

The session cookie the SMS uses to maintain session state across the replica set can be used across the entire domain by enabling the `cookie-for-domain` entry. This effectively enables the SMS for session tracking across a DNS domain.

```
cookie-for-domain = true
```

When using SSL for communications between the plug-in and the SMS a certificate is exchanged. The `ssl-valid-server-dn` entry allows you to specify a list of trusted server distinguished names. The list is formed using the domain names of these root certification authorities. The plug-in will only accept from the SMS certificates signed by the listed root CA domain names. If this entry is not specified then all root CA certificates are considered valid. An example entry might be:

```
ssl-valid-server-dn = cn=sms,o=company,c=us
ssl-valid-server-dn = cn=sms,o=other_company,c=us
```

## Configuring the SMS communication timeout

The `timeout` configuration entry in the **[dsess-cluster]** stanza defines the amount of time the plug-in should wait for a response from the SMS. The default value is 30 seconds but this can be increased or decreased. After this limit is reached, the SMS is marked as unavailable (and remains that way until communication can be reestablished). For example:

```
timeout = 60
```

In architectures where a firewall is used between the plug-in protected Web server and the WebSphere installation running the SMS, communication channels which are waiting for broadcast events may be shut down by the firewall.

The `response-by` configuration entry in the **[dsess-cluster]** stanza is used to specify maximum length of time that the SMS will wait for a response while maintaining the connection that broadcasts information updates to the client. This configuration entry should be set to a value which is less than the maximum amount of time that a connection can remain idle before it is terminated by a firewall.

For example:

```
response-by = 30
```

## Configuring SMS performance

The **dsess-sess-id-pool-size** property in the **[dsess]** stanza defines the number of inactive session ids that have been created in a pool for use at a later date. Such a pool of ready-to-go session ids can aid performance when multiple new users are required in a hurry. The default value is 512 and should only be adjusted when you are certain of the implications.

At plug-in startup, a certain number of communication handles are created and cached for anticipated exchanges with the SMS. The **handle-pool-size** configuration parameter in the **[dsess-cluster]** stanza sets this number of handles. The default setting is 10 and will be adequate for most installations. The entry can be adjusted if communication between the plug-in and the SMS is causing problems. If you do change this property, do not set it too high as doing so might have implications for other server communications that also require handles.

## Configuring session displacement

The SMS can be configured to limit a user to one active session in the system at one time. Two types of session displacement are available: interactive and non-interactive. With interactive session displacement, the user is prompted to either terminate their existing session or terminate the current login attempt when the SMS discovers another valid session exists for the user.

Interactive session displacement is useful when a user session remains active yet the user can no longer access it, for example in the case when the user's browser fails and they want to continue with their session. This facility is also useful for alerting users that they have another session active on another browser.

Non-interactive session displacement will terminate any existing session when a user logs in again. This is useful in situations when a user logs on using one browser and then leaves the session active and moves onto another browser. The original session is terminated when the user logs back in.

The following **pdadmin** command is used to set session displacement policy for a user (whether interactive or non-interactive):

```
policy set max-concurrent-web-sessions displace [-user user-name]
```

The **dsess-displacement-form** entry determines whether session displacement is interactive or non-interactive. When this entry is left out, or left blank, session displacement is non-interactive. When interactive session displacement is enabled,

a user logging on for a second time is advised that they already have an existing session. The page displayed to the user is also determined by the **dsess-displacement-form** entry. That is:

```
dsess-displacement-form = session_displacement.html
```

This defaults to `session_displacement.html`, located by default on the directory *install_path*`/nls/html/`*lang/encoding*`/`. This file is provided as a base from which you can format your own page. A new page can be created and placed on this directory or it can be stored in a new location by providing the full URI within the configuration entry. The HTML page can contain macros. See "Macro support" on page 9 for information on plug-in supported macros.

When the form configured in the **dsess-displacement-form** property is posted, the details are sent to the URI configured using the **dsess-displacement-uri**. For example (default):

```
dsess-displacement-uri = /dsess-displacement.form
```

### Changing session identifiers

The name of the cookie used to identify SMS sessions is configurable. Having configurable session identifiers also permits different virtual hosts within the same domain to participate in different SMS session realms. Configuring different session identifiers for each virtual host avoids name conflicts that would otherwise render participation by two separate virtual hosts impossible.

SMS session identifiers are configured using the `http-cookie-name` and `https-cookie-name` configuration entries in the **[dsess]** stanza of the plug-in configuration file. **http-cookie-name** specifies the name of sessions established over HTTP. **https-cookie-name** specifies the name of sessions established over HTTPS.

## Managing Session State in non-clustered environments

In architectures using a single plug-in, using the SMS to manage sessions will provide no advantage. Instead, one, or a combination of the plug-in session modules should be configured for managing sessions.

The plug-in calls each configured session module in turn and continues to search the configured session module types until one returns a credential. The plug-in then determines if the application references a Multiplexing Proxy Agent. If it is a Proxy Agent, then another session must exist for the real end user.

To find this other session, the plug-in continues to call the rest of the configured session modules. A user credential is returned when an existing session is found for which user authentication has already taken place. This credential is used to authorize the request. If none of the configured session modules returns a user credential, the session is either new or is a session for which no credential has been established.

## Configuring the plug-in session/credentials cache

The plug-in session cache allows a server to store the session ID information from multiple clients. The session cache accommodates both HTTPS and HTTP session state information.

The plug–in cache stores session ID information plus the credential information obtained for each client. Credential information is cached to eliminate repetitive queries to the user registry database during authorization checks. The plug-in cache also maintains session state information for SSL connections between the plug-in and the LDAP user registry.

There are several configuration parameters available for the plug-in cache that allow you to tune the performance of the cache.

**Note:** The values configured in the **[sessions]** stanza of the `pdwebpi.conf` configuration file may be overridden in the **[*module_name*]** stanza, and some may be further overridden in the **[*module_name:virtual_host_name*]** stanza on a per-virtual host basis.

## Setting the maximum concurrent entries value

The **max-entries** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, sets the maximum number of concurrent entries in each session module's session/credentials cache.

This value corresponds to the number of concurrent logon sessions for a particular session module. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm to allow new incoming logons.

The default number of concurrent logon sessions is 4096:

```
[sessions]
max-entries = 4096
```

## Setting the cache entry timeout value

The **timeout** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, sets the maximum lifetime timeout for an entry in the plug-in session/credentials cache.

The plug-in caches credential information internally. The session cache timeout parameter dictates the length of time authorization credential information remains in memory.

The parameter is not an inactivity timeout. The value maps to a "credential lifetime" rather than a "credential timeout". Its purpose is to enhance security by forcing the user to re-authenticate when the specified timeout limit is reached.

The default logon session timeout (in seconds) is 3600:

```
[sessions]
timeout = 3600
```

**Note:** If specified, the EAI session lifetime setting defined in the `am_eai_xattr_session_lifetime` header will override this setting. For details, see “Using an external authentication service” on page 102.

You can configure the session cache lifetime to be reset whenever re-authentication occurs. Each time re-authentication occurs, the session cache **timeout** value is reset. To configure session cache lifetime reset, use the **reauth-lifetime-reset** parameter in the **[sessions]** stanza of the `pdwebpi.conf` configuration file:

```
[sessions]
reauth-lifetime-reset = yes
```

The default value is *no*.

It is possible for the session cache lifetime value to expire while the user is performing a re-authentication. The session cache lifetime can expire after the re-authentication logon form is sent to the user and before the completed logon form is returned.

When the session cache lifetime value expires, the session cache entry is deleted. When the logon form is returned to the plug-in, 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 lifetime if the session cache lifetime expire during re-authentication.

The **reauth-grace-period** parameter in the **[sessions]** stanza of the `pdwebpi.conf` configuration file provides this time extension, in seconds. For example:

```
[reauthentication]
reauth-grace-period = 20
```

The default value, *0*, provides no extension to the session cache timeout value. The **reauth-grace-period** parameter applies to users with existing session cache entries and who are required to reauthenticate. For example, users might perform:

- Reauthentication resulting from POP security policy
- Reauthentication resulting from session cache inactivity
- Step-up authentication

The **reauth-grace-period** option is intended to be used in conjunction with the **reauth-lifetime-reset** = *yes* option.

## Setting the cache entry inactivity timeout value

The **inactive-timeout** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, sets the timeout value for logon session inactivity.

The default logon session inactivity timeout (in seconds) is 600:

```
[sessions]
inactive-timeout = 600
```

To disable this timeout feature, set the parameter value to *0*.

## Changing cookie names

By default session cookies are given a name that uniquely identifies them within your domain. The plug-in provides the ability to change this default identifier.

Session cookie names are configured using the `http-cookie-name` and `https-cookie-name` configuration entries in the **[session-cookie]** stanza of the plug-in configuration file. The `http-cookie-name` entry specifies the name of sessions established over HTTP. The `https-cookie-name` entry specifies the name of sessions established over HTTPS.

## Maintaining session state with the SSL session ID

Security Access Manager Plug-in for Web Servers can track sessions using the SSL session ID of incoming HTTPS requests. This facility is not available on IIS, as IIS does not make SSL session IDs available to the plug-in.

**Note:** SSL session IDs are not used for authenticating requests.

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all session, authentication, and post-authorization methods using the format *module_type = module-name*. To maintain session state using SSL session IDs, assign the word **ssl-id** to the **session** parameter as in the following:

```
[common-modules]
session = ssl-id
```

Ensure the shared library for ssl-id is configured in the **[modules]** stanza of the `pdwebpi.conf` configuration file. That is:

```
[modules]
ssl-id = pdwpi-sslsessid-module
```

## Maintaining session state using Basic Authentication

Basic Authentication (BA) is a method for authenticating users and maintaining session state through the input of a username and password. BA is defined by the HTTP protocol and can be implemented over HTTP and HTTPS.

Basic Authentication maintains session state by caching a record of the content of the Basic Authentication header.

To configure the plug-in to maintain session state using Basic Authentication, use the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. Enter the parameter **session** with the value as *BA*, as in the following:

```
[common-modules]
session = BA
```

If BA is used to maintain session state, it needs to be also used for user authentication. The **[common modules]** stanza of the configuration file should also set to BA for authentication.

```
[common-modules]
session = BA
authentication = BA
```

**Security Warning:**
Using the Basic Authentication authorization header to identify sessions can expose users of the Web server to unlimited password guessing attacks.

This is a limitation of the HTTP Basic Authentication scheme that includes the user's password in the authorization header.

IBM Security Access Manager Plug-in for Web Servers is not enabled by default. The Basic Authentication session identification capability is provided for those Administrators aware of all the security risks associated with Basic Authentication including this one.

Basic Authentication session identification can be securely used when a plug-in secured Web server operates behind a multiplexing proxy agent (for example behind a WebSEAL junction) using Basic Authentication to authenticate itself to the plug-in. In such a situation, the multiplexing proxy agent does not forward Basic Authentication authorization headers from users to the plug-in, making the attack impossible.

## Maintaining session state with Session Cookies

Using session cookies to hold session information is a method for maintaining plug-in session state. 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 identifier) 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 re-authentication of a client only to the server the client 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 a random number identifier that is used to index the cookie in the server's session cache — no other information is exposed in the session cookie. The session cookie cannot compromise security policy.

IBM Security Access Manager Plug-in for Web Servers 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 is located only in the browser memory (it is not written to the browser cookie jar on the disk).
- Cookie has a limited lifetime (configurable).
- Cookie has path and domain parameters that prohibit its use by other servers.

To configure the plug-in to use session cookies to maintain session state, use the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. Enter the parameter **session** with the value as *session-cookie*, as in the following:

```
[common-modules]
session = session-cookie
```

The **resend-pdwebpi-cookies** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, enables or disables the sending of the session cookie to the browser with every response. This action helps to ensure that the session cookie remains in the browser memory. The **resend-pdwebpi-cookies** parameter has a default setting of `no`:

```
[sessions]
resend-pdwebpi-cookies = no
```

Change the default setting to *yes* to send plug-in session cookies with every response.

## Maintaining session state using HTTP headers

IBM Security Access Manager Plug-in for Web Servers can be configured to use HTTP header information to identify sessions and maintain session state.

The plug-in can use HTTP headers for tracking sessions as well as authenticating users. Refer to, "Configuring HTTP header authentication" on page 97 for details on configuring the plug-in to use HTTP headers for client authentication. The HTTP header will only be used if it comes from a trusted MPA, as described in "Supporting Multiplexing Proxy Agents (MPA)" on page 110.

When using HTTP headers to maintain session state, the **[common-modules]** stanza of the `pdwebpi.conf` configuration file must be configured with the following values:

```
[common-modules]
session = http-hdr
```

A standard configuration of HTTP headers permits only one header to be specified, for example:

```
[modules]
http-hdr = pdwpi-httphdr-module
```

To specify multiple HTTP headers, multiple instances of the HTTP header module must be configured.

For example:

```
[modules]
entrust-client-header = pdwpi-httphdr-module
some-other-header= pdwpi-httphdr-module

[entrust-client-header]
header = entrust-client

[some-other-header]
header = some-other
```

## Maintaining session state using IP addresses

IBM Security Access Manager Plug-in for Web Servers can use IP addresses to identify and track sessions.

To configure the plug-in to use IP addresses to track sessions, use the **[common-modules]** stanza in the `pdwebpi.conf`. Enter the parameter **session** with the value as *ip-addr*. That is:

```
[common-modules]
session = ip-addr
```

Ensure the shared library for IP address authentication is configured in the **[modules]** stanza of the `pdwebpi.conf` configuration file. That is:

```
[modules]
ip-addr = pdwpi-ipaddr-module
```

If IP addresses are used to maintain session state they must also be used to authenticate incoming requests. See "Configuring IP address authentication" on page 98 for details on configuring IBM Security Access Manager Plug-in for Web

Servers to use the IP address as the method for client authentication. The usage of IP addresses for authenticating clients, however, does not require them to be used as the method for identifying sessions.

## Maintaining session state using LTPA cookies

LTPA authentication can be used to accept and authenticate based on an LTPA cookie. LTPA Authentication can maintain session state using the LTPA cookie found within each HTTP request.

To configure the plug-in to maintain session state using LTPA Authentication, use the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. Enter the parameter **session** with the value as *ltpa*, as in the following:

```
[common-modules]
session = ltpa
```

If LTPA is used to maintain session state, it also needs to be configured for user authentication. The **[common-modules]** stanza of the configuration file also should set to LTPA for authentication.

```
[common-modules]
authentication = ltpa
session = ltpa
```

## Maintaining session state using iv-headers

IBM Security Access Manager Plug-in for Web Servers can cache iv-header information to improve system performance.

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all session, authentication, and post-authorization methods using the format *module_type* = *module-name*. To cache iv-headers information, assign the value **iv-headers** to the **session** parameter as in the following:

```
[common-modules]
session = iv-headers
```

Ensure the shared library for iv-headers is configured in the **[modules]** stanza of the `pdwebpi.conf` configuration file. That is:

```
[modules]
iv-headers = pdwpi-iv-headers-module
```

# Chapter 5. Security policy

This chapter contains information that describes how you can configure and customize IBM Security Access Manager (Security Access Manager) Plug-in for Web Servers security policy.

This chapter includes the following topics:



## Plug-in-specific Access Control List (ACL) policies

The following security considerations apply for the `/PDWebPI` container in the protected object space:

- The IBM Security Access Manager Plug-in for Web Servers object begins the chain of ACL inheritance for the plug-in 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 this object and any object below this point.

See the *IBM Security Access Manager for Web: Base Administration Guide* for complete information about Security Access Manager ACL policies.

**Note:** The Microsoft IIS Web server provides the ability to specify a default Web page within a directory that is displayed when a user request contains a URL that includes only the directory path.

The ACL check performed by the Plug-in for Web Servers only applies to the directory specified in the request URL and not the default Web page served by the IIS server in response to this request.

You should incorporate this ACL check limitation when implementing your security policy on IIS platforms.

Similarly, due to the nature of the Web Server plug-in architecture, there is nothing stopping you from installing other modules which conflict with the security provided by the plug-in. It is the responsibility of the Web server administrator to ensure that no modules are installed onto the Web server that conflict with the plug-in.

For example the "MultiViews" functionality within Apache and IHS Web servers attempts to dynamically determine the extension of the requested URL. For example, if a request is made to `www.ibm.com/index` then the Web server would dynamically map this to `www.ibm.com/index.html` if such a file existed.

Unfortunately, this mapping occurs after authorization has taken place, which means the authorization check is performed on index rather than `index.html`.

In such situations, disable the "MultiViews" option or set up the policy to capture this mapping. For example, an ACL could be attached to `/PDWebPI/www.ibm.com`, or if greater granularity was required, the ACL could be attached to both `/PDWebPI/www.ibm.com/index` and `/PDWebPI/www.ibm.com/index.html`.

## /PDWebPI/*host* or *virtual_host*

The /PDWebPI/***host*** or ***virtual_host*** subtree contains the object space of a particular plug-in 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.

## Plug-in ACL permissions

The following table describes the ACL permissions applicable for the IBM Security Access Manager Plug-in for Web Servers region of the object space:

*Table 21. Plug-in ACL permissions*

| Permission | Operation | Description |
|---|---|---|
| [PDWebPI]**R** | read | View any element other than a directory. Any HTTP GET or POST request requires this permission. There is no specific "list" permission for requesting a directory listing (A GET of a URL ending in /). |
| [PDWebPI]**d** | delete | Remove the Web object from the Web space. HTTP DELETE commands require this permission. |
| [PDWebPI]**M** | modify | Place/publish a HTTP object in the plug-in object space. A HTTP PUT request requires this permission. |
| [PDWebPI]**p** | proxy | Determines whether a user can act as a Multiplexing Proxy agent. See “Add the MPA account to the pdwebpi-mpa-servers group” on page 113 for more details. |
| **T** | traverse | Required for access to any object below this point. |

The plug-in also supports WebDAV operations as shown below.

*Table 22. Plug-in WebDAV permissions*

| Task | Permission Required |
|---|---|
| PROPFIND | [PDWebPI]**R** |
| PROPPATCH | [PDWebPI]**M** |
| MKCOL | [PDWebPI]**N** |

WebDAV operations are authorized based on the request URI – not on individual members of a collection. In addition, some other WebDAV operations are partially supported:

- **COPY** - Requires [PDWebPI]**R** on the collection so that the 'copy from' can be read. Permissions for the destination are not checked.
- **MOVE** - This is considered a copy then a delete. Requires [PDWebPI]**Rd** on the collection that you are moving from. Permissions for the destination are not checked.

## Default /PDWebPI ACL policy

Core entries for the IBM Security Access Manager Plug-in for Web Servers ACL, **default-pdwebpi**, include:

*Table 23. Core entries for* **default-pdwebpi**

| | |
|---|---|
| Group iv-admin | TcmdbsvaBR[PDWebPI]rR |
| User sec_master | cmdbsvaBR[PDWebPI]rR |
| Any-other | T[PDWebPI]rR |
| Unauthenticated | T |
| Group pdwebpi-mpa-servers | TBR[PDWebPI]p |
| Group webseal-servers | TBR[PDWebPI]p |
| Group webseal-mpa-servers | TBR[PDWebPI]p |

At installation, this default ACL is attached to the /PDWebPI container object in the object space.

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.

## Changing The Mapping of HTTP Request Methods

You can change the mapping of HTTP request methods to permission strings by changing the values specified in the **[http-method-perms]** stanza of the configuration file. For example, to define that an HTTP method PUSH should map to the permission string [PDWebPI]w, add the following entry to the **[http-method-perms]** stanza:

```
PUSH = [PDWebPI]w
```

For more information, see “[http-method-perms]” on page 253.

**Note:** The configuration specified in the **[http-method-perms]** stanza is validated each time the authorization server starts. This ensures that the permission strings specified correspond to actions already defined in the policy database.

The validation process may result in authorization audit events being generated. Such audit events will appear as unauthenticated accesses to the Security Access Manager policy object **/Permission-Configuration**.

If you wish to suppress these audit events, attach a protected object policy (POP) with its audit-level attribute set to *none* to an object with the name **/Permission-Configuration** (this object is not created in the object space by default). The following **pdadmin** commands illustrate the creation of policy that will suppress the generation of these audit events:

```
pdadmin> objectspace create /Permission-Configuration
"Permission configuration validation" 0
pdadmin> pop create permission-configuration
pdadmin> pop modify permission-configuration set audit-level none
pdadmin> pop attach /Permission-Configuration permission-configuration
```

## Setting a logon failure policy

This section describes various settings which allow you to enforce a policy to control repeated login failures. Such a logon policy can help prevent random computer-generated logon attempts, which may occur many times a second.

The logon failure policy, available for LDAP-based Security Access Manager installations, enables you to prevent computer password attacks by specifying:

- A maximum number of failed log on attempts
- A penalty lockout time (how long a user must wait before making further log on attempts)
- Whether an error page is displayed to the user immediately following the maximum number of failed login attempts, or at the following login attempt
- Whether the user's account is disabled for the penalty lockout time only, or whether an administrator reset is required

These settings are designed to provide maximum flexibility in setting a policy that suits your corporate needs. For example, a policy could dictate three failed attempts, followed by a 180 second penalty. An error page could be displayed to the user immediately after their third failed attempt, or else a subsequent attempt (correct or incorrect) could result in the error page being displayed.

The user could be allowed to log in again after the specified penalty period has expired, or else an administrator could be required to reset the user's account before they can successfully log in again.

The standard logon failure policy (sometimes called the "three strikes policy", although the set limit need not be three) requires the joint contribution of two **pdadmin** policy command settings:

- Maximum number of failed log on attempts
  **policy set max-login-failures**
- Penalty for exceeding failed log on attempt setting
  **policy set disable-time-interval**
  The penalty setting can include an account lockout time interval or a complete disabling of the account. The time interval is specified in seconds (the typical minimum 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.

The Web Plug-in returns a server response error page (`acct_locked.html`) that notifies the user of the penalty. The **late-lockout-notification** stanza entry in the **[pdweb-plugins]** stanza of the plug-in configuration file specifies whether this

notification occurs when the user reaches the **max-login-failures** limit, or at the next login attempt after reaching that limit.

For new installations of the Web Plug-in, the default **late-lockout-notification** setting is `no`. Upon reaching the maximum value set by the **max-login-failures** policy, the plug-in immediately sends the account disabled error page to the user. For example:

```
[pdweb-plugins]
late-lockout-notification = no
```

For migrated (upgraded) installations of the Web Plug-in, the default **late-lockout-notification** setting is `yes`. Upon reaching the maximum value set by the **max-login-failures** policy, the plug-in returns another login prompt to the user. The plug-in does not send the account disabled error page to the user until the next login attempt (this represents the pre-version 6.0 behavior for the **max-login-failures** policy. For example:

```
[pdweb-plugins]
late-lockout-notification = yes
```

Finally, a **terminate-on-reath-lockout** setting in the **[pdweb-plugins]** stanza controls whether or not user sessions are terminated after the **max-login-failures** limit has been reached. This setting is enabled by default. For example:

```
[pdweb-plugins]
terminate-on-reath-lockout = yes
```

The **terminate-on-reath-lockout** setting can be overridden on a per-virtual host basis by specifying it in the **[virtual-host-name]** stanza.

**Note:**

1. Setting the **disable-time-interval** to `disable` results in additional administration overhead. You might observe delays in replicating **account valid** information to the plug-in.

   This situation depends on your LDAP environment. In addition, certain LDAP implementations might experience performance degradation as a result of the **account valid** update operation. For these reasons it is typical to use a timeout interval.
2. When multiple plug-in servers are used in a load-balancing architecture, the results of the policy are affected by the fact that each plug-in 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 redirects the request to another server. Now the client has three more opportunities to attempt a successful login. If your business security solution requires a three strikes login policy, understand the implications of setting this policy in load-balanced architectures.
3. The following **pdadmin** commands are appropriate only for use with an LDAP registry.

*Table 24. pdadmin LDAP logon policy commands*

| Command | Description |
|---|---|
| **policy set max-login-failures** {*number* \| **unset**} **[-user** *username***]** | |
| **policy get max-login-failures [-user** *username***]** | |

*Table 24. pdadmin LDAP logon policy commands (continued)*

| Command | Description |
|---|---|
| | Manages the policy controlling the maximum number of failed log on attempts allowed before a penalty is imposed. This command depends on a penalty set in the policy set disable-time-interval command.<br><br>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.<br><br>The default setting is 10 attempts. |
| **policy set disable-time-interval** {*number* \| **unset** \| **disable**} **[-user** *username***]** | |
| **policy get disable-time-interval [-user** *username***]** | |
| | Manages the penalty policy controlling the time period an account should be disabled if the maximum number of failed log on attempts is reached.<br><br>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.<br><br>The default setting is 180 seconds. |

# Password strength policy

A Security Access Manager LDAP-based installation provides two means of controlling the construction of passwords:

- Five **pdadmin** password policy commands
- A pluggable authentication module (PAM) that allows you to customize a password policy

  See the*IBM Security Access Manager for Web: Authorization C API Developer's 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.

The following **pdadmin** commands are appropriate for use only with an LDAP registry. The unset option disables this policy attribute – that is, the policy is not enforced.

*Table 25. pdadmin LDAP password strength commands*

| Command | Description |
|---|---|
| **policy set min-password-length** {*number* \| **unset**} **[-user** *username***]** | |
| **policy get min-password-length [-user** *username***]** | |
| | Manages the policy controlling the minimum length of a password.<br><br>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.<br><br>The default setting is 8. |
| **policy set min-password-alphas** {*number* \| **unset**} **[-user** *username***]** | |
| **policy get min-password-alphas [-user** *username***]** | |
| | Manages the policy controlling the minimum number of alphabetic characters allowed in a password.<br><br>As the administrator, you can apply this penalty policy to a specific user or apply the policy globally to all users listed in the default registry.<br><br>The default setting is 4. |
| **policy set min-password-non-alphas** {*number* \| **unset**} **[-user** *username***]** | |
| **policy get min-password-non-alphas [-user** *username***]** | |
| | Manages the policy controlling minimum number of non-alphabetic (numeric) characters allowed in a password.<br><br>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.<br><br>The default setting is 1. |
| **policy set max-password-repeated-chars** {*number* \| **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.<br><br>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.<br><br>The default setting is 2. |
| **policy set password-spaces {yes \| no \| unset} [-user** *username***]** | |
| **policy get password-spaces [-user** *username***]** | |
| | Manages the policy controlling whether a password can contain spaces.<br><br>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.<br><br>The default setting is unset. |

The following table illustrates several password examples and the policy results based on the default values of the five **pdadmin** parameters:

*Table 26. Password examples*

| Example | Result |
|---|---|
| **password** | Not valid: must contain at least one non-alphabetic character. |
| **pass** | Not valid: must contain at least 8 characters. |
| **passs1234** | Not valid: contains more than two repeated characters. |
| **12345678** | Not valid: must contain at least four alphabetic characters. |
| **password3** | Valid. |

## 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 Protected Object Policy (Step-up)

The authentication-strength Protected Object Policy (POP) makes it possible to control access to objects based on the authentication method that they use.

You can use this functionality – sometimes known as step-up authentication – to ensure that users accessing more sensitive resources use a stronger authentication mechanism. You might want this condition because of the greater threat of improper access.

For example, you can provide greater security to a region of the Web space by applying a step-up POP policy that requires a stronger level of authentication than the client used when initially entering the plug-in domain.

Step-up authentication can also be set for each specific virtual host on a Web server, allowing individual virtual hosts to carry their own step-up levels of authentication without being subject to server-wide policy implementations.

Authentication strength policy is set in the IP Endpoint Authentication Method attribute of a POP policy.

**Note:** Step-up to client certificates is currently unsupported on IBM HTTP server, version 8.0.

## Configuring levels for step-up authentication

The first step in configuring authentication-specific access is to configure the supported authentication methods and determine the order in which these authentication methods should be considered stronger. See Chapter 3, "Authentication and request processing," on page 49 for details on configuring authentication mechanisms.

Any client accessing a Web server through the plug-in has an authentication level, such as "unauthenticated" or "password," which indicates the method by which the client last authenticated through the plug-in.

In some situations it might be necessary to enforce minimum "safe" levels of authentication required to access certain Web space objects. For example, in one environment, authentication by token passcode might be considered more secure than authentication by username and password. Another environment could have different standards.

Rather than forcing clients to restart their sessions when they do not meet the required level of authentication, the step-up authentication mechanism gives clients a second chance to re-authenticate using the required method (level).

Step-up authentication means that users are not immediately shown a "denied" message when they try to access a resource that requires a "higher" authentication level than the one they logged on with. Instead, they are presented with a new authentication prompt that requests information to support the higher authentication level. If they are able to supply this level of authentication, then their original request will be permitted.

You configure authentication levels in the **[authentication-levels]** or **[authentication-levels:***virtual_host_label***]** stanza of the `pdwebpi.conf` configuration file. For example:

```
[authentication-levels]
1 = BA
2 = iv-headers
3 = cert
```

Based on the order of the methods in the list, each method is assigned a level index.

- Unauthenticated is assumed to have a level of 0.

- Subsequent methods can be placed in any order. See "Step-up authentication notes and limitations" on page 139
- There must be at least two entries to enable step-up authentication.
- Levels for authentication mechanisms can be set for specific virtual hosts by specifying the levels using a stanza with the form: **[authentication-levels:***virtual_host_name***]**.

**Note:** See Chapter 3, "Authentication and request processing," on page 49 for detailed information about setting up the required authentication mechanisms.

## Enabling step-up authentication

Step-up authentication is implemented using a POP policy placed on the objects requiring authentication sensitive authorization. You use the IP Endpoint Authentication Method attribute of a POP policy.

The **pdadmin pop modify set ipauth** command specifies both the allowed networks and the required authentication level in the **IP Endpoint Authentication Method** attribute.

The configured authentication levels can be linked to IP address ranges. This method is intended to provide management flexibility. If filtering users by IP address is not important, you can set a single entry for **anyothernw** (any other network). This setting will affect all accessing users, regardless of IP address, and require them to be authenticated at the specified level. This is the most common method for implementing step-up authentication.

Syntax:

```
pdadmin> pop modify pop_name set ipauth anyothernw level_index
```

The **anyothernw** entry is used as a network range that will match any network not otherwise specified in the POP. This method is used to create a default entry which could either deny all unmatched IP addresses or allow anyone access who can meet the authentication level requirement.

By default, **anyothernw** appears in a POP with an authentication level index of 0. The entry appears as "Any Other Network" in the **pop show** command:

```
pdadmin> 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 0
```

During step-up authentication, validation of the supplied user ID can be enabled by setting the **verify-step-up-user** parameter in the **[module-mgr]** stanza to *true*.

```
[module-mgr]
verify-step-up-user = true
```

Enabling the **verify-step-up-user** parameter ensure that when prompted to re-authenticate using a higher level mechanism, the entered identity matches the

identity originally entered. If the identities don't match, a generic authorization server error page is returned. This page can be customized if so desired.

### Step-up authentication example

To configure authentication:

1. Configure authentication levels in `pdwebpi.conf`:

   ```
   [authentication-levels] or [authentication-levels:virtual_host_label]
   1 = BA
   2 = token
   ```

2. Configure IP Endpoint Authentication Method POP attribute:

   ```
   pdadmin> pop modify test set ipauth anyothernw 2
   pdadmin> pop show test
          Protected object policy:test
          Description: Test  POP
          Warning: no
          Audit level: none
          Quality of protection: none
          Time of day access: mon, wed, fri:anytime:local
          IP Endpoint Authentication Method Policy
              Any Other Network 2
   ```

   Users accessing objects that are protected by the test POP require level 2 authentication, or the users will be forced to authenticate with the token method.

   See also "Network-based authentication Protected Object Policy" on page 142.

## Step-up authentication notes and limitations

Note that:

- Step-up authentication is supported over both HTTP and HTTPS.
- You cannot step-up from the HTTP protocol to HTTPS.
- Authentication methods not specified in the **[authentication-levels]** stanza default to level 1.
- Authentication methods can be specified only once in the level list.
- SPNEGO does not step-up to any authentication method that uses POST forms. Configuring step-up behavior using SPNEGO authentication module results in an error page being returned to the client.
- Incorrect configuration of step-up authentication levels results in the disabling of step-up functionality within the plug-in. This situation can lead to unexpected authentication behavior, such as the password logon page being issued for objects protected by a POP that requires the token passcode authentication method.

  After configuring step-up authentication mechanisms, check the `pdwebpi.log` file for reports of any configuration errors.

# Multi-factor authentication

Multi-factor authentication functionality is an extension of step-up authentication functionality that allows you to specify a protected object policy (POP) that forces the user to authenticate using *all* authentication mechanisms with a level lower than the configured POP authentication level. That is, the user is required to have authenticated at all levels up to, and including, the required level before access is granted. Multi-factor authentication can also be used in conjunction with re-authentication to force a multi-factor re-authentication.

Standard authentication-level based authentication allows a policy to be associated with an object that sets a minimum required authentication level that must be achieved before access is granted. The supported authentication mechanisms are given an ordering in the configuration that specifies which mechanisms are considered stronger than others.

When a user first authenticates in order to access an object, they are offered the choice of all authentication methods that meet the required level for that object. It is up to the user to choose which method they will use.

To achieve multi-factor authentication, step-up authentication needs to be configured as discussed in "Authentication-strength Protected Object Policy (Step-up)" on page 136. Once step-up authentication is configured, you will need to add the extended attribute, MULTI-FACTOR-AUTH, to a protected object policy (POP) for a IBM Security Access Manager Plug-in for Web Servers object or objects.

When the MULTI-FACTOR-AUTH attribute is set, all authentication levels up to the specified POP authentication level are required before access to the resource is granted.

As an example, assume the following configuration is set in the configuration file:

```
[authentication-levels]
1 = cert
2 = forms
```

With the above configuration, when a POP attached to a resource which requires an authentication level of 2 and the new MULTI-FACTOR-AUTH attribute is set to *true*, the user must first supply a valid client certificate before entering a forms-based logon. If the POP attached to the resource does not have the MULTI-FACTOR-AUTH attribute enabled, then only form-based authentication is used.

### Enabling multi-factor authentication

Multi-factor authentication is implemented using a POP policy placed on the objects requiring multi-factor authentication.

Syntax:

```
pdadmin> pop modify pop_name set attribute MULTI-FACTOR_AUTH true
```

## Reauthentication Protected Object Policy

IBM Security Access Manager Plug-in for Web Servers can force a user to perform an additional log on (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 session cache inactivity timeout value. This section discusses reauthentication based on security policy as dictated by a POP extended attribute. Refer to "Configuring the plug-in session/credentials cache" on page 121 for details on configuring the Session/Credential Cache.

## Conditions affecting POP reauthentication

Forced reauthentication provides more protection for sensitive resources in the secure domain. Reauthentication that is 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 plug-in authentication methods:

- Forms (user name and password) authentication
- Token authentication

In addition, a custom user name/password external authentication mechanism can be written to support reauthentication.

Reauthentication assumes that the user initially logged in to the secure domain and that a valid credential exists for the user. During reauthentication, the user must log on using the same identity that generated the existing credential. Security Access Manager preserves the user's original session information, including the credential, during reauthentication. The credential is not replaced during reauthentication.

During reauthentication, the plug-in also caches the request that prompted the reauthentication. Upon successful reauthentication, the cached data is used to rebuild the request.

If reauthentication fails, the plug-in returns the logon prompt again. If reauthentication succeeds, but the ACL check fails for that resource, a 403 "Forbidden" message is returned. 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 end the reauthentication process (by requesting another URL) and still participate in the secure domain by accessing other resources that do not require reauthentication.

## 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:

```
pdadmin>pop create secure
pdadmin>pop modify secure set attribute REAUTH true
pdadmin>pop attach /PDWebPI/hostA/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. Reauthentication is required for every access to objects protected by a reauthentication policy.

In situations when most but not all objects in a directory require reauthentication, it is best to attach a POP to the entire directory including the "reauth" extended attribute. For those objects that do not require reauthentication, attach a POP that is identical to that for the directory but exclude the "reauth" extended attribute.

Details about the **pdadmin** command line utility can be found in the *IBM Security Access Manager for Web: Base Administration Guide*.

## Network-based authentication Protected Object Policy

The network-based authentication Protected Object Policy (POP) makes it possible to control access to objects based on the IP address of the user. You can use this functionality to prevent specific IP addresses (or IP address ranges) from accessing any resources in your secure domain.

You can also apply step-up authentication configuration to this policy and require a specific authentication method for each specified IP address range.

Network-based authentication policy is set in the IP Endpoint Authentication Method attribute of a POP policy. You must specify two requirements in this attribute:

- Authentication levels
- Allowed networks

For details on specifying configuration levels, refer to "Configuring levels for step-up authentication" on page 137.

### Specifying IP addresses and ranges

After configuring the authentication levels, you must specify the IP addresses and IP address ranges permitted by this POP policy.

The **pdadmin pop modify set ipauth add** command specifies both the network (or network range) and the required authentication level in the IP Endpoint Authentication Method attribute.

Syntax:

```
pdadmin> pop modify pop_name set ipauth add network netmask level_index
```

The configured authentication levels are linked to IP address ranges. This method is intended to provide flexibility. If filtering users by IP address is not important, you can set a single entry for **anyothernw** (any other network). This setting affects all accessing users, regardless of IP address, and requires them to authenticate at the specified level.

Syntax:

```
pdadmin> pop modify pop_name set ipauth anyothernw level_index
```

Conversely, if you want to ignore the authentication level and want to allow or deny access based only on IP address, you can use level 0 for ranges that you want to allow in and "forbidden" for ranges you want to deny.

The **anyothernw** entry is used as a network range that matches any network not otherwise specified in the POP. This method can be used to create a default entry that could either deny all unmatched IP addresses or allow access to anyone who meets the authentication level requirement.

By default, **anyothernw** appears in a POP with an authentication level index of 0. The entry appears as "Any Other Network" in the **pop show** command:

```
pdadmin> 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 0
```

See “Configuring levels for step-up authentication” on page 137 for a more detailed discussion on setting authentication levels.

### Example

Require users from IP address range 9.0.0.0 and netmask 255.0.0.0 to use level 1 authentication ("password" by default):

```
pdadmin> pop modify test set ipauth add 9.0.0.0 255.0.0.0 1
```

Require a specific user to use level 0 authentication:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 0
```

Prevent all users (other than those specified as in the examples above) from accessing the object:

```
pdadmin> pop modify test set ipauth anyothernw forbidden
```

## Disabling step-up authentication by IP address

To disable step-up authentication by IP address, enter the following command:

```
pdadmin> pop modify pop_name set ipauth remove network netmask
```

For example:

```
pdadmin> pop modify test set ipauth remove 9.0.0.0 255.0.0.0
```

## Network-based authentication algorithm

IBM Security Access Manager Plug-in for Web Servers 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.

## Quality-of-protection Protected Object Policy

The quality-of-protection Protected Object Policy (POP) attribute allows you to specify what level of data protection is required when performing an operation on an object.

```
pdadmin> pop modify pop_name set qop {none|integrity|privacy}
```

*Table 27. QOP level descriptions*

| QOP level | Description |
|---|---|
| privacy | Data encryption is required (SSL). |
| integrity | Use some mechanism to ensure that the data has not changed. |
| none | No method of data protection is used. |

For example:

```
pdadmin> pop modify test set qop privacy
```

The quality-of-protection POP attribute permits a single transaction when the "yes" response to the ACL decision also includes the required quality-of-protection level. If the plug-in cannot guarantee the required level of protection, the request is denied.

## Handling unauthenticated users (HTTP/HTTPS)

IBM Security Access Manager Plug-in for Web Servers accepts requests from both authenticated and unauthenticated users over HTTP and HTTPS. The plug-in then relies on the Authorization Server to enforce security policy by permitting or denying access to protected resources.

The following conditions apply to unauthenticated users who access over SSL:

- The exchange of information between the unauthenticated user and the plug-in is encrypted – as it is with an authenticated user.
- An SSL connection between an unauthenticated user and the plug-in requires only server-side authentication.

### Processing a request from an anonymous client

Steps to process a request from an anonymous client include:

1. An anonymous client makes a request to the Web server through the plug-in (using HTTP or HTTPS).
2. The plug-in creates an unauthenticated credential for this client.
3. The request proceeds, with this credential, to the protected Web object.
4. The Authorization Server 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) permission.
6. If the request fails the authorization decision, the client receives a logon form (BA or Forms-based).

## Forcing user log on

You can force an unauthenticated user to log on by correctly setting the appropriate permissions on the unauthenticated entry in the ACL policy that protects the requested object.

The read [PDWebPI]**r** permission allows unauthenticated access to an object.

To force an unauthenticated user to log on, remove the read [PDWebPI]**r** permission from the unauthenticated entry in the ACL policy that protects the object.

## Applying unauthenticated HTTPS

There are many practical business reasons for supporting unauthenticated access to the plug-in enhanced Web server over HTTPS. These include:

- Some applications do not require a personal log on, 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.

## Controlling unauthenticated users with ACL/POP policies

To control unauthenticated users with ACL/POP policies:

1. To permit unauthenticated user access to public objects, protect the public content with an ACL that contains at least the read [PDWebPI]**r** permission for the unauthenticated and any-other entries:

   ```
   unauthenticated [PDWebPI]r
   any-other [PDWebPI]r
   ```

   **Note:**

   - The "any-other" entry type is also known as the "any-authenticated" entry type.
   - The **unauthenticated** entry is a mask (a bitwise "and" operation) against the **any-other** entry when permissions are determined. A permission for **unauthenticated** is granted only if the permission also appears in the **any-other** entry. Since **unauthenticated** depends on **any-other**, it makes little sense for an ACL to contain **unauthenticated** without **any-other**. If an ACL does contain **unauthenticated** without **any-other**, the default response is to grant no permissions to **unauthenticated**.
2. To require encryption (SSL), protect the content with a Protected Object Policy (POP) that specifies privacy as a condition. See “Quality-of-protection Protected Object Policy” on page 144.

# Policy for unprotected resources

The plug-in allows you to specify the resources in your network that do not require an authorization decision; that is, resources that can be accessed by both authenticated and unauthenticated users.

When identified through special policy, these resources are added to a plug-in cache called the Unprotected Resource Cache (URC). Requests for cached unprotected resources can then be sent directly to the Web server for processing.

The processing bypasses the authorization server and all of the following: authorization decisions, auditing, Web log module processing, dynamic URL module processing, CDSSO or eCSSO processing, tag value processing, and external authentication interface URI processing.

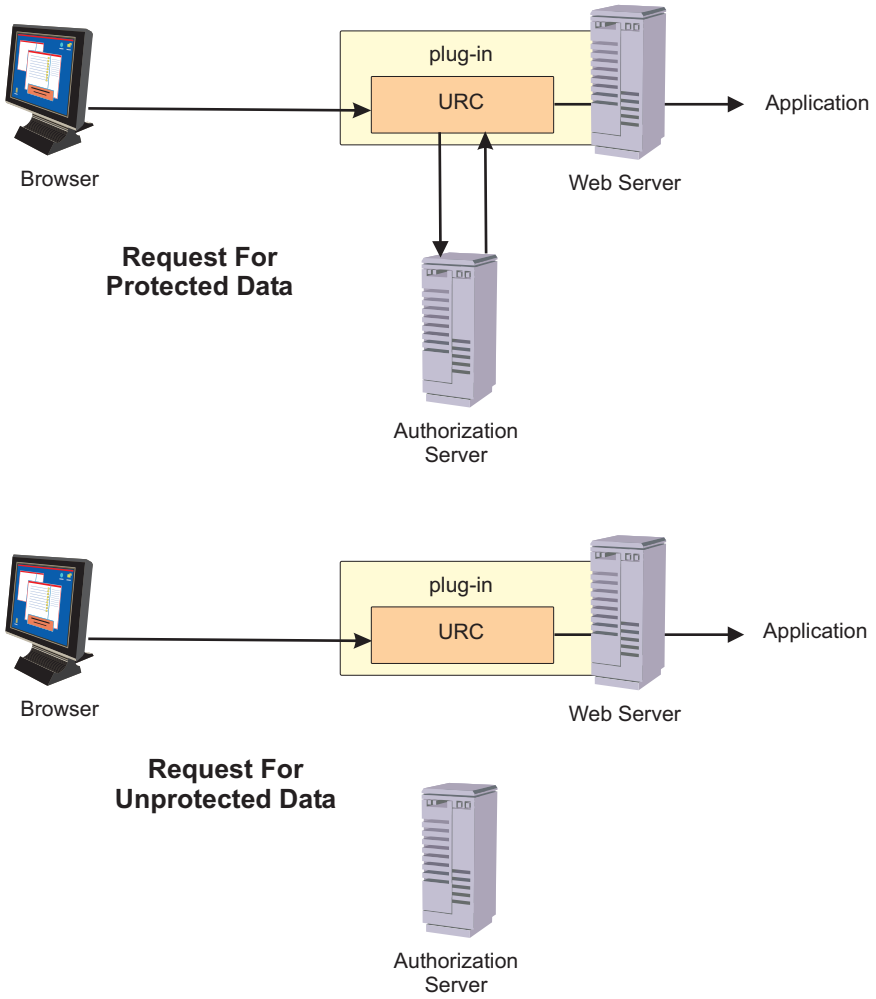


*Figure 8. Bypassing the authentication server for unprotected resources*

Depending on the level of requests for unprotected resources, a correctly configured unprotected resource cache can markedly improve system performance by avoiding the inter-process communication between the plug-in and the authorization server. Unprotected resources are identified using Protected Object Policy (POP).

Security Access Manager namespace inheritance rules apply to the objects defined as unprotected resources. When an unprotected POP is attached to a protected object in the Security Access Manager namespace, that object and all its subordinates inherit the POP policy and become part of the unprotected resource cache.

In the figure below, the unprotected POPs placed on **/docs** as shown will cause those objects and all subordinate objects to become part of the unprotected resource cache. When an unprotected resource POP is removed from the object **private** (POP2), that object and any subordinates then cease to be part of the unprotected resource cache.

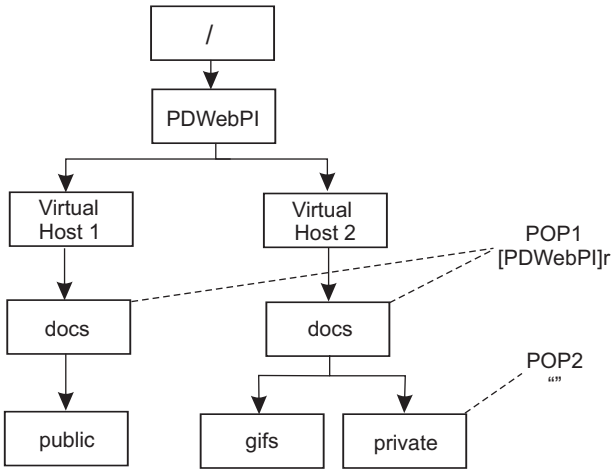


*Figure 9. Unprotected resource cache POP inheritance.*

The authorization server is bypassed for those objects in the unprotected resource cache. As such, care should be taken when applying unprotected policy to objects to ensure that you do not inadvertently permit unrestricted access to objects that would otherwise require authorization.

The format of your object space should be considered and the Security Access Manager namespace inheritance rules fully understood before applying policy for unprotected objects.

**Note:** Objects configured for inclusion in the unprotected resource cache cannot be used in conjunction with pre-authorization URL mapping functions such as the plug-in's dynamic URL mapping functionality.

## Configuring the unprotected resource cache

Configuration entries for the unprotected resource cache are located in the **[unprotected-resource-cache]** stanza of the plug-in `pdwebpi.conf` configuration file.

Use the **enabled** entry to enable or disable the unprotected resource cache. A value of *yes* (or *true*) enables the cache, a value of *no* (or *false*) disables it.

The plug-in polls the authorization server to determine whether its current version of the unprotected resource cache is out of date. The maximum length of time between each poll of the authorization server is set using the entry, `max-poll-interval`. The `max-poll-interval` property is set at 30 seconds by default.

## Setting the unprotected resource cache extended POP attribute

Unprotected resources are identified for inclusion in the unprotected resource cache by attaching POP objects to the required resources within the Security Access Manager namespace. Each of the POP objects needs to include a permission set as an extended attribute. The command syntax for specifying an object for inclusion in the unprotected resource cache is:

```
pdadmin> pop modify POP_name set attribute UNPROTECTED permission_set
```

The command parameter, *permission_set*, identifies the resources, controlled by the POP and with the specified permissions, that are to be considered unprotected. For example; consider the object space in Figure 9 on page 147. The following commands would create the POPs shown in the example and determine the objects that are to be considered as unprotected.

```
pdadmin> pop attach /PDWebPI/virtual_host_1/docs URC-POP
pdadmin> pop attach /PDWebPI/virtual_host_2/docs URC-POP
pdadmin> pop modify URC-POP set attribute UNPROTECTED [PDWebPI]rw
pdadmin> pop attach /PDWebPI/virtual_host_2/docs/private URC-POP-private
pdadmin> pop modify URC-POP-private set attribute UNPROTECTED ""
```

The above commands attach two objects to the POP, **URC-POP**. The POP is then modified so that all resources controlled by **URC-POP**, with the permissions [PDWebPI]rw, are considered unprotected. The wildcard character, '*', can be used in place of a permission set to specify all resources controlled by the specified POP.

In the example commands, another POP is created, **URC-POP-private**. This POP removes the object, `/private`, from the unprotected resource cache. An empty permission string are used to signify that objects controlled by the specified POP are not included in the unprotected resource cache.

**Note:** A trade-off of using the unprotected resource cache is the native Web server log files and audit trails will not contain user identity information or identify the transaction as unauthenticated.

# Chapter 6. Web single sign-on solutions

When IBM Security Access Manager Plug-in for Web Servers is implemented as an authorization service to provide protection to a secure domain, there is often a requirement to provide solutions for single sign-on to resources within that domain. This chapter discusses single sign-on solutions for the Web space protected by IBM Security Access Manager Plug-in for Web Servers.

This chapter includes the following topics:



## Single sign-on concepts

When a protected resource is located on the plug-in enhanced Web application server, a client requesting that resource can be required to perform multiple logons when accessing different secure applications. Each logon is likely to require different logon identities.

The problem of administering and maintaining multiple logon identities can often be solved with a single sign-on (SSO) mechanism. SSO allows the user to access a resource using only one initial logon. Any further logon requirements for resources on the Web server are handled transparent to the user.

There are a number of different single sign-on architectures supported by IBM Security Access Manager Plug-in for Web Servers. These are:

1. One plug-in instance providing single sign-on to more than one secure application on a server.
2. Single sign-on to the plug-in from WebSEAL or other proxy agent such as a WAP gateway.
3. Use of failover cookies to provide single sign-on between different domains.
4. Use of the Global Single sign-on (GSO) Lockbox module to provide access to applications using stored user credential information.
5. Use of Security Provider NEGOtiation (SPNEGO) for permitting access to resources on IIS based Web servers.
6. Forms based authentication as a mechanism for SSO.
7. Cross Domain Single Sign On which provides a mechanism for transferring user credentials across multiple secure domains.
8. e-Community single sign-on, where a user authenticates once and is issued a token that allows them to access other domains within a virtual community of domains without the need to re-authenticate.

The first six SSO scenarios are discussed in this chapter. The seventh and eighth scenarios are the topic of the next chapter.

# Automatically signing-on to a secured application

HTTP headers and LTPA cookies (when the application is WebSphere Application Server) can be used to achieve SSO to applications on a server that are protected by one plug-in instance.

After initial authentication of the client, the plug-in can build an HTTP header containing client identity information that can be used for automatic authentication to secure applications running on the server. In a similar way, an LTPA cookie can be used to achieve SSO to a Web application server such as WebSphere.

## Configuring single sign-on to secure applications using HTTP headers

The HTTP headers used for signing on to an application are generated by the iv-header post-authorization module. The set of headers that can be generated are collectively called IV headers.

After the successful authorization of a user request, the plug-in can insert IV headers that define the client's identity into the request for processing by the application. This header information can be used as proof of the user's identity when the request is handled by an application hosted by the secured Web server. The user is spared the need to log on each time a new secure application is accessed.

Configured for post-authorization processing, IV headers are inserted with one, some, or all of the iv-user, iv-user-l, iv-creds, iv-groups, iv-remote-address HTTP header types. These header types are described in the following table.

*Table 28. IV header field descriptions*

| IV Header Field | Description |
|---|---|
| **iv-user** | The short name of the Security Access Manager user. Defaults to unauthenticated if the client is unauthenticated (unknown). |
| **iv-user-l** | The full domain name of the user (long form), for example, LDAP distinguished name. |
| **iv-groups** | A list of the groups to which the user belongs. |
| **iv-creds** | Encoded opaque data structure representing the user's Security Access Manager credential. |
| **iv-remote-address** | The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT). |

### Enabling and disabling generation of IV headers

To enable the plug-in to insert IV headers into authorized requests, the plug-in needs to be configured to use IV headers for post-authorization processing. The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable IV headers for post-authorization processing, assign the parameter **post-authzn** the keyword value *iv-headers* in the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. That is:

```
[common-modules]
post-authzn = iv-headers
```

### Configuring IV header parameters

IV header authentication parameters are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **generate** parameter specifies the type of IV headers to generate when forwarding proxied requests. By default, the plug-in generates all types of IV headers when forwarding proxied requests. The valid options are all, iv-creds, iv-user, iv-user-l, and iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
generate = iv-creds,iv-user,iv-user-1
```

## Single sign-on to WebSphere application server using LTPA cookies

When the plug-in is installed as a protective layer on a WebSphere application server, accessing clients are faced with two potential logon points – the plug-in and secure applications served by WebSphere. To provide a single point of logon in this situation the plug-in can be configured to generate and pass the cookie-based lightweight third party authentication (LTPA) mechanism onto Web application servers that support LTPA cookies.

When a user makes a request for a resource on a server, the user must first be authenticated to the plug-in. After successful authentication, the plug-in generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token for the Web application server, contains user identity and password information. This information is encrypted using a password-protected secret key shared between the plug-in and the application server.

The plug-in inserts the cookie in the HTTP header of the request that is sent to the Web application server. The application server receives the request, decrypts the cookie, and authenticates the user based on the identity information supplied in the cookie.

To improve performance, the plug-in stores the LTPA cookie in the session cache and uses the cached LTPA cookie for subsequent requests during the same user session. For details on setting the parameters for the session cache, see "Configuring the plug-in session/credentials cache" on page 121.

### Configuring single sign-on to WebSphere using LTPA cookies

Use of LTPA cookies to achieve single sign-on to application servers supporting LTPA cookies, is part of the plug-in's post-authorization processing. To enable this functionality, enter the key value **ltpa** for the parameter **post-authzn** in the **[common-modules]** stanza of the `pdwebpi.conf` configuration file:

```
[common-modules]
post-authzn = ltpa
```

LTPA cookie configuration is performed in the **[ltpa]** stanza of the `pdwebpi.conf` configuration file. The following configuration parameters are available.

*Table 29. LTPA configuration parameters*

| Parameter | Description |
|---|---|
| **ltpa-keyfile** | The full path name of the key file used to encrypt the identity information contained in the cookie. |
| **ltpa-stash-file** | The location of the password stash file. If no password stash file exists, this entry should be commented out. |
| **ltpa-password** | The password to use when a password stash file does not exist. |
| **ltpa-cookie-name** | The name of the cookie in which the LTPA token is stored. If no value is defined, the default value `LtpaToken` is used. |
| **ltpa-lifetime** | The lifetime, in seconds, of the LTPA cookie. |
| **ltpa-generate-unauth** | Determines whether LTPA cookies are created for unauthenticated users. LTPA cookies are not useful for unauthenticated users and may result in unexpected behavior. |

### Technical notes for LTPA single sign-on

For LTPA single sign-on, note that:

- The key file contains information about a specific Web application server. If you add more than one application server to the same plug-in, all servers will share the same key file.
- For single sign-on to succeed, the plug-in and the application server must in some way share the same registry information.
- The application server is responsible for setting up LTPA and the creation of the shared secret key.

## Single sign-on to the plug-in from WebSEAL or other proxy

When the plug-in enhanced Web server receives requests from a trusted application such as WebSEAL or a multiplexing proxy agent, IV headers may be inserted into the requests relayed to the plug-in. IV headers contain information that identify the originating client rather than the relaying server. The information in the headers is used to construct an originating client credential for authorization purposes.

If the plug-in is configured to use IV Headers to perform client authentication, the plug-in creates a client credential using the identity extracted from an IV header found in the transaction request. Because it is easy for clients to fake IV headers, such a credential is created only when the 'use secondary authenticator' flag in the authenticate request is set.

For authentication, IV headers can be configured to accept one, some, or all of iv-user, iv-user-l, iv-creds or iv-remote-address headers in the request as proof of authentication when received via a proxy. The iv-remote-address header is used to record the real remote address of the user. These IV header types are recognized by Security Access Manager and WebSEAL.

*Table 30. IV header field descriptions*

| IV Header Field | Description |
|---|---|
| **iv-user** | The short name of the client. Defaults to unauthenticated if the client is unauthenticated (unknown). |

*Table 30. IV header field descriptions (continued)*

| IV Header Field | Description |
|---|---|
| **iv-user-l** | The full domain name of the user (long form). |
| **iv-groups** | A list of the groups to which the client belongs. |
| **iv-creds** | Encoded opaque data structure representing a Security Access Manager credential. |
| **iv-remote-address** | The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT). |

**Note:** Access Manager only trusts headers received from trusted frontends. A frontend is considered trusted if it is recognized as a Multiplexing Proxy Agent (MPA). For details on configuring the plug-in for supporting MPAs refer to "Supporting Multiplexing Proxy Agents (MPA)" on page 110.

To be accepted as proof of client identity, WebSEAL or other proxy must itself be authenticated to the plug-in. This is typically achieved by a mutually authenticated SSL connection between the proxy and the Web server secured by the plug-in.

## Enabling and disabling authentication using IV headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using IV headers, assign the reference 'iv-header' to the **authentication** parameter:

```
[common-modules]
authentication = iv-header
```

## Configuring IV header parameters

IV header authentication parameters are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **accept** parameter specifies the types of IV header that are accepted for performing IV header authentication. By default the plug-in accepts all types of IV header. The valid options are all, iv-creds, iv-user, iv-user-l, and iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
accept = iv-creds,iv-user
```

# Using the Failover cookie for single sign-on

With failover cookies configured for post-authorization processing, the plug-in encrypts a client's credential data in either a server-specific or domain-wide cookie. The cookie is placed on the browser when the client first connects.

When the client attempts to access another secure server within the domain, the cookie is presented to the next server that the client is redirected to. The cookie is used for automatic re-authentication so the client is spared the task of

re-authenticating manually. The plug-ins on replicated servers share a common key that decrypts the credential information held in the cookie, establishing a new session.

For further details on configuring failover cookie authentication refer to "Configuring failover authentication" on page 81.

### Enabling single sign-on using Failover cookies

Failover cookies can be configured to perform authentication and post-authorization tasks.

Plug-ins configured for post-authorization processing using failover cookies, encrypt and store a credential as a failover cookie in the transaction response.

Plug-ins configured for using failover cookies for performing authentication, re-authenticate clients using the encrypted credential from a failover cookie found in the transaction request.

To enable SSO using failover cookies, assign the reference 'failover' to the **authentication** and **post-authzn** parameters in the **[common-modules]** stanza of the configuration file:

```
[common-modules]
authentication = failover
post-authzn = failover
```

## Using global single sign-on (GSO)

IBM Security Access Manager Plug-in for Web Servers can be configured to grant 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.

To create a GSO solution, Security Access Manager GSO resources and GSO resource groups must first be created using the Web portal manager or the **pdadmin** utility. For details on creating GSO resources and GSO resource groups refer to the *IBM Security Access Manager for Web: Base Administration Guide*.

The Basic Authentication (BA) post-authorization module is called after a request has been authorized to determine if a resource credential is available for the requested resource. The resource credential is a username/password combination that is mapped into each resource and stored in the user registry.

The BA post-authorization module retrieves the resource credential appropriate for the user and the requested application resource and creates an HTTP Basic Authentication header using the retrieved resource credential and adds this BA header to the HTTP request. The resource credential is retrieved from the user registry for the first request only. For all subsequent requests, the resource credential is retrieved as session information.

The following figure illustrates how the GSO mechanism is used to retrieve user names and passwords for backend application resources.

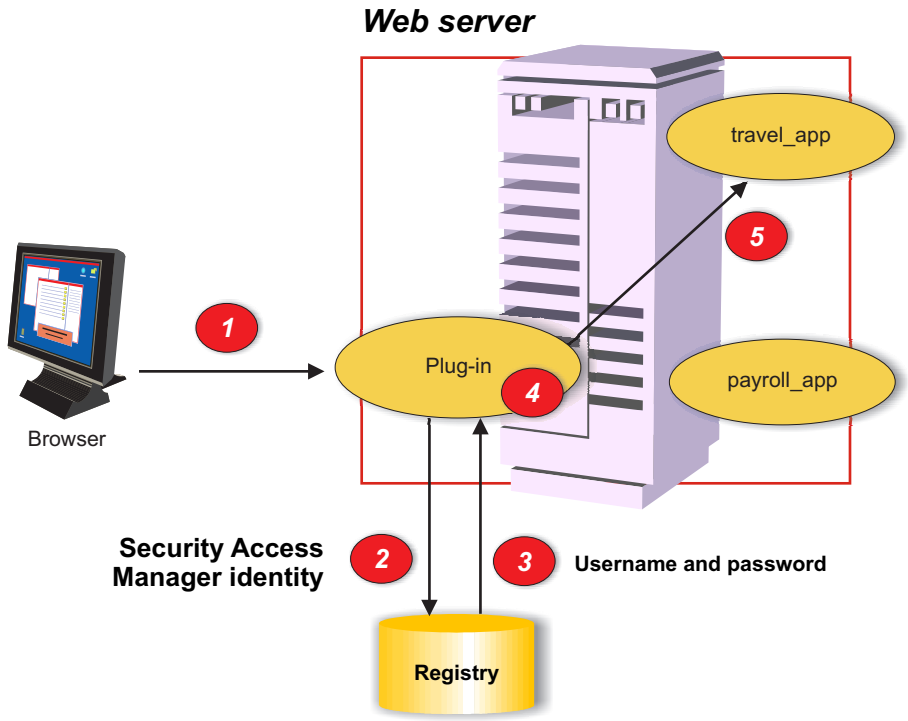


*Figure 10. User access to secure applications using GSO.*

1. User Michael requests access to the protected back-end Web server application, *travel-app*. Security Access Manager authenticates the client and a Security Access Manager identity is obtained. If the resource requested is unprotected, the request is forwarded to the Web server for handling.

   **Note:** The single sign-on process is independent of the initial authentication method.

2. The plug-in passes the Security Access Manager identity to the user registry server (LDAP or URAF).

   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 only be created for registered users.

   The following table illustrates the structure of the GSO resource credential database:

| Michael | Jane |
|---|---|
| resource: travel-app<br>username=mike<br>password=123 | resource: travel-app<br>username=Jane<br>password=abc |
| resource: payroll-app<br>username=smith<br>password=456 | resource: payroll-app<br>username=Jones<br>password=xyz |

3. The registry returns username "mike" and password "123" to the plug-in.

4. The plug-in inserts Michael's username and password information in the HTTP Basic Authentication (BA) header of the request that is sent back to the Web server.
5. The Web server authenticates Michael (for the resource he has requested) based on his credentials in the BA header inserted in the request from Step 4, as if it is from the client.

## Configuring Global single sign-on

To enable Global Single sign-on functionality you need to configure `pdwebpi.conf`. In the **[common-modules]** stanza, specify the value *BA* for the **post-authzn** parameter as in the following:

```
[common-modules]
authentication = ...
session = ...
post-authzn = BA
```

Ensure that in the **[modules]** stanza the parameter **BA** is assigned at least the default module; that is:

```
[modules]
BA = pdwpi-ba-module
```

Within the **[BA]** stanza of the `pdwebpi.conf` configuration file there are a number of parameters used for configuring the BA post-authorization module. These are:

- **basic-auth-realm**
- **strip-hdr**
- **add-hdr**
- **gso-resource-name**
- **supply-password**
- **supply-username**

For achieving GSO to back-end applications, the parameters **add-hdr** and **gso-resource-name** need configuring. Other BA parameters are discussed in more detail in "Configuring Basic Authentication" on page 61.

The **add-hdr** parameter controls the addition of a new BA header once the request has been authenticated. For achieving GSO, set this parameter to the value *gso*:

```
[BA:virtual_host1]
...
add-hdr = gso
```

The setting of the **add-hdr** parameter to the value *gso* means a new BA header is added to the HTTP request based on resource information that is stored in the user registry. The **gso-resource-name** parameter in the **[BA]** stanza of the configuration file specifies the name of the Web server resource that is to be GSO enabled. This can be specified on a per virtual host basis. The resource credential stored in the user registry is mapped to each resource stored in the user registry.

Set the **gso-resource-name** parameter to the name of the Security Access Manager resource that is to be GSO enabled. For example:

```
[BA:virtual_host1]
...
gso-resource-name = payroll-app
```

Only one GSO resource name can be specified per virtual host. If no value is specified for **gso-resource-name**, the virtual host name is used as the GSO resource name.

**Note:** If you are sharing an LDAP registry between Sun Java System (formerly Sun ONE) and Security Access Manager, you cannot create GSO resource credentials within Security Access Manager with target user names the same as those usernames expecting to authenticate to the Sun Java System Web Server. This is because the Sun Java System Web Server cannot limit the LDAP search criteria when authenticating users to search only for objects of the correct LDAP object class.

## Security Provider NEGOtiation (SPNEGO) single sign-on

Using SPNEGO as an authentication mechanism within the plug-in provides a single sign-on capability that permits users to access resources on a secure IIS Web server from a Windows client without the need for authentication other than the initial logon to the domain. Operational and configuration details of SPNEGO single sign-on are included in "Configuring SPNEGO authentication" on page 72.

## Single sign-on using forms

Single sign-on forms authentication allows IBM Security Access Manager Plug-in for Web Servers to transparently log an authenticated Security Access Manager user in to a plug-in secured Web server that requires authentication using an HTML form.

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 the plug-in. Plug-in form-based single sign-on provides a quick integration solution that should be regarded as an interim solution to be used while a more trusted and efficient method for authentication is developed.

Enabling single sign-on forms authentication produces the following results:

- The plug-in interrupts the authentication process initiated by the back-end application.
- The plug-in supplies data required by the login form and submits the login form on behalf of the user.
- 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.

The plug-in must be configured to:

- Recognize and intercept the login form
- Fill in the appropriate authentication data

The administrator enables forms single sign-on by configuring how the login form is to be recognized, completed, and processed.

### Forms single sign-on process flow

The following scenario assumes that the plug-in has already authenticated the user.

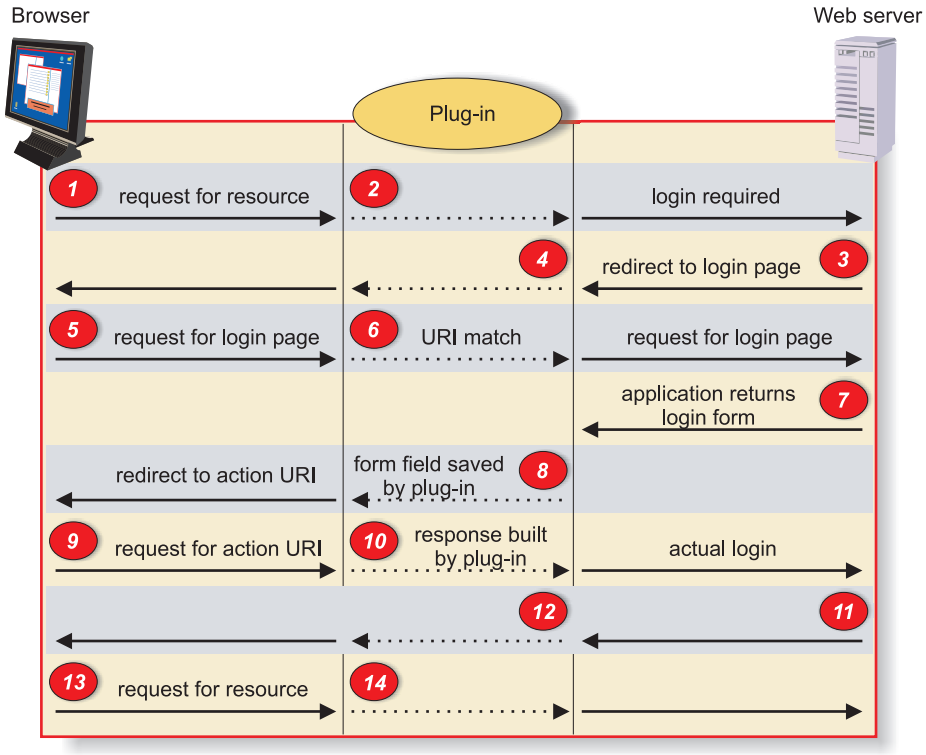


*Figure 11. Forms single sign-on process flow.*

1. The user requests a resource on a protected virtual host.
2. The plug-in passes the request to the back-end application.
3. Because the back-end application requires the user to authenticate, a redirect to the application's login page is sent back to the plug-in.
4. The plug-in passes the redirect to the browser.
5. The browser follows the redirect and requests the login page.

   **Note:** Everything to this point in the process flow is standard plug-in functionality.
6. The plug-in has been configured for forms single sign-on. The plug-in FSSO module recognizes the request as a request for a login page, based on information contained in the plug-in configuration file. The request is sent to the application.
7. The application returns the login page and perhaps application-specific cookies.
8. The plug-in intercepts the response and parses the HTML returned to identify the login form. When the plug-in finds a HTML form in the document it compares the action URI in the form to the value of the **login-form-action** parameter from the plug-in configuration file.

   If there is a match, the plug-in uses the form found, otherwise the plug-in keeps searching for other forms. If no form in the page matches the action URI pattern from the configuration file then the plug-in aborts forms single sign-on processing and passes the unmodified response back to the browser.

   If a login form is found, the plug-in parses the form HTML in the document to identify the request method, the action URI, and any other input fields in the form, saving them for use in step 10. It then sends the browser a redirect

to the login form's action URI with a unique request identifier appended as a query. Any application specific cookies are also included with the redirect.

9. The browser follows the redirect and requests the action URI.
10. The plug-in recognizes the inbound request by its unique query string and generates the authentication request using the rules from the argument-stanza and data saved in step 8. The completed login form (authentication request) is then sent to the back-end application.
11. The application authenticates using the plug-in supplied authentication data in the form. The application returns a redirect to the originally requested resource.
12. The plug-in returns the redirect to the browser.

    **Note:** This completes the forms SSO-specific functionality.
13. The browser follows the redirect and requests the resource.
14. The plug-in passes the request to secure resource.

During this process, the browser makes four requests to the plug-in. From the user's perspective, only a single request for the resource 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:

1. The login page or pages for the application must be uniquely identifiable using a single regular expression or several regular expressions.
2. 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 is 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 the plug-in's HTML filtering. The regular expression should match the action URI prior to being filtered.
3. Client-side scripting may be used to validate input data, but it must not modify the input data. This precludes support for Web sites using Javascript to build logon forms dynamically or to set cookies in the user's browser.
4. Login data is submitted at only one point in the authentication process.
5. The logon URI to be intercepted in step 8 in the previous section, must be processed as a single request by the underlying Web server. PHP scripts handled as external commands in Apache, for example, spawn multiple sub-request and cannot be intercepted.

## Enabling forms single sign-on

The FSSO module handles the forms single sign-on process. The module needs to be called after the authorization of the request and before the Web server responds to the request. The FSSO module therefore needs to be configured as a **post-authzn** module and as a **response** module. These are specified in the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. That is:

```
[common-modules]
...
response = fsso
```

The **response** module is used to capture the login form presented by the Web server so that it can be processed.

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for fsso exists:

```
[modules]
fsso = pdwpi-fsso-module
```

Since the primary role of the plug-in is to protect Web resources from unauthorized access, it must authorize all requests for resources even if they are part of a form-based single sign-on process. The plug-in checks the ACL database before allowing access to the back-end application login page and also checks before allowing access to the URI specified in the form action (where the completed login form is sent). If the security policy does not give permission for the current user to access these pages, then the form-based single sign-on will fail.

## Configuring forms single sign-on

The forms single sign-on configuration information is located in the `pdwebpi.conf` configuration file under the **[fsso]** or **[fsso:***virtual-host***]** stanza. The stanza contains one or more **login-page-stanza** entries which point to other custom-named stanzas that contain configuration information for the login pages found on the back-end application.

The ability to support multiple login pages is important because a server might host several applications that each use a different authentication method.

For example:

```
[fsso]
login-page-stanza = login-from-1
login-page-stanza = login-form-2
```

### 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:

| Parameter | Description |
|---|---|
| **login-page** | This parameter specifies a pattern, using a regular expression, that uniquely identifies requests for an application's login page. The configured pattern is compared against the request URI. |
| **login-form-action** | 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 multiple forms match the pattern, then the first is used. |
| **argument-stanza** | This parameter points to another custom stanza that lists the fields and data required for completing the login form. |
| **gso-resource** | This parameter supplies the name of the Security Access Manager resource that is to be used when loading GSO sourced data defined in the arguments-stanza. Only one GSO resource name can be specified per custom login page stanza. If no value is specified for **gso-resource**, the virtual host name is used as the GSO resource name. |

For example:

```
[login-form-1]
login-page = /cgi-bin/getloginpage*
login-form-action = *
argument-stanza = form1-data
gso-resource = payroll-app
```

**About the login-page parameter:**

The value of the **login-page** parameter is a regular expression that the plug-in uses to determine if an incoming request is actually a request for a login page. If this is the case, the plug-in 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.

**Note:** Where two or more pages match the **login-page** regular expression, the first match defines the down stream processing of the **login-form**.

The **login-page** regular expression is compared against the request URI. In the following example, the URI of a request to a protected virtual host called `myserver1` might appear as follows:

```
https://myserver1.mycompany.com/auth/login.html
```

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 page returned by the back-end server following a request matching the **login-page** parameter. 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 the plug-in 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 **login-form-action** 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 special characters allowed in the regular expressions used in the forms single sign-on configuration are defined in Appendix F, “Special characters allowed in regular expressions,” on page 295.

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:

```
<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 **gso-resource** 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 Security Access Manager user name and DN. To use the user's Security 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 using 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
```

**Note:**

1. The plug-in does not execute script code (Javascript, AxcitveX, etc.) before the form is submitted, which may cause problems if the code is required for the login process. Problems will not arise if this code simply checks the input prior to submission but problems may occur if the code modifies the user input.
2. Although forms-based SSO can make use of information in the GSO database, it does not interfere with the GSO capability supplied by the BA module.

   It is possible, if required, to have one GSO target in use to fill in the Basic Authentication headers sent to the backend server and another specified in the forms-based SSO configuration for use when filling in login forms.

## 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:**

```
<form name="confirm" method="post" action="../files/wcls_hnb_welcomePage2.cgi">
<p>
Employee Serial Number:&nbp;
<input name="data" size="10" maxlength="6">
<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>
<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 7. Cross-domain sign-on solutions

When IBM Security Access Manager Plug-in for Web Servers is implemented to provide protection for a secure domain, there is often a requirement to provide solutions for single sign-on to resources. This chapter discusses two methods for achieving single sign-on across different plug-in protected domains: e-Community single sign-on and cross domain single sign-on (CDSSO). Both solutions use a trusted token to pass user authentication information between different domains.

Which solution you choose will depend on the amount of flexibility required. e-Community single sign-on uses a central server which coordinates the single sign-on process among the different domains. With CDSSO, there is no central authentication server and no automated re-directs which provides more flexibility.

The following topics are covered:



## Cross domain single sign-on (CDSSO)

Security Access Manager Cross-Domain Single sign-on (CDSSO) provides a mechanism for transferring user credentials across multiple secure domains.

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. The CDSSO authentication mechanism does not rely on a Master Authentication Server as "e-Community single sign-on" on page 171 does.

With CDSSO, 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 second domain now has the user's identity (as authenticated in the first domain) and the user is not forced to perform another login.

CDSSO domains are based on DNS domains. All servers in the same DNS domain share the same symmetric key. In order to perform CDSSO with servers in another DNS domain (which may or may not also be in a different Security Access Manager domain) a different key is needed.

### Authentication process flow for CDSSO

The CDSSO process flow is described in the diagram and text below. Any user who wants to participate in multiple domains must have a valid user account in the primary domain, in this case domainA, 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.

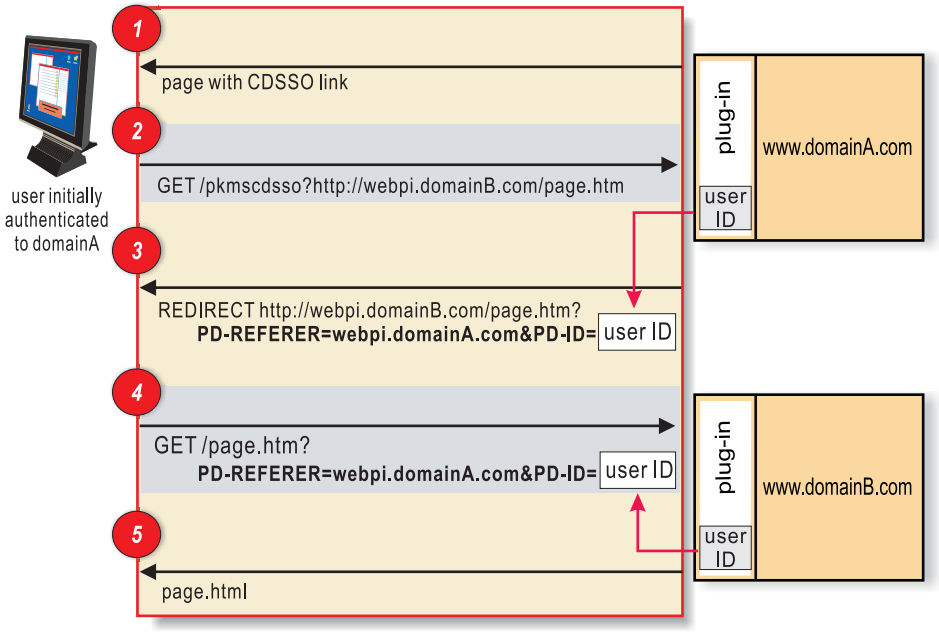


*Figure 12. CDSSO process flow*

1. The user makes a request to access a resource in domain B using a custom link on a Web page in domain A.
2. The link contains a special CDSSO expression specified by the **uri** parameter in the **[cdsso]** stanza of the `pdwpi.conf` configuration file. The default value is *pkmscdsso*:

   `/pkmscdsso?`*`destination-URL`*

   For example:

   `/pkmscdsso?https://www.domainB.com/index.html`

   The request is first processed by the plug-in server in domain A. The plug-in builds an authentication token that contains the user's Security Access Manager identity (short name), the current domain ("A"), additional user information, and a time stamp.

   The additional user information (extended attributes) is 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.

   Plug-in triple-DES encrypts this token data with the symmetric key generated by the **cdsso_key_gen** utility. This key file is shared and stored in the **[cdsso-domain-keys]** stanza of the `pdwebpi.conf` configuration file on both domain A and domain B plug-in enhanced Web servers.

   The token contains a configurable time stamp (**authtoken-lifetime**) that defines the lifetime of the token. The time stamp, when properly configured, can prevent replay attacks.
3. The domain A plug-in server re-directs the request plus the encrypted token back to the browser and then to the domain B plug-in server (HTTP redirection).
4. The domain B plug-in server uses its version of the same key file to decrypt and validate the token as coming from the referring domain.

The domain B plug-in server now calls out to a CDSSO authentication mechanism library. This CDSSO library in turn calls out to the 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 CDSSO library. The CDSSO library uses this information to build a credential.

5. The domain B 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.

## Enabling and disabling CDSSO authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable CDSSO authentication, assign the term *cdsso* to the authentication parameter:

```
[common-modules]
authentication = cdsso
```

When using CDSSO authentication, the plug-in must also be configured for CDSSO post-authorization processing. In the **[common-modules]** stanza of the `pdwebpi.conf` configuration file, add the parameter **post-authzn** as in the following:

```
[common-modules]
authentication = cdsso
post-authzn = cdsso
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for forms authentication exists:

```
[modules]
cdsso = pdwpi-cdsso-module
```

## Encrypting the authentication token data

The plug-in 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 plug-in enhanced Web server in each participating domain. Each participating plug-in server in each domain needs to use the same key.

**Note:** The creation and distribution of key files is not a part of the Security Access Manager CDSSO process.

The **cdsso_key_gen** utility requires that you specify the location (absolute pathname) of the key file when you run the utility:

UNIX: `# cdsso_key_gen` *`absolute-pathname`*

Windows: `MSDOS> cdsso_key_gen` *`absolute-pathname`*

Enter this key file location in the **[cdsso-domain-keys]** stanza of the `pdwebpi.conf` configuration file of the participating plug-in server in each domain. The **[cdsso-domain-keys]** stanza derives its name from the `pdwpi-cdsso-module` name defined in the **[modules]** stanza. It takes the form **[***cdsso-module-name***-domain-keys]**. The domain keys can be specified on a per virtual host basis by creating a

**[***cdsso-module-name***-domain-keys:***virtual-host-name***]** stanza. The format of the entry includes the domain name and the key file location:

```
[cdsso-domain-keys]
domain-name = keyfile-location
```

**Domain A Configuration Example:**

```
[cdsso-domain-keys]
www.domainB.com = pathname/A-B.key
```

**Domain B Configuration Example:**

```
[cdsso-domaina-keys]
www.domainA.com = pathname/A-B.key
```

In the above example, the A-B.key file would be generated on one machine (Plug-in A, for example) and manually (and securely) copied to the other machine (Plug-in B, for example).

## Configuring the token time stamp

The token contains a configurable time stamp that defines the lifetime of the authentication token. Once 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 `pdwebpi.conf` configuration file, sets the token lifetime value. The value is expressed in seconds. The default value is 180:

```
[cdsso]
authtoken-lifetime =180
```

This value may be overriden on a per-virtual host basis. You must take into account any clock skew between the participating domains.

## Including credential attributes in the authentication tokens

You can include credential attributes in the CDSSO tokens by specifying them in the **[cdsso-token-attributes]** stanza of the plug-in configuration file. The attributes to be included can be specified on a peer-to-peer or per-domain basis. The credential attributes listing in this stanza is only relevant when the default SSO token creation and consumption libraries are in use. If you do not require credential attributes in CDSSO vouch-for tokens, then you can leave this stanza empty.

The default name of this stanza is derived from the module name for the pdwpi-cdsso-module defined in the **[modules]** stanza. It is of the form **[***cdsso_module_name***-token-attributes]**.

The values in the **[cdsso-token-attributes]** stanza are default across all virtual hosts and can be overridden on a per virtual-host basis by creating a **[***cdsso_module_name***-token-attributes:***virtual_host***]** stanza.

The format of the entries is: *domain_name = pattern1, pattern2, ... pattern n*.

Credential attributes matching the specified patterns for a target host or domain are included in CDSSO vouch-for tokens constructed for that target host or domain. Only a single value for each attribute is used, and only string values are supported. Other types of credential attribute values are ignored. Patterns can be specified using the pattern matching characters explained in Appendix F, "Special characters allowed in regular expressions," on page 295.

For example:

```
[cdsso-token-attributes]
ibm.com = attrprefix_*, *name*
tivoli.com = *_attrsuffix, some_exact_attribute
```

A default set of attributes can be configured using a `<default>` entry in this stanza. Such a default set of attributes is used when there is no other entry matching a particular target host. If the `<default>` entry is not present, then no attributes will be included by default.

### Accepting and rejecting credential attributes from CDSSO authentication tokens

You can specify the credential attributes to accept and those to reject from incoming CDSSO authentication tokens by specifying the values in the **[cdsso-incoming-attributes]** stanza.

Unlike the outgoing attributes configuration, incoming attributes cannot be configured on a per-peer or per-domain basis. Only one set of attribute patterns can be configured, and these patterns will be applied to incoming tokens regardless of source. This processing only takes place if the default SSO token creation and consumption libraries are in use.

The default name of this stanza is derived from the module name for the pdwpi-cdsso-module defined in the **[modules]** stanza. It is of the form **[***cdsso_module_name***-incoming-attributes]**. The values in this stanza are default across all virtual hosts. However, they may be overridden on a per virtual-host basis by configuring a **[***cdsso_module_name***-incoming-attributes:***virtual_host***]** stanza.

The format of entries in this stanza is:

```
attribute_pattern = preserve|refresh
```

Attributes in CDSSO tokens that match a *refresh* entry are removed from the token before the CDMF library is called to map the remote user into the local domain. Attributes matching a *preserve* entry, or matching none of the entries, are retained. If no entries are configured, then all attributes are retained.

## Specify the sso-create and sso-consume libraries

To specify **sso-create** and **sso-consume** libraries, edit the plug-in configuration file. In the **[authentication-mechanisms]** stanza, uncomment the entry for sso-create and sso-consume and add the name of the plug-in failover cookie library appropriate for the operating system type.

The default configuration file entry is:

```
[authentication-mechanisms]
sso-create = /opt/pdwebrte/lib/ibssocreate.so
sso-consume = /opt/pdwebrte/lib/libssoconsume.so
```

Alternatively, when you have developed an external authentication mechanism library that implements a customized version of sso-create and sso-consume functionality, insert the name of the custom external authentication mechanism as the value for the configuration file keyword. For example, if you developed a custom external authentication mechanism for sso-create, enter the absolute path name:

```
[authentication-mechanisms]
sso-create = /dir_name/custom_cdas_sso-create.so
```

## Expressing CDSSO links

Links to resources on a secondary secure domain must contain a special CDSSO expression which is configured using the **uri** parameter in the **[cdsso]** stanza on the configuration file. The default value is /pkmscdsso:

```
/pkmscdsso?destinationURL
```

When configured as a post-authorization module, requests to /pkmscdsso?*remote-uri* will redirect the client to, *remore-uri*?PD-REFERER=*this-host* &*argument=authentication-token*

The name of the query string argument specifying the authentication token is configured using the **cdsso-argument** parameter in the **[cdsso]** stanza of the pdwebpi.conf configuration file. The default value is PD-ID. This may be overriden on a per-virtual host basis.

The default value, PD-ID, of the **cdsso-argument** parameter should only be changed when a custom SSO create/consume library is in use. When using the shipped SSO create/consume libraries, the default, PD-ID, must be used.

## Protecting the authentication token

While the authentication token does not contain authentication information (such as username 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 plug-in enhanced Web 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, a malicious user 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 plug-in 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.

## e-Community single sign-on

IBM Security Access Manager Plug-in for Web Servers e-community single sign-on functionality allows users to access resources across multiple servers in multiple domains without requiring re-authentication.

An "e-community" is a group of distinct domains (Security 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 different 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 Security 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 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 be authenticated.

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 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 single sign-on process flow” on page 172.

## e-Community single sign-on features and requirements

e-Community single sign-on has the following features and requirements:

- e-Community functionality supports access using direct URLs (bookmarks) to resources.
- e-Community implementation requires a consistent configuration across all plug-ins in all domains participating in the e-community.
- 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.

  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 identities. 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.
- The e-community implementation is supported on both HTTP and HTTPS.
- Configuration for e-community is set in the `pdwebpi.conf` file of each participating plug-in.

## e-Community single sign-on process flow

An e-community consists of a plug-in-enhanced master authentication server (MAS) and additional plug-in-enhanced servers acting as an e-community. The e-community single sign-on solution can also interoperate with WebSEAL protected resources.

The e-community implementation is based on a vouch-for system. Normally, when unauthenticated users request a resource through the plug-in they are prompted for authentication information. In an e-community configuration, the plug-in server identifies a vouch-for server and requests verification from this vouch-for server that the user has authenticated. The vouch-for server stores valid credential information for the 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.

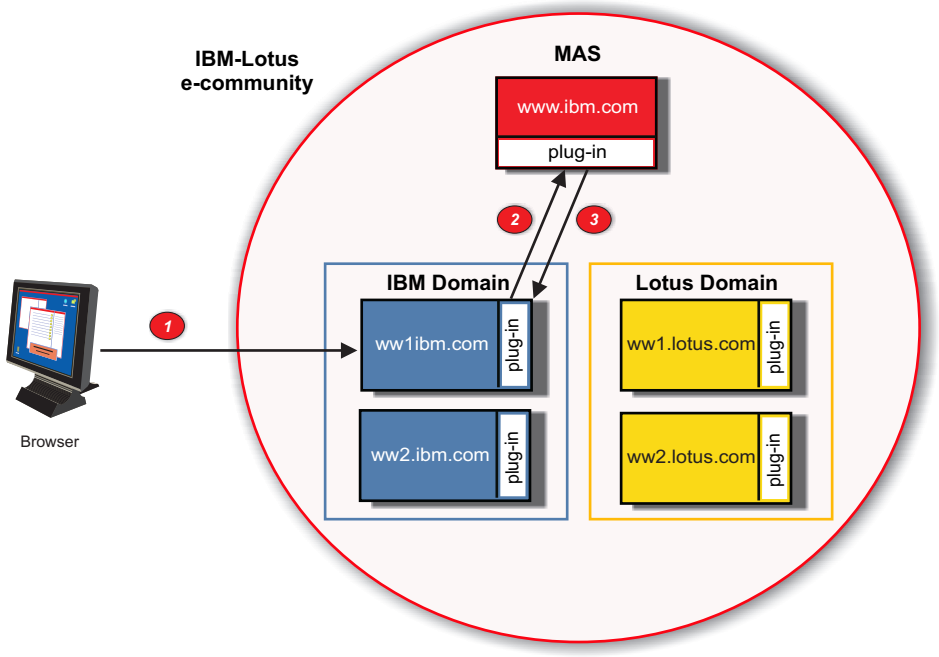


*Figure 13. Logging into an e-community*

The example above shows two domains, IBM domain and Lotus® domain, that exist within an e-community. The following processes take place the first time a user logs on to a secure Web site within the e-community:

1. The user requests access to a resource on the Web server ww1.ibm.com. The plug-in intercepts the request and confirms that ww1.ibm.com is configured as part of the Tivoli-IBM-Lotus e-community. The MAS server in the e-community is identified from the ww1.ibm.com configuration.
2. The request is passed to the MAS - www.tivoli.com. The MAS authenticates the request on behalf of ww1.ibm.com and issues a vouch-for token that becomes the user's e-community identity. The user identity information in the token is encrypted.
3. The MAS sends the vouch-for token to ww1.ibm.com. ww1.ibm.com treats this vouch-for token as proof that the user has successfully authenticated to the MAS and can now access the requested resource based on normal authorization controls.

## The e-community cookie

The e-community cookie has these characteristics:

- The e-community cookie is a domain-specific cookie set by one plug-in, stored in the memory of the user's browser, and transmitted to other plug-in instances (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 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 is located plays a less significant role.
- The cookie has a lifetime (timeout) value that is set in the `pdwebpi.conf` 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.

## 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. These URLs are constructed during the e-community vouch-for HTTP re-directs based on the configuration information in `pdwebpi.conf`.

### 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 an HTTP re-direct to the vouch-for server (either the MAS or a 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 re-direct the browser back to the originally requested page.

The **pkmsvouchfor** vouch-for URL is configurable.

For example:

`https://www.tivoli.com/pkmsvouchfor?companyABC&https://ww2.lotus.com/index.html`

### The vouch-for reply

The vouch-for reply is the response from the vouch-for server to the target server.

The vouch-for reply contains the following information:

`https://`*`target_url`*`?PD-VFHOST=`*`vouch_for_server`*`&PD-VF=`*`encrypted_token`*

The **PD-VFHOST** parameter 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 (PD-VF). The **PD-VF** parameter represents the encrypted vouch-for token.

For example:

`https://ww2.lotus.com/index.html?PD-VFHOST=www.tivoli.com&PD-VF=3qhe9fjkp...ge56wgb`

## 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 re-direct 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) at 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 `pdwebpi.conf` configuration file. This value can be very short (seconds) to reduce the risk of a re-play attack.

## Encrypting the vouch-for token

IBM Security Access Manager Plug-in for Web Servers must encrypt the authentication data placed in the token using a key generated by the **cdsso_key_gen** utility located in the `pdwebrte/bin` directory. You must "synchronize" this key by sharing the key file with each plug-in server in each participating domain. Each participating plug-in server in each domain needs to use the same key.

**Note:** The creation and distribution of key files is not a part of the Security 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 full path to the utility and the location (absolute pathname) of the key file when you run the utility:

**UNIX:**

```
# /opt/pdwebrte/bin/cdsso_key_gen absolute_pathname
```

**Windows:**

```
MSDOS> install_path/pdwebrte/bin/cdsso_key_gen absolute_pathname
```

The encryption keys are configured in the **[ecsso-domain-keys]** stanza of the `pdwebpi.conf` configuration file. Details of this configuration are covered in the next section, “Configuring an e-community.”

## Configuring an e-community

This section reviews all the configuration parameters required for an e-community implementation. These parameters are in the `pdwebpi.conf` file. You must carefully configure this file for each participating plug-in in the e-community.

**Enabling and Disabling e-Community Members**

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable a plug-in server to operate within an e-community, assign the term *ecsso* to the **authentication** and **pre-authzn** parameters as in the following:

```
[common-modules]
authentication = ecsso
pre-authzn = ecsso
```

When configuring for non-MAS e-community members, ecsso authentication must take precedence over other authentication schemes.Ecsso must be specified before other authentication schemes in the list of authentication modules. If the ecsso module is to take precedence over an authentication module specified with a higher authentication level than the default of 1, then the ecsso module itself must be configured with at least the same authentication level.

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for e-community SSO exists:

```
[modules]
ecsso = pdwpi-ecsso-module
```

**e-community-name**
The **e-community-name** parameter identifies the name of the e-community the server belongs to. For example:

```
[ecsso]
e-community-name = companyABC
```

The **e-community-name** value must be the same for all members of an e-community.

**is-master-authn-server**
This parameter identifies whether this server is the MAS or not. Possible values are `yes` or `no`. The parameter would be set as follows for the e-community MAS:

```
[ecsso]
is-master-authn-server = yes
```

Multiple plug-ins 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 plug-in servers in the e-community.

If **is-master-authn-server** is set to `yes`, then the server accepts vouch-for requests from other plug-in instances whose **e-community-name** is the same and whose domain keys are listed in the **[ecsso-domain-keys]** stanza.

**master-authn-server**
If the **is-master-authn-server** parameter is set to `no`, then the **master-authn-server** parameter must be uncommented and specified. The parameter identifies the fully qualified domain name of the e-community MAS. For example:

```
[ecsso]
 master-authn-server = www.ibm.com
```

**master-http-port**
Assign the port number that the master authentication server uses to receive HTTP requests. If the port number is not the standard port 80 then the non-standard port number must be specified here.

```
[ecsso]
master-http-port = port_number
```

**master-https-port**
Assign the port number that the master authentication server uses to receive HTTPS requests. If the port number is not the standard port 443 then the non-standard port number must be specified here.

```
[ecsso]
master-https-port = port_number
```

**vf-token-lifetime**
This 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 between participating servers. By default the parameter is set as:

```
[ecsso]
vf-token-lifetime = 180
```

**vf-url**
This parameter specifies the vouch-for URL The value must begin with a forward-slash (/). The default setting value is:

```
[ecsso]
vf-url = /pkmsvouchfor
```

You can also express an extended URL:

```
vf-url = /ecommA/pkmsvouchfor
```

**vf-argument**
The value of the **vf-argument** parameter is the argument name of the vouch-for token as it appears in the vouch-for reply. The default value of PD-VF should only be changed if custom create and consume modules are in use and a different argument name is used to represent the vouch-for token.

The value is used to construct vouch-for replies by the MAS and to distinguish incoming requests as ones with couch-for information by participating ECSSO servers.

```
[ecsso]
vf-argument = PD-VF
```

**allow-login-retry**
A MAS that employs a username/password-based authentication scheme has two options when a user has performed an unsuccessful logon: it can prompt the users to input their credentials again, or it can immediately redirect users back to the server they originally attempted to access without vouching for the user. In the later case, users are forced to authenticate directly to the subordinate server. The **allow-login-retry** parameter controls this behavior at the MAS. This parameter is applicable only to the configuration of the MAS within an ecsso community.

**Note:** Users can attempt to reset an expired password.

Other login failures occurring at the MAS, such as account locked, cause an immediate redirection back to the subordinate server irrespective of the value of the **allow-login-retry** parameter. By default the parameter is set as:

```
[ecsso]
allow-login-retry = true
```

**use-utf8**
This parameter controls the string encoding within the ECSSO vouch-for tokens and e-community cookies. The value of this parameter only affects vouch-for tokens created and consumed by the default SSO create and consume libraries.

```
[ecsso]
use-utf8 = true
```

**disable-ec-cookie**
When set to yes, this option will disable the use of the e-community cookie, and only the MAS will generate vouch-for tokens. This will force the single-sign-on

process to always use the MAS, allowing the MAS to detect all hosts that sign on across the e-communities. This supports customers who wish to construct their own eCSSO Single Sign Off solution.

Disabling eCSSO cookies may provide extra security by ensuring that any compromise of a user's session in a slave domain will not result in impersonation of that user in either the master domain or other slave domains.

```
[ecsso]
use-utf8 = true
```

**no-mas-logout-uri**
This configuration entry is used to define the URI of a new form of the /pkmslogout page.

This page operates identically to /pkmslogout, except it does not redirect to the MAS's /pkmslogout page after logging out of the current host. Instead it simply performs the normal logout success process and returns the page, as defined by the `no-mas-logout-success` configuration entry. This supports customers wanting to use alternate methods of signing out all the hosts at the MAS.

Instead of using the technique by which each host to be signed out is visited sequentially by the MAS, all hosts could be visited simultaneously using features of HTML like iframes. A single page of iframes, one for each host to sign out, could be generated at the MAS. Each form would access /pkmslogout-nomas of each host. If they accessed /pkmslogout of each host, then each iform would be redirected back to the MAS, making it difficult to control the ensuing /pkmslogout recursion.

This single page signout method would be more robust in the case of a single host failing to respond. If a single host failed using the sequential sign out process then the sign out sequence would halt possibly leaving some hosts signed in.

```
[ecsso]
no-mas-logout-uri = /pkmslogout-nomas
```

**no-mas-logout-success**
This configuration entry is used to define the action which is taken by the server after the client has been successfully logged out through the `no-mas-logout-uri`.

The entry should correspond to either a macro HTML file, which is relative to the translated PDWebPI HTML directory (for example, `/opt/pdwebpi/nls/html/C/utf-8`), or a valid redirect URI. The redirect URI can be either absolute or server-relative, and can also contain macros. Be aware, however, that some clients impose restrictions on the maximum length of a URI; care should be taken to include only those URI elements that are required.

```
[ecsso]
no-mas-logout-success = logout_success.html
```

**ecsso Domain Keys**
Defined in the**[ecsso-domain-keys]** stanza of the configuration file are the locations of the key files required for encrypting and decrypting tokens between the MAS and participating servers in remote domains. Configuration of the MAS involves defining the keys for each domain for which it is the master. Configuration of e-community members other than the MAS involves defining the key for the domain and for the MAS. You must specify the fully qualified domain names for the servers and the absolute path names for the key file locations.

The following MAS configuration example provides the MAS, located within the ibm.com domain, with key files for communicating with two remote domains:

```
[ecsso-domain-keys]
ibm.com = /abc/xyz/ibm.key
lotus.com = /abc/xyz/lotus.key
ibm = /abc/xyz/ibm.key
```

**Note:** In the preceding example, it is crucial to have **ibm.key** for data exchange between servers in the ibm domain.

Configuration for servers in the domains involves specifying the MAS domain and the corresponding key used to exchange information with the MAS. A key is also required for data exchange between servers in the domain. For example the **[ecsso-domain-keys]** stanza for a server in a domain participating in an e-community may look like this:

```
[ecsso-domain-keys]
#the key for data exchange between the MAS (ibm.com)
#and the ibm.com domain servers
ibm.com = /abc/xyz/ibm.key
#the key for data exchange between servers in the ibm.com domain
ibm.com = /abc/xyz/ibm.key
```

Failure to configure ecsso keys correctly generates warnings within the plug-in log file.

## Including credential attributes in the vouch-for tokens

You can include credential attributes in the eCSSO vouch-for tokens by specifying them in the **[ecsso-token-attributes]** stanza of the plug-in configuration file. The attributes to be included can be specified on a peer-to-peer or per-domain basis. The credential attributes listing in this stanza is only relevant when the default SSO token creation and consumption libraries are in use. If you do not require credential attributes in eCSSO vouch-for tokens, then you can leave this stanza empty.

The default name of this stanza is derived from the module name for the pdwpi-ecsso-module defined in the **[modules]** stanza. It is of the form **[***ecsso_module_name***-token-attributes]**.

The values in the **[ecsso-token-attributes]** stanza are default across all virtual hosts and can be overridden on a per virtual-host basis by creating a **[***ecsso_module_name***-token-attributes:***virtual_host***]** stanza.

The format of the entries is: *domain_name = pattern1, pattern2, ... pattern n*.

Credential attributes matching the specified patterns for a target host or domain are included in eCSSO vouch-for tokens constructed for that target host or domain. Only a single value for each attribute is used, and only string values are supported. Other types of credential attribute values are ignored. Patterns can be specified using the pattern matching characters explained in Appendix F, “Special characters allowed in regular expressions,” on page 295.

For example:

```
[ecsso-token-attributes]
ibm.com = attrprefix_*, *name*
tivoli.com = *_attrsuffix, some_exact_attribute
```

A default set of attributes can be configured using a `<default>` entry in this stanza. Such a default set of attributes is used when there is no other entry matching a particular target host. If the `<default>` entry is not present, then no attributes will be included by default.

## Accepting and rejecting credential attributes from vouch-for tokens

You can specify the credential attributes to accept and those to reject from incoming vouch-for tokens by specifying the values in the **[ecsso-incoming-attributes]** stanza. Unlike the outgoing attributes configuration, incoming attributes cannot be configured on a per-peer or per-domain basis.

Only one set of attribute patterns can be configured, and these patterns will be applied to incoming tokens regardless of source. This processing only takes place if the default SSO token creation and consumption libraries are in use. The default name of this stanza is derived from the module name for the pdwpi-ecsso-module defined in the **[modules]** stanza. It is of the form **[***ecsso_module_name***-incoming-attributes]**.

The values in this stanza are default across all virtual hosts. However, they may be overridden on a per virtual-host basis by configuring a [*ecsso_module_name*-incoming-attributes:*virtual_host*] stanza.

The format of entries in this stanza is:

```
attribute_pattern = preserve|refresh
```

Attributes in eCSSO vouch-for tokens that match a *refresh* entry are removed from the token before the CDMF library is called to map the remote user into the local domain. Attributes matching a *preserve* entry, or matching none of the entries, are retained. If no entries are configured, then all attributes are retained.

## Specify the sso-create and sso-consume libraries

To specify **sso-create** and **sso-consume** libraries, edit the plug-in configuration file. In the **[authentication-mechanisms]** stanza, uncomment the entry for sso-create and sso-consume and add the name of the plug-in failover cookie library appropriate for the operating system type.

The default configuration file entry is:

```
[authentication-mechanisms]
sso-create = /opt/pdwebrte/lib/ibssocreate.so
sso-consume = /opt/pdwebrte/lib/libssoconsume.so
```

Alternatively, when you have developed a external authentication mechanism library that implements a customized version of sso-create and sso-consume functionality, insert the name of the custom external authentication mechanism as the value for the configuration file keyword. For example, if you developed a custom external authentication mechanism for sso-create, enter the absolute path name:

```
[authentication-mechanisms]
sso-create = /dir_name/custom_cdas_sso-create.so
```

## Configuring e-community single sign-on - an example

In the following example there are two e-communities configured – lotus-domino and ibm-db2 – with a single MAS authenticating the requests for both communities.

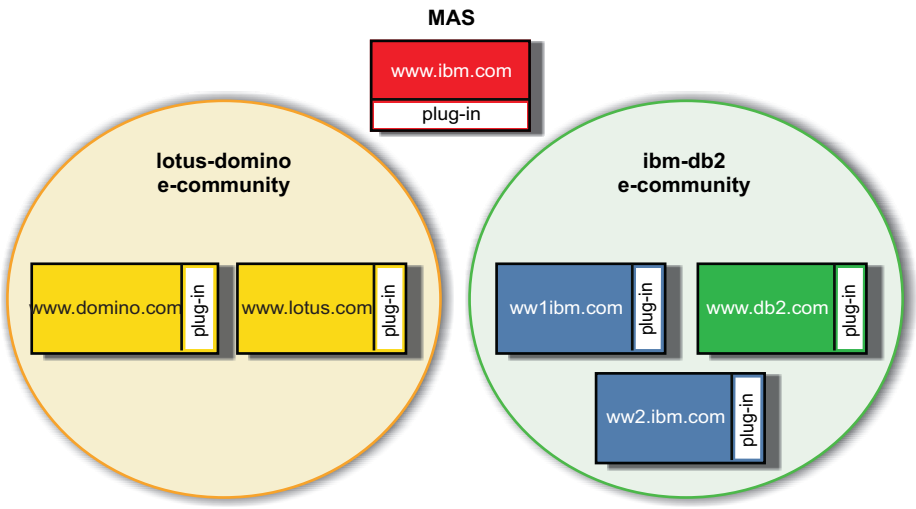


*Figure 14. e-Community single sign-on configuration example*

The following conditions apply for this example:

- www.ibm.com is the MAS for both e-communities.
- Two distinct domains (one server in each domain for simplicity) exist within the lotus-domino e-community – domino.com and lotus.com. Users accessing one of these domains can access the other without the need to re-authenticate as all access is granted via the MAS.
- The ibm-db2 e-community contains two distinct domains – ibm.com and db2.com. Users accessing one of these domains can access the other without the need to re-authenticate.
- Users accessing one of the ibm.com servers can access the other using a vouch-for token. Single sign-on in this case is achieved without the need for the MAS to grant access.

In the above example, the following configuration options apply:

**Configuration of the MAS – www.ibm.com**

As the MAS is the control center for more than one e-community, two distinct instances of the ecsso module need to be configured and the e-community names that the MAS controls need to be defined. The MAS needs to have specified all the keys of the main domains within all the communities it controls. Configuration is set as follows:

```
[modules]
ecsso1 = pdwpi-ecsso-module
ecsso2 = pdwpi-ecsso-module

[common-modules]
authentication = ecsso1
authentication = ecsso2

pre-authzn = ecsso1
pre-authzn = ecsso2
```

```
[ecsso1]
e-community-name = lotus-domino
is-master-authn-server = yes
.....etc

[ecsso2]
e-community-name = ibm-db2
is-master-authn-server = yes
.....etc

[ecsso1-domain-keys]
# one key for each domain the MAS controls
domino.com = /abc/ibmkeys/ibm-domino.key
lotus.com = /abc/ibmkeys/ibm-lotus.key
db2.com = /abc/ibmkeys/ibm-db2.key
ibm.com = /abc/ibmkeys/ibm.key
```

**Configuration of www.domino.com**

```
[modules]
ecsso = pdwpi-ecsso-module

[common-modules]
authentication = ecsso

pre-authzn = ecsso

[ecsso]
e-community-name = lotus-domino
is-master-authn-server = no
master-authn-server = www.ibm.com
.....etc

[ecsso-domain-keys]
#key for encrypting/decrypting data
#between servers in the domino.com domain
domino.com = /abc/domino-keys/domino.key
#key for encrypting/decrypting data between
#servers in the domino.com domain and the MAS
ibm.com = /abc/domino-keys/ibm-domino.key
```

**Configuration of www.lotus.com**

The configuration parameters for achieving single sign-on to www.lotus.com will be identical to those configured for www.domino.com except the domain keys will be different. Domain keys configuration for www.lotus.com would be as follows:

```
[ecsso-domain-keys]
#key for encrypting/decrypting data
#between servers in the lotus.com domain
lotus.com = /abc/lotus-keys/lotus.key
#key for encrypting/decrypting data
#between servers in the lotus.com domain and the MAS
ibm.com = /abc/lotus-keys/ibm-lotus.key
```

**Configuration of www.db2.com**

```
[modules]
ecsso = pdwpi-ecsso-module

[common-modules]
authentication = ecsso

pre-authzn = ecsso

[ecsso]
e-community-name = ibm-db2
is-master-authn-server = no
master-authn-server = www.ibm.com
```

```
.....etc

[ecsso-domain-keys]
#key for encrypting/decrypting data
#between servers in the db2.com domain
db2.com = /abc/db2-keys/db2.key
#key for encrypting/decrypting data between
#servers in the db2.com domain and the MAS
ibm.com = /abc/db2-keys/ibm-db2.key
```

**Configuration of ww1.ibm.com**

The e-community single sign-on configuration for ww1.ibm.com is identical to that of www.db2.com. Two keys are required, one for encrypting/decrypting data between the MAS and the ibm.com domain and a key for encrypting/decrypting data between servers within the ibm.com domain (i.e. ww1.ibm.com and ww2.ibm.com in this example).

```
[ecsso-domain-keys]
ibm.com = /abc/ibm-keys/ibm.key
```

**Configuration of ww2.ibm.com**

The definition of keys for ww2.ibm.com will be identical to that for ww1.ibm.com.

```
[ecsso-domain-keys]
ibm.com = /abc/ibm-keys/ibm.key
```

# Chapter 8. Application integration

IBM Security Access Manager Plug-in for Web Servers supports third-party application integration through environment variables and dynamic URL capability. The plug-in 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, the plug-in can provide access control on dynamic URLs, such as those that contain query text.

The following topics are covered in this chapter:



## Maintaining session state between the client and back-end applications

As described in Chapter 4, "Managing session state," on page 115, the plug-in can maintain session state with clients over HTTP and HTTPS using a variety of methods. The plug-in can also supply session information to back-end applications. This allows back-end applications to identify user sessions and request the plug-in to terminate them when required.

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.

### Enabling user session ID management

The **add-session-id-to-cred** entry in the **[performance]** stanza of the plug-in configuration file enables or disables the creation of a unique user session ID in the client credential. The default value is *true* (enabled):

```
[performace]
add-session-id-to-cred = true
```

To disable the creation of unique user session IDs, set **add-session-id-to-cred** to *false*.

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 "tag value" prefix that is configurable using the **tag-value-prefix** parameter in the **[pdweb-plugins]** stanza of the configuration file. Specifying a prefix prevents 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 plug-in instance name (to support multiple plug-in instances) and the standard plug-in session ID for the user.

A single user that logs in multiple times (for example, from different machines) has multiple plug-in session IDs. Because the user session ID is based on the plug-in session ID, there exists a one-to-one mapping between them. The unique user session ID is stored as an attribute in the user's credential. This allows the value to be passed across a junction as a 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 application server. This goal is accomplished by configuring the **HTTP-Tag-Value** extended attribute on the object.

Use the **pdadmin object modify set attribute** command to set an extended attribute on an object in the plug-in protected object space.

```
pdadmin> object modify object_name set attribute attribute_name attribute_value
```

The *attribute_name* command option allows the plug-in to perform a specific type of functionality. For example, specifying the **HTTP-Tag-Value** attribute enables the plug-in to extract a value from a credential extended attribute and send the value to the server in a 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 configured 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. In this example, a header called **PD-USER-SESSION-ID** is used:

```
pdadmin> object modify /PDWebPI/host set attribute \
HTTP-Tag-Value user_session_id=PD-USER-SESSION-ID
```

When the plug-in processes a user request to a back-end application server, it looks for any **HTTP-Tag-Value** extended attributes configured on the object.

In this example, the plug-in 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_number
```

In summary:

| | |
|---|---|
| Value of **HTTP-Tag-Value** attribute set on the plug-in object: | user_session_id=PD-USER-SESSION-ID |
| Attribute name and value as they appear in the user credential: | tagvalue_user_session_id:user_session_id_number |
| HTTP header name and value: | PD-USER-SESSION-ID:user_session_id_number |

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_PD-USER-SESSION-ID=user_session_id
```

## Terminating user sessions

User session ID management functionality can be used to terminate user sessions given a unique user session ID or a Security Access Manager username. These commands can be run from the PDADMIN command line (using server task) but they are intended for use by a back-end application through the PDAdmin API. Terminating a session using the User Session ID causes the plug-in to discard the single session that the User Session ID identifies. Other sessions from the same user can continue.

Terminating using the Security Access Manager username causes the plug-in to discard ALL sessions that are owned by the given username. This command may end many sessions if the user is logged in multiple times from different locations or different browsers.

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. Described below is two methods for terminating user sessions at an administration level:

An administrator can use the **pdadmin** utility to terminate a single user session using the user ID.

```
pdadmin> server task pdwebpi-plugin-instance-name terminate all_sessions user-id
```

Using the **all-sessions** command as shown above, each session for the specified user is terminated on all virtual-hosts on the machine.

This command can be refined to terminate user sessions for a particular user on a particular virtual host using the *-vhost* parameter as in the following (entered as one line):

```
pdadmin> server task pdwebpi-plugin-instance-name terminate all_sessions
user-id -vhost "virtual-host-name"
```

# Providing access control to dynamic URLs

IBM Security Access Manager Plug-in for Web Servers can apply authorization to Web objects based on a pattern match of the entire request string rather than just the object's URL. This is useful for Web applications that dynamically generate URLs in response to each user request that still require strong protection from unwanted use or access. This is also useful for assigning different permissions to the different methods of scripts.

For example, the query string **GET /cgi-bin/servercontrol?action=showstatus** would have different security requirements to **GET /cgi-bin/servercontrol?action=shutdown**. Each of these requests may need to be represented uniquely in the objectspace so that different policy can be applied to each.

The **dynurl** module allows a set of patterns to be defined that are matched against incoming requests. The patterns are matched against the entire request string so matching on information in the query string is possible.

For each DynURL pattern, a Security Access Manager object is defined. This object appears in the objectspace so that policy can be associated with it. At runtime, any request that matches a dynurl pattern is authorized using the object associated with the pattern rather than the object that represents the URL. By defining different patterns that independently match different query strings, different Security Access Manager objects can be used and different policy can be applied.

**Note:** Dynamic URL mapping functionality is incompatible with the plug-in's unprotected resource cache entries as discussed in "Policy for unprotected resources" on page 145.

## Configuring dynamic URLs

The**[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods.

To enable pattern-matching of dynamic URLs for incoming requests, configure the `dynurl` module as a pre-authorization module. Doing so allows the `dynurl` module to change the requested object before the authorization engine is reached.

```
[common-modules]
...
pre-authzn = dynurl
...
```

Ensure that the entry for dynamic URLs exists in the **[modules]** stanza of the `pdwebpi.conf` configuration file:

```
[modules]
...
dynurl = pdwpi-dynurl-module
...
```

The **[dynurl]** stanza, by default or the stanza name that matches the configured module name, contains the definitions for the dynamic URL pre-authorization module. This stanza can be overwritten on a per-virtual host basis, that is, **[dynurl:***virtual-host***]**.

The entries within the **[dynurl]** stanza are of the form *object = pattern*. Each entry is on a separate line. The order of the list determines the order in which the rules are processed. Entries which occur earlier in the stanza take precedence over entries that occur later in the stanza. For example:

```
[dynurl]
/servershutdown = /servercontrol.asp\?*action=shutdown*
/serverreset =/servercontrol.asp\?*action=reset*
/helppages = *help.html
```

**Note:** The objects all start with a backslash /. The last entry in the preceding configuration shows a second use of the **dynurl** module. In this case, the pattern matches a set of URLs, which is any URLs ending in `help.html`.

**Note:** In this case, DynURL does a many-to-one mapping of URLs to Security Access Manager objects. All requests for pages that are called `help.html` regardless of their path is authorized against the same Security Access Manager object. This might be useful in situations where files with similar names (that can be grouped

by a pattern) all have the same security requirements. However, each pattern that is defined is matched against every request and so slows down every authorization.

The use of the pattern `*help.html` might have security implications for scripts The following sample request

```
/servercontrol.asp?action=some_other_action&pointless_variable
_used_to_evade_acl_attached_to_server_control.asp=help.html
```

matches the `*help.html` dynamic URL. Therefore, access is evaluated based on the `/helppages` object rather than the `/servercontrol.asp` object. Similarly, a request for

```
/someotherscript?action=someaction&other_var=help.html
```

is evaluated based on the `/helppages` object rather than the `/someotherscript` object.

For a list of the special characters that are allowed in regular expressions that are used in the forms single sign-on configuration file, see Appendix F, “Special characters allowed in regular expressions,” on page 295.

In most cases, special characters are not required because the login page request is a single identifiable URI. In some cases, you can use `*` 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.

# Chapter 9. Authorization decision information retrieval

This chapter contains information that describes how IBM Security Access Manager Plug-in for Web Servers can provide, or acquire, authorization decision information (ADI) required to evaluate authorization rules that protect resources in the Security Access Manager domain.

The following topics are covered in this chapter:



## Overview of ADI retrieval

The Security 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 Security Access Manager for Web: 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 Security Access Manager for Web: 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 the plug-in, can be configured to provide ADI from its own environment. For example, the plug-in 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 dynamic ADI retrieval services.

  ADI can be obtained externally through the AMWebARS Web service. A call is made to the AMWebARS Web service through the resource manager's entitlement service. ADI from the external source is returned in XML format to the authorization rules evaluator.

  ADI can be obtained dynamically through calls to specific entitlement services that have been configured to retrieve ADI dynamically during rule evaluation. A call is made to each dynamic ADI retrieval service and the ADI values are returned to the authorization rules evaluator. Examples of dynamic ADI retrieval services shipped with Security Access Manager are the "Registry attribute entitlement service", which retrieves ADI values from the user registry, and the AMWebARS entitlement service, which retrieves ADI values using the

AMWebARS web service. Both are discussed in more detail in the *IBM Security Access Manager for Web: Authorization C Developer's Reference*.

## Retrieving ADI from the plug-in client request

Authorization decision information (ADI) may be contained in the request header, the request query string, and the request POST body. You can create authorization rules that refer to this authorization decision information (ADI). This is done using plug-in specific XML containers that refer to the ADI to be acquired.

The **resource-manager-provided-adi** parameter in the **[aznapi-configuration]** stanza of the `pdwebpi.conf` configuration file specifies—to the authorization rules evaluation process— the prefixes 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:

The following container names contain prefixes that are appropriate for the plug-in:

- 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.

Prefixes can be specific to any resource manager. Accordingly, the resource manager must be designed to respond appropriately to a request for ADI.

Authorization rules are written that specify the ADI required from client requests. For example, if the host name contained within the HTTP header is required as ADI, the `AMWS_hd_` prefix is used in the XML container name specified in the rule.

This plug-in-specific prefix alerts the authorization evaluation process that the required ADI is available in the client request and that the plug-in knows how to find, extract, and return this ADI. The `AMWS_hd_host` container name is sent to the plug-in. The plug-in 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. The plug-in returns the "host" header value (as an XML container) to the authorization rules evaluation process. The authorization rules evaluation process uses the value as ADI in its evaluation of the rule.

## 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 the plug-in knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_hd_host = "machineA"'>!TRUE!</xsl:if>
```

The plug-in is designed to know how to handle the extraction of ADI information from the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_hd_
```

The plug-in understands this information can be found in the request header name *host*. The plug-in 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 the plug-in knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_qs_zip = "99999"'>!TRUE!</xsl:if>
```

The plug-in is designed to know how to handle the extraction of ADI information form the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_qs_
```

The plug-in understands this information can be found in the request query string under the field name "zip". The plug-in 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 the plug-in knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_pb_purchase-total < "1000.00"'>!TRUE!</xsl:if>
```

The plug-in is designed to know how to handle the extraction of ADI information form the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_pb_
```

The plug-in understands this information can be found in the request POST body under the field name "purchase-total". The plug-in 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_ext()) actually contains the credential information of the user.

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.

The technique for creating extended attributes in the user credential is explained in “Adding extended attributes for credentials” on page 106.

## Supplying a failure reason

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 Security Access Manager authorization rules evaluator, the plug-in 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 the plug-in to reject this standard response and allow denied requests to proceed to a back-end service application.

The request is accompanied by the failure reason provided in the authorization rule. The back-end service application then has the opportunity to proceed with its own response to the situation. This optional configuration in specified using the **pass-on-rule-failure-reason** parameter in the **[boolean-rules]** stanza in the `pdwebpi.conf` file.

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 Security 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 Security 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 credit limit of the user. 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.

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.

## Configuring dynamic ADI retrieval

Rules can be written requiring authorization decision information (ADI) that cannot be found in any of the information that the Security Access Manager authorization service can access.

In these cases, it is necessary to retrieve the ADI from an outside source. The retrieval can be done in real time by a dynamic ADI entitlement retrieval service. The AMWebARS Web service, currently provided with the WebSEAL Attribute Retrieval Service, is one type of entitlement retrieval service.

The Attribute Retrieval Service (ARS) provides communication and format translation services between the plug-in entitlement service library and an external provider of authorization decision information. The following diagram illustrates the process flow for the AMWEBARS Web service:

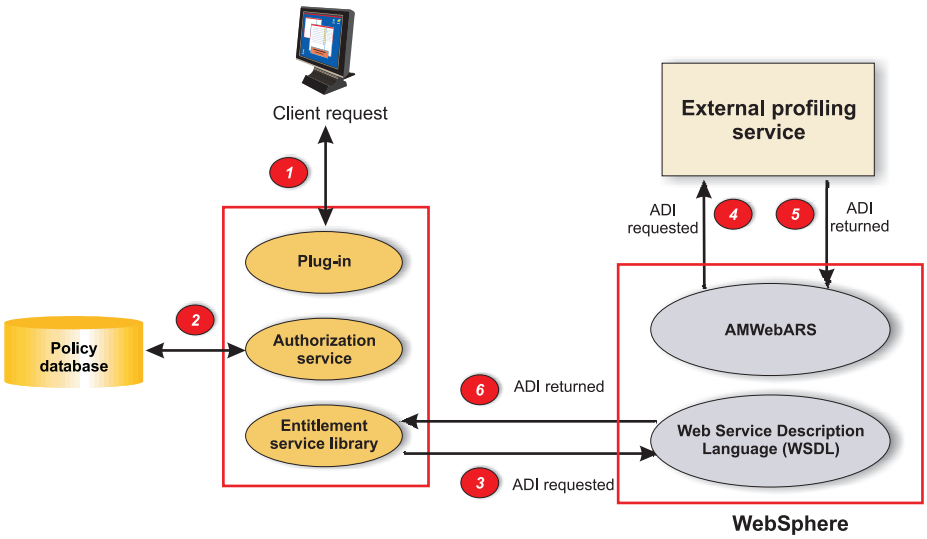


*Figure 15. Attribute retrieval service process flow.*

Process flow:

1. The client makes a request for a protected resource by an authorization rule.
2. The authorization rules evaluator, which is a part of the authorization service, determines that specific authorization decision information (ADI) is required to complete the rule evaluation. The ADI requested is not available from the user credential, the authorization service, or the plug-in.
3. The task of ADI retrieval is sent to the AMWebARS Web service through the entitlement 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 AMWebARS Web service.
4. The AMWebARS Web service formats the request appropriately for the external profiling service that is to provide the ADI.
5. The external profiling service returns the appropriate ADI.
6. The ADI is formatted in another SOAP container and returned to the entitlement service of the plug-in. Now the authorization rules evaluator has the necessary information to evaluate the rule and decide to accept or deny the original client request.

For information about deploying the attribute retrieval service, see the *IBM Security Access Manager: WebSEAL Administration Guide*.

## Configuring the plug-in to use the AMWebARS Web service

Perform the following tasks to configure plug-in to use the AMWebARS Web service.

### Procedure

1. In the `pdwebpi.conf` configuration file, specify the identification name (ID) of the dynamic ADI entitlement retrieval service that is queried when missing ADI is detected during a rules evaluation. In this case, the AMWebARS Web service is specified: `[aznapi-configuration] dynamic-adi-entitlements-services = AMWebARS`
2. In the `pdwebpi.conf` configuration file, use the configured dynamic ADI entitlement retrieval service ID 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] dynADI = azn_ent_amwebars`
3. In the `pdwebpi.conf` configuration file, specify the URL to the dynADI Web service located in the WebSphere environment (entered as one line): `[amwebars] service-url = http://websphere_hostname:websphere_port \ /dynadi/dynadi/ServiceToIServicePortAdapter`
4. Restart the plug-in.

# Appendix A. DynADI Web service reference

This appendix contains the administration and configuration reference for the dynamic authorization decision information (dynADI) Web service.

Topics in this appendix include:



## Basic configuration

The dynADI Web service has the following basic configuration:

### Configuration files

The following dynADI Web service configuration and XML files are located within the working directory of the supporting application. The current implementation of the dynADI Web service is installed in a WebSphere environment:

*websphere_install_directory*/WebSphere?ApServer/bin/

#### dynadi.conf

The `dynadi.conf` configuration file contains parameters and values that specify the general configuration of the DynADI Web service.

#### ContainerDescriptorTable.xml

The `ContainerDescriptorTable.xml` file contains a list of all container descriptors that can be retrieved by the dynADI Web 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 dynADI Web 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 dynadi.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 dynADI providers used by the dynADI Web service. |
| protocol_table_filename | The file name of the ProtocolTable. The ProtocolTable contains information about the different dynADI protocols used by the dynADI Web 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 dynADI Web service's keystore. The keystore is a central storage for all client keys used by the dynADI Web 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 (typically used). 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. |
| trace_verbose_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 dynADI Web 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 dynADI 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". |
| 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 dynADI Web service is configured using different data tables. These tables tell the service, for example, what providers can be accessed, what dynADI 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 dynADI containers
- ProviderTable, which contains the dynADI providers available
- ProtocolTable, which describes the protocols used by the dynADI Web service

## Provider table

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 dynADI Web service.

| | |
|---|---|
| Filename: | `ProviderTable.xml` |
| Format: | XML |
| Table name: | ProviderTable |

| | |
|---|---|
| Element name: | Provider |

### 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:<br>`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_Securtities_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 containters the dynADI Web service can retrieve. You have to add a ContainerDescriptor entry to this table if you want the service to retrieve another type of dynADI container.

| | |
|---|---|
| Filename: | ContainerDescriptor.xml |
| Format: | XML |
| Table name: | ContainerDescriptorTable |
| Element name: | ContainerDescriptor |

## 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 dynADI Web Service. It is generated the following way, if the namespace is present:<br>`container_type_id = namespace_prefix + ": " +container_name`<br><br>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 dynADI 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 DynAdiProtocols. 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 dynADI 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:<br><br>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 DynAdiClients. 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.<br><br>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 DynAdiProvider uses attribute names not compatible with XML element names. Such a mapping is generated the following way:

The key has the structure:

"map_provider_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. 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 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>
  <namespace_prefix>http://ese.erant.com/attributes</namespace_prefix>
  <cost>1 USD</cost>
  <protocol_id>ese_entitlement_protocol</protocol_id>
  <provider_id>Erandt_Securities_Entitlements</provider_id>
  <properties />
  <client_init_properties>
  <property key='map_attribute_name_erandt.com_core_attr_address">
  //erandt.com/attr/address
  </property>
  <property key="map_provider_attribute_name__//erandt.com/attr/address">
  erandt.com_attr_address
  </property>
 </client_init_properties>
 <ContainerPayloadFormat>
 <ese_test_container_address_line>
  <address_line />
  </ese_test_container_address_line>
 </ContainerPayloadformat>
 </ContainerDescriptor>
</ContainerDescriptorTable>
```

## ProtocolTable

The ProtocolTable describes all protocols the dynADI Web service uses. You have to add a Protocol entry to this table if you want the service to retrieve another protocol type.

| | |
|---|---|
| Filename: | ProtocolTabler.xml |
| Format: | XML |
| Table name: | ProtocolTable |
| Element name: | Protocol |

### 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 DynAdiProtocol (required). "Fully qualified" refers to the inclusive package path. |

### Example ProtocolTable

Example for a ProtocolTable with only one protocol:

```
<?xml version="1.0" encoding="UTF-8"?>
 <ProtocolTable>
  <Protocol>
   <protocol_id>file_reader_protocol</protocol_id>
   <class_name>dynadientitlementservice.protocol.FileReaderProtocol</class_name>
  </Protocol>
 </ProtocolTable>
```

## Creating custom protocol plug-ins

You can create custom protocol plug-ins for the dynADI Web service.

### Overview

The dynADI Web 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 dynADI Web 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 dynADI Web 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 dynADI Web service (included with Security 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 dynADI Web 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 dynADI Web 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 dynADI Web 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:

- DynADI class documentation
  `/opt/pdweb/java/dynadi/dynadi_class_doc.zip`
- Example protocol plug-in modules (Java)
  `/opt/pdweb/java/dynadi/protocol_protocol/exampleProtocol.java`
- Compiled (built) version of the example module
  `/opt/pdweb/java/dynadi/protocol_protocol/exampleProtocol.class`
- README file, that explains how to customize and compile the example code
  `/opt/pdweb/java/dynadi/protocol_protocol/README`

# Appendix B. Using pdbackup to backup plug-in data

The **pdbackup** utility allows you to back up and restore Security Access Manager data. The **pdbackup** utility uses, as an argument, a backup list file that specifies the files and directories that require backing up. Each major Security Access Manager component (such as Base, WebSEAL and the plug-in) has its own list file. The `pdinfo-pdwebpi.lst` file specifies the plug-in files and directories that are backed up by the **pdbackup** utility.

This appendix describes how to use the **pdbackup** utility to back up and restore plug-in data. The complete reference for the **pdbackup** utility is located in the *IBM Security Access Manager for Web: Command Reference*.

**Note: On Windows 2008 systems with Tivoli Access Manager 6.0, 6.1, or 6.1.1:** The **pdbackup** utility on Windows 2008 may hang while waiting for user input. If you encounter this issue, use either of the following approaches to continue backing up or restoring your data:

- Type an "A" in the command window. The utility resumes normally.
- Apply the following fix pack for your respective Tivoli Access Manager release, then rerun the **pdbackup** utility:
  - **Tivoli Access Manager 6.0**: Fixpack 28 or later
  - **Tivoli Access Manager 6.1**: Fixpack 08 or later
  - **Tivoli Access Manager 6.1.1**: Fixpack 04 or later

## Functionality

You can back up and restore plugin files and directories.

### Backing up plug-in data

The **pdbackup** utility backs up the list of files and directories contained in the plug-in backup list file, `pdinfo-pdwebpi.lst`.

#### UNIX:

By default, `pdinfo-pdwebpi.lst` is located in `/opt/pdwebpi/etc/`.

By default, the resulting backup archive is stored as a single `.tar` file in `/var/PolicyDirector/pdbackup`.

The default `.tar` file name is constructed using the backup list filename, plus a date and time stamp:

*list-file-name_ddmmyyy.hh_mm*`.tar`

For example:

`pdinfo-pdwebpi.lst_30jul2003.10_39.tar`

Alternatively, you can specify:

- A custom file name for the `.tar` file (use the **–file** option)

  This custom file name does not contain a date and time stamp.

- A custom directory location for the `.tar` file (use the **–path** option)

The contents of the `.tar` file extract into the following directories:

```
opt/ var/ tmp/
```

### Windows:

By default, the `pdinfo-pdwebpi.lst` file is located in `C:\Program Files\Tivoli\PDWebpi\etc\`

By default, the resulting backup archive is stored as a directory tree in the `C:\Program Files\Tivoli\PDWebpi\pdbackup\` directory.

The default `.dar` directory name is constructed from the backup list file name, plus a date and time stamp:

*list-file-name_ddmmmyyyy.hh_mm*`.dar`

For example:

```
pdinfo-pdwebpi.lst_30jul2003.10_39.dar
```

The contents of the `.dar` file extract into a sub-directory and a file:

```
%C%
Registry
```

The `%C%` directory contains the complete backup tree. The name of this directory is determined by the letter designation of the drive where the plug-in files and directories are located. The Registry file stores the registry keys (`.reg` extensions).

Alternatively, you can specify:

- A custom file name for the `.dar` directory archive (use the **–file** option)
  This custom file name does not contain a date and time stamp.
- A custom directory location for the `.dar` directory archive (use the **–path** option)

## Restoring plug-in data

Locations for restored plug-in data depend on the operating system.

### UNIX:

Archived files and directories are restored from the `.tar` file to the `/opt/pdwebpi` directory.

### Windows:

Archived files are restored to their original installation directory locations.

# Syntax

A complete reference for the **pdbackup** utility can be found in the *IBM Security Access Manager for Web: Command Reference*.

**pdbackup –a backup –l** *backup-list-pathname* ****
**[–path** *custom-pathname*]**[–file** *archive-pathname*] **[–usage] [–?]**

**pdbackup –a restore –file** *archive-pathname* \
**[–path** *custom-pathname*] **[–usage] [–?]**

| Option | Description |
|---|---|
| –a [backup \| restore \| extract] | Specifies backup, restore, or extract operation. |
| –l *backup-list-pathname* | Specifies the fully qualified path to the backup list file (`pdinfo-pdwebpi.lst`). |
| –path *custom-pathname* | Used for backup, specifies a custom archive directory location. |
| –file *archive-pathname* | Used for backup, specifies a custom name for the archive file.<br><br>Used for restore, specifies the fully qualified path to the archive file that is to be restored. |

You can use short versions of the command option names, but the abbreviation must be unambiguous. For example, you can type `a` for **action**. However, values for arguments to these options cannot be abbreviated.

# Examples

## UNIX examples

1. The following example performs a standard back up with default values:
   ```
   pdbackup -a backup -l /opt/pdwebpi/etc/pdinfo-pdwebpi.lst
   ```
   This results in a file named `pdinfo-pdwebpi.lst_`*`date.time`*`.tar` that is stored in the `/var/PolicyDirector/pdbackup` directory.
2. The following example performs a back up and stores the default archive file in the `/var/backup` directory:
   ```
   pdbackup -a backup -l /opt/pdwebpi/etc/pdinfo-pdwebpi.lst -path /var/backup
   ```
   This results in a file named `pdinfo-pdwebpi.lst_`*`date.time`*`.tar` located in the `/var/pdbackup` directory.
3. The following example performs a back up and creates an archive file named `amwebarchive.tar`:
   ```
   pdbackup -a backup -l /opt/pdwebpi/etc/pdinfo-pdwebpi.lst -file amwebarchive
   ```
   The default archive extension (`.tar`) is appended to the custom `amwebarchive` file name. This file is stored in the default `/var/PolicyDirector/pdbackup` directory.
4. The following example restores the archive file from the default directory location:
   ```
   pdbackup -a restore -file pdinfo-pdwebpi.lst_29Aug2003.07_24.tar
   ```
5. The following example restores the archive file from the `/var/pdback` directory:
   ```
   pdbackup -a restore -file /var/pdback/pdinfo-pdwebpi.lst_29Aug2003.07_25.tar
   ```

## Windows examples

1. The following example performs a standard backup with default values:
   ```
   pdbackup -a backup -l install_path\etc\pdinfo-pdwebpi.lst
   ```

This results in a file named pdinfo-pdwebpi.lst_*date.time*.dar located in the plug-in *install_path*\pdbackup directory.

2. The following example performs a back up using the default archive file name and stores the file in the C:\pdback directory:

   ```
   pdbackup -a backup -l install_path\etc\pdinfo-pdwebpi.lst -path c:\pdback
   ```

3. The following example performs a back up and creates a file named pdarchive.dar:

   ```
   pdbackup -a backup -l install_path\etc\pdinfo-pdwebpi.lst -file pdarchive
   ```

   The default archive extension (.dar) is applied to the custom pdarchive file name. The file is stored in the default *install_path*\pdbackup directory.

4. The following example performs a back up to the \pdback directory on the F: drive:

   ```
   pdbackup -a backup -l pdinfo-pdwebpi.lst -path f:\pdback
   ```

5. The following example restores the archive file from the default directory (*single directory shown over two lines*):

   ```
   pdbackup -a restore -file install_path\pdbackup
   \pdinfo-pdwebpi.lst_29Jun2003.07_24.dar
   ```

6. The following example restores files from the H:\pdbackup directory:

   ```
   pdbackup -a restore -file h:\pdbackup\pdinfo-pdwebpi.lst_29Jun2003.07_25.dar
   ```

## Contents of pdinfo-pdwebpi.lst

```
[UNIX FILES]
# fully qualified file names
./opt/pdwebpi/etc
./var/pdwebpi/audit
./var/pdwebpi/db
./var/pdwebpi/keytab
./var/pdwebpi/log

[UNIX CONF FILES]
# configuration files that specify a file to include
# file:stanza:option
/opt/pdwebpi/etc/pdwebpi.conf:uraf-ad:ad-server-config
/opt/pdwebpi/etc/pdwebpi.conf:ldap:ldap-server-config
/opt/pdwebpi/etc/pdwebpi.conf:ldap:ssl-keyfile
/opt/pdwebpi/etc/pdwebpi.conf:ldap:ssl-keyfile-stash
/opt/pdwebpi/etc/pdwebpi.conf:failover:failover-cookies-keyfile
/opt/pdwebpi/etc/pdwebpi.conf:ltpa:ltpa-keyfile
/opt/pdwebpi/etc/pdwebpi.conf:ltpa:ltpa-stash-file
/opt/pdwebpi/etc/pdwebpi.conf:iis:query-log-file
/opt/pdwebpi/etc/pdwebpi.conf:iis:log-file
/opt/pdwebpi/etc/pdwebpi.conf:iplanet:query-log-file

[WINDOWS FILES]
BASEDIR=SOFTWARE\Tivoli\Access Manager Plug-in for Web Servers:Path
<BASEDIR>etc
<BASEDIR>log
<BASEDIR>audit
<BASEDIR>db
<BASEDIR>keytab

[WINDOWS CONF FILES]
# configuration files that specify a file to include
# file:stanza:option
<BASEDIR>etc/pdwebpi.conf:uraf-ad:ad-server-config
<BASEDIR>etc/pdwebpi.conf:ldap:ldap-server-config
<BASEDIR>etc/pdwebpi.conf:ldap:ssl-keyfile
<BASEDIR>etc/pdwebpi.conf:ldap:ssl-keyfile-stash
<BASEDIR>etc/pdwebpi.conf:failover:failover-cookies-keyfile
<BASEDIR>etc/pdwebpi.conf:ltpa:ltpa-keyfile
```

```
<BASEDIR>etc/pdwebpi.conf:ltpa:ltpa-stash-file
<BASEDIR>etc/pdwebpi.conf:iis:query-log-file
<BASEDIR>etc/pdwebpi.conf:iis:log-file
<BASEDIR>etc/pdwebpi.conf:iplanet:query-log-file

[WINDOWS REGISTRY]
# specify keys to backup
SOFTWARE\Tivoli
```

## Additional backup data

The following stanzas and parameters are not listed in the **pdinfo-pdwebpi.lst** file and are therefore not automatically backed up. If you require this data to be backed up, you must edit the **pdinfo-pdwebpi.lst** and add this information. Follow the format described at the beginning of the **pdinfo-pdwebpi.lst** file.

```
[cdsso-domain-keys]
<domain name> = <key file>

[ecsso-domain-keys]
<domain name> = <key file>
```

# Appendix C. Plug-in configuration file reference

IBM Security Access Manager Plug-in for Web Servers is controlled through the configuration of parameters located in the `pdwebpi.conf` configuration file. The configuration file is located in the `etc` directory in the default installation location.

The configuration file contains sections or stanzas under which parameters for controlling similar plug-in functionality are grouped. Stanzas appear within brackets, for example, `[performance]`. The parameters grouped under stanzas take the basic form, *key* = *value*, though their actual implementation varies greatly depending on what they are representing. This appendix is a reference to the various stanzas and parameters available for configuring plug-in operation.

## Guidelines for configuring stanzas

These guidelines are provided to help you make changes to the IBM Security Access Manager Plug-in for Web Servers configuration file.

The guidelines are divided into these types:

- General guidelines
- Default values
- Strings
- Defined strings
- File names
- Integers
- Boolean values

### 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, the entry must contain a valid key and value.
- Do not change the names of the keys in the configuration files. Changing the name of the key might cause unpredictable results for the servers.
- Stanza entries and key names are case-sensitive. For example, `auth-data` and `Auth-Data` are treated as different entries.
- 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 typically used.
- Non-printable characters (such as tabs, carriage returns, and line feeds) that occur at the end of a stanza entry are ignored. Non-printable characters are ASCII characters with a decimal value less than 32.

## Default values

Use the following guidelines when changing default configuration settings:

- Many values are created or modified only by using configuration programs. Do not manually edit these stanzas or values.
- Some values are filled in automatically during 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 key value pairs are not applicable to certain servers 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 code set.
- Additional or different restrictions on the set of allowable string characters might be imposed. For example, many strings are restricted to ASCII characters. Consult each stanza entry description for any restrictions.
- Double quotation marks are sometimes, but not always, required when you use spaces or more than one word 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 `[iv-headers]` stanza contains the entry, `accept`, where the valid values are: *all*, *iv-creds*, *iv-user*, *iv-user-l* or *iv-remote-address*. The value for `accept` is expected to one of these. Any other value is invalid and results in an error.

## 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:

- Filename

  No directory path included.
- Relative filename

  A directory path is allowed but not mandatory.

  These files typically are expected to be located relative to the location of a standard plug-in directory. The stanza entry for each relative path name 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 can be determined by the file system and by the local code set. For Windows, file names cannot have these characters: a backward slash (\), 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 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.

  For example, in the **[aznapi-configuration]** stanza, the entry `logsize = 0` means the log files do not rollover to a new file.
- For some entries requiring integer values, the plug-in does not impose an upper limit for the maximum number allowed. For such an entry, the maximum number is limited only by the size of memory allocated for an integer data type. This number can vary, based on the type of operating system. For systems that allocate 4 bytes for an integer, this value is 2147483647.

  However, as the administrator, use a number that represents the value that is most logical for the value you are trying to set.

## Boolean values

Many stanza entries represent a Boolean value. The plug-in recognizes the Boolean values `yes` and `no`.

Some of the entries in the configuration files 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`

Anything other than `yes|true`, including a blank value, will be interpreted as `no|false`.

The recognized Boolean entries are listed for each stanza entry. Refer to the individual descriptions to determine when `true` or `false` are also recognized.

# [acctmgmt]

This stanza contains entries for user account management operations such as changing user passwords, and logging out.

The values can be either:

- A macro HTML file located on the translated plug-in HTML directory, *install_path* `/nls/html/`*lang* */charset*
- A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.

See “Macro support” on page 9 for information on plug-in supported macros.

| **[acctmgmt] stanza** | |
|---|---|
| help-page = *help filename* | |
| | The file name of the page displayed when a user requests help.<br><br>Default: help.html. This file is part of the installed product and can be edited to suit your requirements.<br><br>A valid redirect URI can be specified. For example: help-page = http://www.organization.com/TAM/help/help_index.html |
| logout-success = *logout success filename* | |
| | The file name of the page displayed when a user successfully logs out.<br><br>Default value: logout_success.html. This file is part of the installed product and can be edited to suit your requirements.<br><br>A valid redirect URI can be specified. For example: logout-success = http://www.organization.com/TAM/logout/after_logout.html |
| password-change-form = *password change filename* | |
| | The file name of the page displayed when a user requests to have their password changed.<br><br>Default value: password_change.html. This file is part of the installed product and can be edited to suit your requirements.<br><br>A valid redirect URI can be specified. For example: password-change-form = http://www.organization.com/TAM/password/password_utils.html |
| password-change-success = *password change success filename* | |
| | The page displayed when a user completes a change of password successfully.<br><br>Default value: password_change_success.html. This file is part of the installed product and can be edited to suit your requirements.<br><br>A valid redirect URI can be specified. For example: password-change-success = http://www.organization.com/TAM/password/success.html |
| password-change-failure = *password change failure filename* | |
| | The page displayed to inform the user their password change request failed.<br><br>Default value: password_change_failure.html. This file is part of the installed product and can be edited to suit your requirements.<br><br>A valid redirect URI can be specified. For example: password-change-failure = http://www.organization.com/TAM/password/failure.html |
| password-change-form-uri = *password change URI* | |
| | The URI accessed when a user requests a change of password.<br><br>Default value: /pkmspasswd.form |
| password-change-uri = *URI post password change* | |
| | The URI destination after a password change.<br><br>Default value: /pkmspasswd.form |
| logout-uri = *destination URI after logout* | |
| | The URI destination after user logout.<br><br>Default value: /pkmslogout |
| help-uri = *help page URI* | |

| | |
|---|---|
| | The location of the help page.<br><br>Default value: `/pkmshelp` |

## [apache]

Configuration parameters specific to the Security Access Manager Plug-in for Apache.

| **[apache] stanza** | |
|---|---|
| `query-contents = query contents program` | |
| | Specifies the query contents program to use for browsing the Apache Web space by the **pdadmin> object list** command.<br><br>Default value: `/opt/pdwebpi/bin/wpi_apache_ls` |
| `query-log-file = path to query log file` | |
| | Location of log file for errors encountered by the query contents program.<br><br>Default value: *install-path*`/log/msg__pdwebpi-ls.log` |
| `doc-root = apache-branch-doc-root` | |
| | Specifies the documentation root that provides the Web space browse capability needed for performing **pdadmin> object list** commands. This parameter is set by the configuration utility when setting up virtual hosts. It is specified on a per-policy branch basis in an `[apache:`*branch*`]` stanza, for example, `[apache:/PDWebPI/lotus.com]`<br><br>There is no default value. |

## [auth-data]

Contains one or more entries of the form: *name* = *method*:*value*. This parameter is located in the stanza specified by the **argument-stanza** parameter.

**name** matches the name attribute of an input element within the login form.

**method** is either cred, gso, or string.

**value** contains a string interpreted according to the method value.

| **[auth-data] stanza** |
|---|
| *name = method*:*value* |

| | |
|---|---|
| | The value of the name parameter is set to the value of the name attribute of the HTML input tag. For example: `<input name=uid type=text>Username</input>`<br><br>The value can also use the value of the name attribute of the HTML *select* or *textarea* tags.<br><br>The *method*:*value* combination retrieves the authentication data required by the form.<br><br>Default:<br>`userid = cred:AZN_CRED_PRINCIPAL_NAME`<br>`pin = string:password`<br>`#userid = gso:username`<br>`#pin = gso:password`<br><br>Example: `uid =string:brian` |

## [authentication-levels]

IBM Security Access Manager Plug-in for Web Servers supports a variety of authentication methods. You configure the authentication methods for use with your installation in the **[common-modules]** stanza. The **[authentication-levels]** stanza defines the ordering and step-up authentication levels for the authentication methods defined in the **[common-modules]** stanza. If an authentication module is not given a level within the **[authentication-levels]** stanza it will default to an authentication level of 1.

The format of the entries in this stanza is, *level* = *module_name*. Authentication order is determined as the highest authentication level down to the lowest authentication level for the authentication methods defined. If an authentication level is shared by several authentication methods the sub-order is determined by the order in which the modules appear within the **[modules]** stanza.

| **[authentication-levels] stanza** | |
|---|---|
| *level = module-name* | |
| | There are no default values configured.<br><br>Example:<br>`1 = BA`<br>`2 = forms` |

## [authentication-mechanisms]

Plug-in authentication modules that are defined in the **[modules]** stanza, extract authentication information from requests. The actual authentication of requests is performed by authentication mechanisms which validate authentication information. This separation of roles allows custom external authentication mechanism libraries written for WebSEAL to be used with the plug-in.

This stanza lists the mechanisms for all authentication methods supported by the plug-in. Uncomment the line and supply the full path to a library to enable a mechanism.

| **[authentication-mechanisms] stanza** |
|---|

| | |
|---|---|
| `passwd-cdas = passwd-cdas-library` | |
| | Fully qualified path for the custom shared library for external authentication mechanisms that implements client access with username and password for a third-party registry.<br><br>There is no default. |
| `passwd-ldap = passwd-ldap-library` | |
| | Fully qualified path for a local built-in authenticator that implements client access with LDAP username and password.<br><br>There is no default. |
| `passwd-uraf = uraf-authn-library` | |
| | Fully qualified path for a library that implements basic authentication using the Security Access Manager URAF interface to underlying user registry types.<br><br>There is no default. |
| `token-cdas = token-cdas-library` | |
| | Fully qualified path for the custom shared library for external authentication mechanisms that implements client access using the LDAP username and token passcode.<br><br>There is no default. |
| `cert-ssl = cert-ssl-library` | |
| | Fully qualified path for the local built-in authenticator that implements client access using a client-side certificate over SSL.<br><br>There is no default. |
| `http-request = http-request-library` | |
| | Fully qualified path for the local built-in authenticator that implements client access using a special HTTP header, IP address, or IV Header with `iv-remote-address` activated.<br><br>There is no default. |
| `sso-create = sso-creation-library` | |
| | Used for CDSSO and e-Community SSO.<br><br>There is no default.<br><br>Example:<br><br>AIX: `sso-create = /opt/pdwebrte/lib/ibssocreate.a`<br><br>Other UNIX: `sso-create = /opt/pdwebrte/lib/ibssocreate.so`<br><br>Windows: `sso-create = C:\Program Files\Tivoli\PDWebRTE\bin\ssocreate.dll` |
| `sso-consume = sso-consumption-library` | |

| | |
|---|---|
| | Used for CDSSO and e-Community SSO.<br><br>There is no default.<br><br>Example:<br><br>AIX: `sso-consume =/opt/pdwebrte/lib/libssoconsume.a`<br><br>Other UNIX: `sso-consume =/opt/pdwebrte/lib/libssoconsume.so`<br><br>Windows: `sso-create = C:\Program Files\Tivoli\PDWebRTE\bin\ ssoconsume.dll` |
| `kerberosv5 = stli-authn-library` | |
| | Required when enabling SPENGO authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `kerberosv5 = /opt/PolicyDirector/lib/libstliauthn.a`<br><br>Other UNIX: `kerberosv5 = /opt/PolicyDirector/lib/libstliauthn.so`<br><br>Windows: `kerberosv5 = C:\PROGRA~1\Tivoli\POLICY~1 \bin\stliauthn.dll` |
| `passwd-strength = passwd-strength-library` | |
| | The fully qualified path for the library that checks new passwords entered on the password change form.<br><br>There is no default. |
| `cred-ext-attrs = cred-ext-attrs-library` | |
| | The fully qualified path for the library that allows custom attributes (name/value pairs) to be specified for inclusion in the credential.<br><br>There is no default. |
| `su-password = su-password-library` | |
| | The fully qualified path to the custom external authentication mechanism for switch user<br><br>There is no default.<br><br>Example:<br><br>AIX: `su-password = /opt/PolicyDirector/lib/libsuformauthn.a`<br><br>Other UNIX: `su-password = /opt/PolicyDirector/lib/libsuformauthn.so`<br><br>Windows: `su-password = C:\PROGRA~1\Tivoli\POLICY~1 \bin\suformauthn.dll`<br><br>See “Configuring switch user (SU) for administrators” on page 24. |
| `su-token-card = su-token-card-library` | |

| | |
|---|---|
| | Fully qualified path for a library that implements switch user authentication for token authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `kerberosv5 = /opt/PolicyDirector/lib/libsucustom.a`<br><br>Other UNIX: `kerberosv5 = /opt/PolicyDirector/lib/libsucustom.so`<br><br>Windows: `kerberosv5 = C:\PROGRA~1\Tivoli\POLICY~1\bin\sucustom.dll`<br><br>See "Configuring switch user (SU) for administrators" on page 24. |
| `su-certificate = su-certificate-library` | |
| | Fully qualified path for a library that implements switch user authentication for certificate authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `kerberosv5 = /opt/PolicyDirector/lib/libsucert.a`<br><br>Other UNIX: `kerberosv5 = /opt/PolicyDirector/lib/libsucert.so`<br><br>Windows: `kerberosv5 = C:\PROGRA~1\Tivoli\POLICY~1\bin\sucert.dll`<br><br>See "Configuring switch user (SU) for administrators" on page 24. |
| `su-http-request = su-http-request-library` | |
| | Fully qualified path for a library that implements switch user authentication for HTTP header or IP address authentication.<br><br>There is no default.<br><br>See "Configuring switch user (SU) for administrators" on page 24. |
| `su-cdsso = su-cdsso-library` | |
| | Fully qualified path for a library that implements switch user authentication for cross-domain single sign-on authentication.<br><br>There is no default.<br><br>See "Configuring switch user (SU) for administrators" on page 24. |
| `su-kerberosv5 = failover-kerberosv5-library` | |
| | Fully qualified path for a library that implements switch user authentication for SPNEGO (Kerberos) authentication.<br><br>There is no default.<br><br>See "Configuring switch user (SU) for administrators" on page 24. |
| `failover-password = failover-password-library` | |

| | |
|---|---|
| | Fully qualified path to the library that implements failover cookie authentication for token card authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-password = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-password = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-password = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `failover-token-card = failover-token-card-library` | |
| | Fully qualified path to the library that implements failover cookie authentication for token card authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-token-card = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-token-card = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-token-card = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `failover-certificate = failover-certificate-library` | |
| | Fully qualified path to the library that implements failover cookie authentication for certificate authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-certificate = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-certificate = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-certificate = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `failover-http-request = failover-http-request-library` | |

| | |
|---|---|
| | Fully qualified path to the library that implements failover cookie authentication for HTTP header authentication or IP address authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-http-request = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-http-request = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-http-request = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `failover-cdsso = failover-cdsso-library` | |
| | Fully qualified path to the library that implements failover cookie authentication for cross-domain single sign-on authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-cdsso = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-cdsso = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-cdsso = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `failover-kerberosv5 = failover-kerberosv5-library` | |
| | Fully qualified path for a library that implements failover cookie authentication for SPNEGO authentication.<br><br>There is no default.<br><br>Example:<br><br>AIX: `failover-kerberosv5 = /opt/pdweb/lib/libfailoverauthn.a`<br><br>Other UNIX: `failover-kerberosv5 = /opt/pdweb/lib/libfailoverauthn.so`<br><br>Windows: `failover-kerberosv5 = C:\PROGRA~1\Tivoli\PDWebRTE\bin\ failoverauthn.dll`<br><br>See "Configuring failover authentication" on page 81. |
| `post-pwdchg-process = post-pwdchg-process-library` | |
| | Fully qualified path for a library that implements post password change processing. This is called by WebSEAL when the user changes a password using the **pkms** password change page.<br><br>There is no default.<br><br>Example:<br><br>AIX: `post-pwdchg-process = /opt/PolicyDirector/lib/reg2update.a`<br><br>Other UNIX: `post-pwdchg-process = /opt/PolicyDirector/lib/reg2update.so`<br><br>Windows: `post-pwdchg-process = C:\PROGRA~1\Tivoli\POLICY~1\bin\ reg2update.dll` |

## [aznapi-configuration]

This stanza contains entries for plug-in auditing and cache database configuration.

The auditing configured in this stanza is different to the status and error message logging configured in the **[pdweb-plugins]** stanza. The auditing configured in the **[aznapi-configuration]** stanza is for the capture of authentication (authn) and authorization (azn) events.

| **[aznapi-configuration] stanza** | |
|---|---|
| `logsize = log size in bytes` | |
| | The size in bytes that triggers the creation of a new audit log file. If set to 0, a new log file is not created. If set to a negative integer, a new log file is created daily regardless of size.<br><br>Default value: 2000000 |
| `logflush = flush interval in seconds` | |
| | The interval in seconds at which the audit logs are flushed. The maximum value is 21600 (6 hours).<br><br>Default value: 20 |
| `logaudit = {yes\|no}` | |
| | Enable or disable audit logging for the authorization API.<br><br>Default value: no |
| `auditlog = audit log file` | |
| | The name of the audit log file.<br><br>Default value: `audit.log` |
| `auditcfg = {authn\|azn\|wpi}` | |
| | Enable or disable one or more component-specific audit records. This entry is required when auditing is enabled (logaudit = yes). The valid values are:<br><br>*authn* - Capture authentication events.<br><br>*azn* - Capture authorization events.<br><br>*wpi* - Plug-in specific authentication events.<br><br>Default value: azn<br><br>Example:<br>`auditcfg = azn`<br>`auditcfg = authn`<br>`auditcfg = wpi` |
| `db-file = path to cache file` | |
| | Full path to the ACL database cache file.<br><br>This is not set by default.<br><br>Example: `db-file =/var/pdwebpi/db/pdwebpi.db` |
| `cache-refresh-interval = {disable\|default\|number of seconds}` | |

| | |
|---|---|
| | The interval, in seconds, between checks for updates to the master authorization server. The local cache is only rebuilt if an update is detected.<br><br>Valid values are:<br>**disable** The interval value in seconds is not set.<br>**default** An interval of 600 seconds is used.<br>*positive integer* The number of seconds between each check of updates to the master authorization server.<br><br>Default value: `disable`<br><br>Example: `cache-refresh-interval = 60` |
| `listen-flags = {enable\|disable}` | |
| | Enables or disables the reception of policy cache update notifications from the mast authorization server. A value of `enable` activates the notification listener. Set to `disable`, the notification listener is deactivated. This parameter is set by the **svrsslcfg** utility.<br><br>Default value: `disable`<br><br>Example: `listen-flags = enable` |
| `policy-cache-size = max number of cache entries` | |
| | The maximum size in memory of the policy cache. The policy cache stores: policies, the relationships between policies and resources, and indications that a resource has no directly associated policy.<br><br>When setting the maximum cache size you should consider the number of policy objects defined, the number of resources protected, and the available memory.<br><br>A simple algorithm to begin with is:<br>(number of policy objects * 3) + (number of protected resources * 3).<br>This value controls how much information is cached.<br><br>A larger cache can improve application performance but requires greater memory. Size is specified as the number of entries.<br><br>Default value: The entry is commented by default.<br><br>Example: `policy-cache-size = 32768` |
| `resource-manager-provided-adi = ADI retrieval prefix` | |
| | Prefixes for ADI retrieval from the HTTP request. These prefixes cannot be changed or configured in any way.<br><br>Default value:<br>`resource-manager-provided-adi = AMWS_qs_`<br>`resource-manager-provided-adi = AMWS_hd_`<br>`resource-manager-provided-adi = AMWS_pb_`<br><br>See Chapter 9, “Authorization decision information retrieval,” on page 191. |
| `input-adi-xml-prolog = prolog` | |

| | |
|---|---|
| | This entry specifies the prolog that is added to the top of the XML document created using the Access Decision Information (ADI) needed to evaluate a Boolean authorization rule. If not specified then the default XML prolog is used, `<?xml version='1.0' encoding='UTF-8'?>`.<br><br>You should read and understand the Boolean authorization rules documentation before attempting to change this setting from the default.<br><br>Default value: The entry is commented by default.<br><br>Example: `input-adi-xml-prolog = <?xml version='1.0' encoding='UTF-8'?>` |
| `xsl-stylesheet-prolog = prolog` | |
| | This entry specifies the prolog to be added to the top of the XSL stylesheet created using the XSL text that defines a boolean authorization rule. If not specified then the default XSL stylesheet prolog is: `<?xml version='1.0' encoding='UTF-8'?> <xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transfo rm' version='1.0'> <xsl:output method = 'text' omit-xml-declaration='yes' indent='no'/> <xsl:template match='tex t()'> </xsl:tem plate>`<br><br>You should read and thoroughly understand the boolean authorization rules documentation before attempting to change this setting from the default.<br><br>Default value: This entry is commented by default.<br><br>Example: `xsl-stylesheet-prolog = <?xml version='1.0' encoding='UTF-8'?> <xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transfo rm' version='1.0'> <xsl:output method = 'text' omit-xml-declaration='yes' indent='no'/> <xsl:template match='tex t()'> </xsl:tem plate>` |
| `dynamic-adi-entitlement-services = service ID of DynADI Entitlement Service` | |
| | A list of configured entitlements service IDs queried by the rules engine when missing ADI is detected during an authorization rule evaluation.<br><br>The entries must adhere to the input and output specifications for a dynamic ADI retrieval service. These are outlined in the *Authorization Programmer's Guide*.<br><br>The entries must refer to existing entitlement services that were loaded using entries in the **[aznapi-entitlement-services]** stanza or initialization attribute.<br><br>There is no default for this entry.<br><br>Example:<br>`dynamic-adi-entitlement-services = dynADI_A`<br>`dynamic-adi-entitlement-services = dynADI_B` |
| `cred-attribute-entitlement-services = service ID` | |
| | A list of configured entitlement service IDs called during credential creation to gather attributes for insertion into the credential. The `azn_ent_cred_attrs` entitlement service can be used here to retrieve attributes from the user registry. The entries must refer to existing entitlement service IDs loaded using service entries in the **[aznapi-entitlement-services]** stanza or using the initialization attribute list.<br><br>There is no default for this entry.<br><br>Example:<br>`cred-attribute-entitlement-services = TAM_CRED_ATTRS_SVC` |
| `stats = component [interval [count]] [logagent]` | |

| | |
|---|---|
| | Enables the recording of plug-in statistics.<br><br>*component*<br>The statistics component name. Statistics are gather for this plug-in component. This is a mandatory parameter. Statistics for this component can also be recorded in a log file by specifying the optional arguments to this command.<br><br>*interval*<br>The time interval between reports of information. This argument is optional and results in statistics being sent to a log file. When this option is specified, statistics are sent, by default, the plug-in log file. You can specify another output location using the *logagent* argument. If *interval* is not specified, no statistics are sent to any log file. However, the statistic component is still enabled. You can obtain reports dynamically at any time using **pdadmin stats get**.<br><br>*count*<br>The number of reports sent to a log file. This argument is optional and requires that the *interval* argument be specified. If *interval* is specified without *count*, the duration of reporting is indefinite. After the count value is reached, reporting to a log file stops. However, the statistic component is still enabled. You can obtain reports dynamically at any time using **pdadmin stats get**.<br><br>*logagent*<br>Optionally specifies a destination for the statistics information gathered for the specified component.<br><br>There is no default for this entry.<br><br>Example:<br>`stats = pdwebpi.authn 40` |
| `trace = component [level] [logagent]` | |
| | Enables the recording of plug-in tracing.<br><br>*component*<br>The trace component name. Tracing is recorded for this plug-in component. This is a mandatory parameter. Statistics for this component can also be recorded in a log file by specifying the optional arguments to this command.<br><br>*level*<br>Optionally specifies the tracing level.<br><br>*logagent*<br>Optionally specifies a destination for the statistics information gathered for the specified component.<br><br>There is no default for this entry.<br><br>Example:<br>`trace = pdwebpi.authn 7` |
| `logcfg = stats.component:destination` | |

| | |
|---|---|
| | Directs the statistics gathered by the **stats** configuration entry and directs the results to a specific destination.<br><br>*component*<br>The statistics component name. Statistics are gather for this plug-in component. This is a mandatory parameter. Statistics for this component can also be recorded in a log file by specifying the optional arguments to this command.<br><br>*destination*<br>The destination file for statistics gathering.<br><br>There is no default for this entry.<br><br>Example:<br>`logcfg = stats.pdwebpi.authn:file path=/tmp/authn.log,rollover_size=-1,flush=20` |
| `xsl-max-logical-expressions=`*`numeric value`* | |
| | Limits the number of logical expressions allowed in a rule to a specific numeric value.<br><br>The default is zero, which means no limit.<br><br>This option is dependent on the `xsl-eval-expressions-check` option. |
| `xsl-eval-expressions-check=TRUE\|FALSE` | |
| | Causes the number of logical expressions in a rule to be checked when the rule is evaluated.<br><br>Because this check occurs at runtime, performance can be impacted.<br><br>If this option is set to `TRUE`, the `xsl-max-logical-expressions` option must be set to a value greater than zero in order for the rule to be checked.<br><br>The default value is FALSE. |

## [aznapi-entitlement-services]

aznAPI service definitions. For more information refer to the *Authorization API Programmer's Guide*.

| **[aznapi-entitlement-services] stanza** | |
|---|---|
| *service-id* = *path-to-dll* [& *params*...} | |
| | The *service ID* is the string used by the aznAPI client to identify the service. The *path-to-dll* is the fully qualified path to the DLL containing the service executable code. Parameters can be specified that are passed to the service when it is initialized by the aznAPI. Parameters are specified following the '&' symbol.<br><br>Default value: `AZN_ENT_EXT_ATTR = azn_ent_ext_attr`<br><br>Example: `dynADI = azn_ent_amwebars` |

# [BA]

The stanza where all configuration parameters related to Basic Authentication are gathered.

| **[BA] stanza** | |
|---|---|
| basic-auth-realm = "*realm name*" | |
| | The Basic Authentication realm name. Realm names should be enclosed in double quotes. The realm name is displayed in the authentication dialog.<br><br>Default value: "Access Manager"<br><br>Example: basic-auth-realm = "Access Manager" |

| strip-hdr = {ignore\|always\|unauth} | |
|---|---|
| | This entry controls the removal of the BA header from the request.The valid options are:<br>**ignore**: Do nothing to the BA header.<br>**always**: Strip the BA header from the request.<br>**unauth**: Strip the BA header from the request if the plug-in did not authenticate the header<br><br>Default value: always<br><br>Example: strip-hdr = always |
| add-hdr = {none\|gso\|supply} | |
| | This entry controls the addition of a new BA header to requests. The valid options are:<br>**none**: Do not add a new BA header to the request.<br>**gso**: Add a GSO BA header to the request.<br>**supply**: Supply a static password or a username, or both a username and password in the BA header.<br><br>When **gso** is specified, the gso-resource-name entry must be set.<br><br>When **supply** is set, the supply-password entry must be set with the supply-username entry being optional.<br><br>Default value: none<br><br>Example: add-hdr = gso |
| gso-resource-name = *name of GSO resource* | |
| | Contains the name of the GSO resource used to create GSO BA headers. Specifying a value is optional when add-hdr is set to **gso**. When add-hdr is set to **gso** and gso-resource-name is not set, the name of the virtual host handling the request is used.<br><br>There is no default value for this entry.<br><br>Example: gso-resource-name = citrixres01 |
| supply-password = *BA password* | |
| | A value is required here if add-hdr is set to **supply**. When set, the parameter specifies the static password used in the created BA header.<br><br>There is no default value for this entry. |
| supply-username = *BA user name* | |

| | |
|---|---|
| | Contains the static user name used in the created BA header. The setting of this parameter is optional when the `add-hdr` parameter is set to **supply**. When the `add-hdr` parameter is set to **supply** and `supply-username` has not been set (that is, it remains commented out) the name of the authenticated user is used in the created BA header.<br><br>There is no default value for this entry.<br><br>Example: |
| `use-utf8 = {true\|false}` | |
| | This entry defines what character set to use to encode the generated BA header. If the value is set to **false** the generated BA header is encoded in the local code page.<br><br>Default value: `true` |

## [boolean-rules]

This stanza contains an entry used to affect the outcome of access decisions.

| **[boolean-rules] stanza** | |
|---|---|
| `pass-on-rule-failure-reason = {true \| false}` | |
| | This entry either allows or denies failed authorizations to proceed to the requested backend application accompanied by a reason why the authorization failed. The back-end application then has the opportunity to proceed with its own response to the situation.<br><br>Set to the default **false**, or absent, failed authorization decisions will result in the request being denied.<br><br>Set to **true**, if an authorization fails and a failure reason string associated with the rule object in the policy database exists, then access proceeds. The failure reason string is inserted into the HTTP request in the AM_AZN_FAILURE header.<br><br>Default value: `false`<br><br>Example: `pass-on-rule-failure-reason = true` |

## [cdsso]

Cross-domain single sign-on (CDSSO) is a method for transferring user credentials across multiple secure domains. CDSSO is configured in the stanzas: **[cdsso], [cdsso-token-attributes], [cdsso-incoming-attributes]** and **[cdsso-domain-keys]**. CDSSO needs to be configured as an authentication module for CDSSO functionality to take effect.

| **[cdsso] stanza** |
|---|
| `uri = cdsso redirection uri` |

| | |
|---|---|
| | The special URI tag that indicates a CDSSO redirection and single sign-on. When a user makes a request to access a resource in another domain using a custom link on a Web page, the expression specified by the `uri` entry indicates it is a CDSSO request.<br><br>Therefore using the default, a requested URI might look like, `/pkmscdsso?https://www.domainB.com/index.html`<br><br>Default value: `/pkmscdsso`<br><br>Example: `uri = /pkmscdsso` |
| `cdsso-argument = authentication token query string` | |
| | The name of the query string argument that specifies the authentication token. This is used in the re-direction URI. For example: *remote-uri*`?PD-REFERER=`*this-host*`&PD-ID=`*authentication-token*<br><br>When using the default SSO create and consume modules, (defined in the **[authentication-mechanisms]** stanza), the domain name of the host that generated the URI is used to locate the key to decrypt the token.<br><br>Default value: PD-ID<br><br>Example: `cdsso-argument = PD-ID` |
| `authtoken-lifetime = length in seconds` | |
| | Specifies the lifetime of the authentication token in seconds. The token is considered invalid once it has expired and is no longer used. This prevents replay attacks by setting a value short enough to prevent the token from being stolen and replayed within its lifetime.<br><br>If specifying this value on a pre-virtual host basis, you need to take into consideration any clock skew between participating domains.<br><br>Default value: `180`<br><br>Example: `authtoken-lifetime = 120` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables the encoding of strings in UTF8 format within the CDSSO token. This option only affects CDSSO tokens created and consumed by the default SSO create and consume libraries.<br><br>Default value: `true`<br><br>Example: `use-utf8 = true` |

## [cdsso-token-attributes]

This stanza is used when you require credential attributes to be included in CDSSO authentication tokens. It defines attributes, specified for a specific peer or domain, that are to be included in CDSSO authentication tokens. The inclusion of the specified attributes will only take place if the default `sso-create` and `sso-consume` libraries (defined in the **[authentication-mechanisms]** stanza), are in use.

The default name of this stanza is derived from the module name for the **pdwpi-cdsso-module** (that is, cdsso) defined in the **[modules]** stanza. It is of the form [*cdsso-module-name*`-token-attributes`].

Credential attributes matching the patterns specified in this stanza for a target host or domain are included in CDSSO authentication tokens constructed for that target host or domain.

| **[cdsso-token-attributes] stanza** | |
|---|---|
| `domain-name = pattern-1, pattern-2, ..., pattern-n` | |
| | The patterns specified in this entry relate to the credential attributes for a target host or domain that you want included in CDSSO authentication tokens constructed for that target host or domain.<br><br>Only a single value for each attribute is used, and only string values are supported. Other types of credential attribute values are ignored.<br><br>Patterns can be specified using shell-style wildcards.<br><br>A default set of attributes can be configured with a *default* entry in this stanza. This set of attributes is used when there is no other entry matching a particular target host. If the *default* entry is not present, then no attributes is included in tokens by default.<br><br>There is no default for this entry.<br><br>Example:<br>`ibm.com = attrprefix_*, *name*`<br>`tivoli.com = *_attrsuffix, an_exact_attribute` |

## [cdsso-incoming-attributes]

This stanza defines the sets of attributes to be accepted and rejected from incoming CDSSO authentication tokens. Unlike the outgoing attributes configured in **[cdsso-token-attributes]**, incoming attributes cannot be configured for a specific peer or domain. Only one set of attribute patterns can be configured, and these patterns are applied to incoming tokens regardless of source.

This processing only takes place if the default `sso-create` and `sso-consume` libraries (defined in the **[authentication-mechanisms]** stanza), are in use.

The default name of this stanza is derived from the module name for the **pdwpi-cdsso-module** (that is, `cdsso`) defined in the **[modules]** stanza. It is of the form [*cdsso-module-name*`-incoming-attributes`].

| **[cdsso-incoming-attributes] stanza** | |
|---|---|
| *attribute pattern* `= {preserve\|refresh}` | |
| | Attributes in CDSSO authentication tokens that match a **refresh** entry are removed from the token before the CDMF library is called to map the remote user into the local domain. Attributes matching a **preserve** entry, or matching none of the entries, are kept. If no entries are configured, then all attributes are kept.<br><br>There is no default for this entry.<br><br>Example:<br>`*_attrsuffix = preserve`<br>`*name* - refresh` |

## [cdsso-domain-keys]

This stanza defines the keys to use for CDSSO. The name of this stanza is derived from the module name for the **pdwpi-cdsso-module** defined in the **[modules]** stanza. It is of the form [*cdsso-module-name*-domain-keys].

| **[cdsso-domain-keys] stanza** | |
|---|---|
| *domain name = key file* | |
| | Keys for domains that are participating in CDSSO.<br><br>There is no default for this entry.<br><br>Example:<br>`ibm.com= /keys/ibm.com.key`<br>`tivoli.com = /keys/tivoli.com.key` |

## [common-modules]

This stanza is where the authentication methods for incoming requests are configured.

See "Configuring authentication" on page 49 for more information about the [common-modules] stanza

| **[common-modules] stanza** | |
|---|---|
| pre-authzn = {boolean-rules \| switch-user \| dynurl \| acctmgmt \| cred-refresh \| forms \| token \| ecsso \| login-redirect \| ext-auth-int} | |
| | Configured pre-authorization modules perform specific processing on requests prior to access to a requested resource is authorized.<br><br>Pre-authorization modules provide functions that do not require authorization or they support capabilities that require access to the request prior to the authorization decision.<br><br>Example:<br>`pre-authzn = accmgmt`<br>`pre-authzn = switch-user` |
| authentication = {BA \| cdsso \| cert \| ecsso \| failover \| forms \| http-hdr \| ip-addr \| iv-headers \| ltpa \| ltpa2 \| spnego \| token \| ext-auth-int} On Windows, ntlm and web-server-authn are also available. | |
| | This entry specifies the modules used for authenticating clients.<br><br>The **[authentication-levels]** stanza takes precedence over the order of the authentication modules definitions in the **[common-modules]** stanza.<br><br>Example:<br>`authentication = BA`<br>`Authentication = forms` |
| session = {BA \| http-hdr \| ip-addr \| iv-headers \| ltpa \| ltpa2 \| session-cookie \| ssl-id \| dsess} | |

| | |
|---|---|
| | Incoming requests are examined for existing authenticated session information. This entry specifies the information used for identifying sessions.<br><br>Example:<br>`session = session-cookie`<br>`session = ltpa` |
| `post-authzn = {BA \| boolean-rules \| cdsso \| failover \| forms \| fsso \| iv-headers \| ltpa \| ltpa2 \| tag-value \| ext-auth-int}` | |
| | This entry specifies the modules to be used if further processing of requests is required after the authorization decision has been made.<br><br>Example:<br>`post-authzn = BA`<br>`post-authzn = failover` |
| `response = {fsso \| ext-auth-int}` | |
| | Specifies the module to use if the plug-in is to process responses from a Web server prior to passing an alternative response to the user.<br><br>Example: `response = ext-auth-int` |
| `transaction = {web-log}` | |
| | Specifies a module that is used for all requests.<br><br>Example: `transaction = web-log` |

## [cred-refresh]

The plug-in can be configured to refresh credential information for a user while keeping their session current. This is useful for updating a user's access without having them logout of their current session. To enable credential refresh functionality the **cred-refresh** module needs to be configured as a pre-authorization module.

The entries in the **[cred-refresh]** stanza determine which attributes to preserve from the original credential and those to refresh in the new credential.

This stanza can be specified for particular virtual hosts by creating a stanza of the form, `[cred-refresh:`*`virtual-host`*`]`.

| **[cred-refresh] stanza** |
|---|
| *`attribute pattern`* `= {preserve \| refresh}` |

| | |
|---|---|
| | For an actual attribute or a matching pattern specify whether that attribute or those matching the pattern should be preserved or refreshed.<br><br>Preserved attributes are retained from the old credential into the new. Refreshed attributes are reloaded into the new credential.<br><br>The standard plug-in pattern-matching rules apply except character comparisons aren't case sensitive. Rules that appear earlier in the stanza will have precedence over those that appear later. If an attribute does not match any of the patterns, it is refreshed by default.<br><br>The following attributes are preserved regardless of the configuration in this stanza:<br>`AZN_CRED_AUTHNMECH_INFO`<br>`AZN_CRED_BROWSER_INFO`<br>`AZN_CRED_IP_ADDRESS`<br>`AZN_CRED_PRINCIPAL_NAME`<br>`AZN_CRED_QOP_INFO`<br>`AZN_CRED_IP_FAMILY`<br>`AZN_CRED_NETWORK_ADDRESS_BIN`<br>`AZN_CRED_NETWORK_ADDRESS_STR`<br>`Attributes marked as read-only`<br><br>Default value:<br>`AUTHENTICATION_LEVEL = preserve`<br>`AZN_CUSTOM_ATTRIBUTES = preserve`<br>`tagvalue_* = preserve` |

## [dsess]

The **[dsess]** stanza contains the configuration entries for the distributed session module. The DSESS module maintains session state information using a cookie and utilizes the session management server for distributing session information.

An number of these entries use the term, replica sets. Remember, replica sets are organized groups of replicated Web security servers.

| **[dsess] stanza** | |
|---|---|
| `use-same-cookie = (yes \| no)` | |
| | This entry specifies whether the HTTP and HTTPS protocols should use the same session ID or different session IDs.<br><br>When this entry is set to *yes*, the cookie name specified in `http-cookie-name` is used for sessions created of either HTTP or HTTPS, and the value specified for the `https-cookie-name` configuration parameter is ignored.<br><br>Default value: `no`<br><br>Example: `use-same-cookie = yes` |
| `dsess-replica-name = unique web server name` | |
| | This entry uniquely identifies the current installation within the replica set.<br><br>This entry is disabled by default. The entry defaults to the authorization server name when not otherwise set.<br><br>Example: `dsess-replica-name = web_server_1` |
| `dsess-replica-set = unique replica set name` | |

| | |
|---|---|
| | The name of the replica set to which this installation belongs.<br><br>This entry is disabled by default. If no replica set is specified the name defaults to that of the virtual host.<br><br>Example: `dsess-replica-set = replica_set_1` |
| `dsess-sess-id-pool-size = max number session IDs` | |
| | The maximum number of session IDs that are pre-allocated within the session ID pool.<br><br>Default value: `512`<br><br>Example: `dsess-sess-id-pool-size = 512` |
| `dsess-propagate-unauth = (true \| false)` | |
| | Enables or disables the storage of unauthenticated sessions in the session management server (SMS). Enabling this option means the authentication process can be distributed across multiple servers. Caution should be used, however, because it opens the SMS to a denial of service attack.<br><br>Default value: `false`<br><br>Example: `dsess-propagate-unauth = true` |
| `cookie-for-domain = (true \| false)` | |
| | Set to `true` the session cookie is available across the domain. Set to `false` it is available only within the scope of the current session.<br><br>Default value: `false`<br><br>Example: `cookie-for-domain = true` |
| `dsess-cluster-name = cluster name` | |
| | The name of the SMS cluster to which this SMS server belongs. This field must be defined and reference an existing **[dsess-cluster]** stanza, qualified by the value of this entry.<br><br>There is no default value.<br><br>Example:<br>`dsess-cluster-name = dsess` |
| `dsess-displacement-form = form name` | |
| | The name of the form presented to the user to confirm a session displacement.<br><br>If no form name is configured the session displacement will not take place.<br><br>An entry can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* `/nls/html/`*lang* `/`*charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>Refer to "Macro support" on page 9 for information on plug-in supported macros.<br><br>Default value: `session_displacement.html`<br><br>Example: `dsess-displacement-form =http://www.organization.com/TAM/dsess/displace.html` |
| `dsess-displacement-uri = displacement uri` | |

| | |
|---|---|
| | The URI that accepts the response from the session displacement form configured in the entry, `dsess-displacement-form`.<br><br>Default value: `/dsess-displacement.form`<br><br>Example: `dsess-displacement-uri = /dsess-displacement.form` |
| `http-cookie-name = session ID name` | |
| | Specifies the name of session established over HTTP.<br><br>Default value: AM-DSESS-SESSION-ID<br><br>Example: `http-cookie-name = AM-DSESS-SESSION-ID` |
| `https-cookie-name = session ID name` | |
| | Specifies the name of sessions established over HTTPS.<br><br>Default value: AM-DSESS-SECURE-SESSION-ID<br><br>Example: `https-cookie-name = AM-DSESS-SECURE-SESSION-ID` |

## [dsess-cluster]

The **[dsess-cluster]** stanza contains all of the defaults for a definition of a cluster of SMS (distributed session) servers.

| **[dsess-cluster] stanza** | |
|---|---|
| `server = ([0-9],)URL` | |
| | A specification for a single SMS server which is a member of this cluster. Values of this entry are defined as `([0-9],)URL`, where the first digit (if present) represents the priority of the server within the cluster (9 being the highest, 0 the lowest). If the priority is not assigned, 9 is assumed.<br><br>The URL can be any well-formed HTTP or HTTPS URL. There is no default URL value.<br><br>Multiple values can be specified for failover and load balancing purposes. Multiple servers can be set at the same priority value. The complete set of these server entries defines the membership of the cluster.<br><br>Example: `server = 9,http://sms.example.com/Dsess/services/Dsess` |
| `response-by = period in seconds` | |
| | The length of time in seconds to maintain a connection to the Web service while waiting for session events.<br><br>This entry is disabled by default. There is no default value.<br><br>`response-by = 20` |
| `basic-auth-user = user_name` | |
| | The name of the user that is included in the basic authentication header.<br><br>This entry is optional. There is no default value.<br><br>`basic-auth-user = userA` |
| `basic-auth-passwd = password` | |

| | |
|---|---|
| | The password that is included in the basic authentication header.<br><br>This entry is optional. There is no default value.<br><br>`basic-auth-passwd = myPassword` |
| `handle-pool-size = maximum number of cached handles` | |
| | The maximum number of cached handles used during communications with the session management server (SMS).<br><br>Default value: `10`<br><br>Example: `handle-pool-size = 10` |
| `handle-idle-timeout = period in seconds` | |
| | The period of time, in seconds, the client waits before am idle handle will be removed from the handle pool cache.<br><br>Example: `handle-idle-timeout = 240` |
| `timeout = period in seconds` | |
| | The period of time, in seconds, the client waits for a response from the session management server (SMS).<br><br>Default value: 30<br><br>Example: `timeout = 240` |
| *The following SSL entries are only required if:*<br>1. *At least on server entry indicates that SSL is to be used; that is, starts with https:.*<br>2. *A certificate is required other than the one used for policy server communication. Details of the default certificate can be found in the `[ssl]` stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.* | |
| `ssl-keyfile = fully qualified path to keyfile` | |
| | The name of the key database file which houses the client certificate to be used.<br><br>If no value is specified the value for the entry, `ssl-keyfile`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.<br><br>The default relates to the certificate.<br><br>Example for UNIX: `ssl-keyfile = var/pdwebpi/keytab/dsess_key.kdb` |
| `ssl-keyfile-stash = fully qualified path to password stash file` | |
| | The name of the password stash file for the key database file.<br><br>If no value is specified the value for the entry, `ssl-keyfile-stash`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.<br><br>The default relates to the certificate.<br><br>Example for UNIX: `ssl-keyfile-stash = var/pdwebpi/keytab/dsess_key.sth` |
| `ssl-keyfile-label = client certificate label` | |
| | The label of the client certificate within the key database.<br><br>If no value is specified the value for the entry, `ssl-keyfile-label`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.<br><br>The default relates to the certificate.<br><br>Example: `ssl-keyfile-label = Distributed Session` |

| ssl-valid-server-dn= *domain name* | |
|---|---|
| | Specifies the DN of the server (obtained from the server SSL certificate) which will be accepted. If no entry is configured all DNs will be considered to be valid. Multiple DNs can be specified by including multiple configuration entries in this name.<br><br>This configuration entry is optional.<br><br>There is no default value.<br><br>Example `ssl-valid-server-dn = cn=was61.ibm.com,0=ibm,c=us` |
| `ssl-fips-enabled = (yes \| no)` | |
| | Determines whether Federal Information Process Standards (FIPS) mode is enabled on the session management server.<br><br>If no configuration entry is present, the setting from the `ssl-fips-enabled` entry in the **[ssl]** stanza of the policy server takes effect. When set to *yes* (or when the setting in the policy server configuration file is set to *yes*), Transport Layer Security (TLS) version 1 (TLSv1) is the secure communication protocol used. When set to *no* (or the setting in the policy server configuration file is set to *no*), SSL version 3 (SSLv3) is the secure communication protocol used.<br><br>This configuration entry is optional.<br><br>There is no default value. If a FIPS level other than that set for the policy server is required, the administrator must manually edit the configuration file to change this value.<br><br>Example `ssl-fips-enabled = no` |

## [dsess-cluster:*cluster_name*]

This stanza defines the cluster of SMS servers associated with the configuration that is defined for *cluster_name* in the default **[dsess]** stanza.

The valid entries and their associated values are the same as for the **[dsess-cluster]** stanza above.

**Note:** If any entries are missing for this stanza, they will be retrieved from the unqualified **[dsess-cluster]** stanza.

| **[dsess-cluster:***cluster_name***] stanza** | |
|---|---|
| `server = ([0-9],)`*URL* | |
| | A specification for a single SMS server which is a member of this cluster. Values of this entry are defined as `([0-9],)`*URL*, where the first digit (if present) represents the priority of the server within the cluster (9 being the highest, 0 the lowest). If the priority is not assigned, 9 is assumed.<br><br>The URL can be any well-formed HTTP or HTTPS URL. There is no default URL value.<br><br>Multiple values can be specified for failover and load balancing purposes. Multiple servers can be set at the same priority value. The complete set of these server entries defines the membership of the cluster.<br><br>Example: `server = 9,http://sms.example.com/Dsess/services/Dsess` |
| `response-by = period in seconds` | |

| | |
|---|---|
| | The length of time in seconds to maintain a connection to the Web service while waiting for session events.<br><br>This entry is disabled by default. There is no default value.<br><br>`response-by = 20` |
| `basic-auth-user = user_name` | |
| | The name of the user that is included in the basic authentication header.<br><br>This entry is optional. There is no default value.<br><br>`basic-auth-user = userA` |
| `basic-auth-passwd = password` | |
| | The password that is included in the basic authentication header.<br><br>This entry is optional. There is no default value.<br><br>`basic-auth-passwd = myPassword` |
| `handle-pool-size = maximum number of cached handles` | |
| | The maximum number of cached handles used during communications with the session management server (SMS).<br><br>Default value: `10`<br><br>Example: `handle-pool-size = 10` |
| `handle-idle-timeout = period in seconds` | |
| | The period of time, in seconds, the client waits before an idle handle will be removed from the handle pool cache.<br><br>Example: `handle-idle-timeout = 240` |
| `timeout = period in seconds` | |
| | The period of time, in seconds, the client waits for a response from the session management server (SMS).<br><br>Default value: 30<br><br>Example: `timeout = 240` |
| *The following SSL entries are only required if:*<br>1. *At least on server entry indicates that SSL is to be used; that is, starts with https:.*<br>2. *A certificate is required other than the one used for policy server communication. Details of the default certificate can be found in the* ***[ssl]*** *stanza of the IBM Security Access Manager for Web configuration file,* `pd.conf.` | |
| `ssl-keyfile = fully qualified path to keyfile` | |
| | The name of the key database file which contains the client certificate to be used.<br><br>If no value is specified the value for the entry, `ssl-keyfile`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf.`<br><br>The default relates to the certificate.<br><br>Example for UNIX: `ssl-keyfile = var/pdwebpi/keytab/dsess_key.kdb` |
| `ssl-keyfile-stash = fully qualified path to password stash file` | |

| | |
|---|---|
| | The name of the password stash file for the key database file.<br><br>If no value is specified the value for the entry, `ssl-keyfile-stash`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.<br><br>The default relates to the certificate.<br><br>Example for UNIX: `ssl-keyfile-stash = var/pdwebpi/keytab/dsess_key.sth` |
| `ssl-keyfile-label = client certificate label` | |
| | The label of the client certificate within the key database.<br><br>If no value is specified the value for the entry, `ssl-keyfile-label`, from the **[ssl]** stanza of the IBM Security Access Manager for Web configuration file, `pd.conf`.<br><br>The default relates to the certificate.<br><br>Example: `ssl-keyfile-label = Distributed Session` |
| `ssl-valid-server-dn= domain name` | |
| | Specifies the DN of the server (obtained from the server SSL certificate) which will be accepted. If no entry is configured all DNs will be considered to be valid. Multiple DNs can be specified by including multiple configuration entries in this name.<br><br>This configuration entry is optional.<br><br>There is no default value.<br><br>Example `ssl-valid-server-dn = cn=was61.ibm.com,0=ibm,c=us` |
| `ssl-fips-enabled = (yes \| no)` | |
| | Determines whether Federal Information Process Standards (FIPS) mode is enabled on the session management server.<br><br>If no configuration entry is present, the setting from the `ssl-fips-enabled` entry in the **[ssl]** stanza of the policy server takes effect. When set to *yes* (or when the setting in the policy server configuration file is set to *yes*), Transport Layer Security (TLS) version 1 (TLSv1) is the secure communication protocol used. When set to *no* (or the setting in the policy server configuration file is set to *no*), SSL version 3 (SSLv3) is the secure communication protocol used.<br><br>This configuration entry is optional.<br><br>There is no default value. If a FIPS level other than that set for the policy server is required, the administrator must manually edit the configuration file to change this value.<br><br>Example `ssl-fips-enabled = no` |

## [dynurl]

This stanza contains the definitions for the dynamic URL pre-authorization module.

Stanzas for specific virtual hosts can be created using the format `[dynurl:virtual-host]`.

| **[dynurl] stanza** |
|---|
| *Security Access Manager object = pattern* |

| | |
|---|---|
| | The patterns defined here are matched against incoming requests so that URLs generated dynamically by Web applications can be protected against unwanted use or access.<br><br>Entries which occur earlier in the stanza have precedence over those that occur later in the stanza. All objects are relative to the virtual host branch, /PDWebPI/*virtual-host*, and all patterns are relative to the virtual host (http://*virtual-host* or https://*virtual-host*)<br><br>There are no default values for the stanza.<br><br>Example:<br>`/servershutdown =/servercontrol.asp \?*action=shutdown*`<br>`/serverreset =/servercontrol.asp \?*action=reset*`<br>`/helppages =*help.html` |

## [ecsso]

e-Community single sign-on (eCSSO) allows users to access resources across multiple servers in multiple domains without requiring re-authentication. This stanza holds the configuration entries for eCSSO.

The stanza name must match the module name for the **pdwpi-ecsso-module** defined in the **[modules]** stanza.

For correct handling of the logout URI (`/pkmslogout` by default) with regard to e-Community Single Sign-On, the **eCSSO** pre-authorization module must be configured before the **acct-mgmt** pre-authorization module.

The name of this stanza is derived from the module name assigned to **pdwpi-ecsso-module** in the **[modules]** stanza. It is of the form [*ecsso-module-name*].

This stanza can be specified for particular virtual hosts by creating a stanza of the form, [*ecsso-module-name*:*virtual-host*]. For example, [ecsso:lotus.com-http].

| **[ecsso] stanza** | |
|---|---|
| `e-community-name = name` | |
| | An e-community as it relates to eCSSO is a group of distinct domains. This group is given a name using this entry. The name is displayed in vouch-for tokens and requests. This name should be consistent across all participants in the e-community.<br><br>Default value: Unless specified, the e-community name defaults to the module name configured in the **[modules]** stanza.<br><br>Example: `e-community-name = ibm_ecommunity` |
| `is-master-authn-server = {yes \| no}` | |

| | |
|---|---|
| | In eCSSO, a master authentication server (MAS) is assigned within the 'home domain' to perform all authentication of users. This should be the only task of the master authentication server.<br><br>By setting this entry to *yes* you are indicating that the current server is the master authentication server. All other servers in the e-community should have this entry set to *no*. When set to *yes*, this server accepts vouch-for requests from other plug-in instances whose domain keys are listed in the **[ecsso-domain-keys]** stanza.<br><br>Default value: `no`<br><br>Example: `is-master-authn-server = yes` |
| `master-authn-server = server name` | |
| | This entry indicates the fully qualified domain name of the master authentication server in the e-community. This entry is mandatory if *is-master-authn-server* is set to *no*.<br><br>There is no default value for this entry.<br><br>Example: `master-authn-server = www.ibm.com` |
| `master-http-port = port number` | |
| | When eCSSO authentication is configured over the HTTP protocol, this entry indicates the port on the master authentication server that listens for the HTTP requests. This entry is ignored if the server is the master authentication server.<br><br>Default value: 80<br><br>Example: `master-http-port = 80` |
| `master-https-port = port number` | |
| | When eCSSO authentication is configured over the HTTPS protocol, this entry indicates the port on the master authentication server that listens for HTTPS requests. This parameter is ignored if the server is the master authentication server.<br><br>Default value: `443`<br><br>Example: `master-https-port = 443` |
| `vf-token-lifetime = time period in seconds` | |
| | The vouch-for token lifetime in seconds. Clock skew between participating servers needs to be taken into consideration when setting this entry as the token lifetime is calculated using the creation time stamped on the cookie.<br><br>Default value: 180<br><br>Example: `vf-token-lifetime = 120` |
| `vf-url = vouch-for URL` | |
| | The vouch-for URL that is contained in a vouch-for request to identify it as such. For example, using the default entry a vouch-for request would have the format: `https://vouch_for_server /pkmsvouchfor?ecommunity_name&target_url`<br><br>Default value: `/pkmsvouchfor`<br><br>An extended URL can be configured. For example: `vf-url = /ecommA/pkmsvouchfor` |
| `vf-argument = argument name` | |

| | |
|---|---|
| | The argument name of the vouch-for token as it appears in the vouch-for reply. The default value of `PD-VF` should only be changed if custom create and consume modules are in use and a different argument name is used to represent the vouch-for token.<br><br>Default value: `PD-VF`<br><br>Example: `vf-argument = PD-VF` |
| `allow-login-retry = {true \| false}` | |
| | Enables or disables the retry of a user login when an unauthenticated user is redirected to the master server for authentication.<br><br>Set to *true*, the master server allows users to re-enter their username/password after an initial failed attempt. Set to *false*, the user is redirected back to the slave server without vouching for the user.<br><br>Default value: `true`<br><br>Example: `allow-login-retry = true` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables UTF8 string encoding within the ECSSO vouch-for tokens and e-community cookies. The value of this parameter only affects vouch-for tokens created and consumed by the default SSO create and consume libraries.<br><br>Default value: `true`<br><br>Example: `use-utf8 = true` |
| `store-slave-request = {true \| false}` | |

| | |
|---|---|
| | This parameter is provided for backwards compatibility only.<br><br>In Tivoli Access Manager for e-business versions 5.1 and earlier, **eCSSO** recorded information about a request by a slave for authentication at both the master and the slave.<br><br>This ensured that the master could remember the exact URL that was requested at the slave, so that when authentication was complete the user could be correctly redirected back to the originally requested URL at the slave. This was necessary when the authentication mechanism used was not capable of recording such information.<br><br>Currently there are no authentication mechanisms that require this, although old (pre-5.1) style forms-based logins did require this (before the ability to record this information in hidden fields of the login forms was added).<br><br>At the master, this can cause problems if a stateful authentication scheme is in use and parallel requests for authentication from the same browser are being processed and session cookies are used to maintain session state. In particular this affects NTLM authentication on Windows systems.<br><br>At both the master and the slave this will also require the storage of session data before authentication is complete. This can cause problems when using the Session Management Server, since session data is only replicated for authenticated sessions.<br><br>By default information about slave requests is not stored at either the master or the slave.<br><br>This parameter can be overridden on a per-virtual host basis.<br><br>Default value: `false`<br><br>Example: `store-slave-request = false` |
| `disable-ec-cookie = {yes \| no}` | |
| | When set to *yes*, this option will disable the use of the e-community cookie, and only the MAS will generate vouch-for tokens.<br><br>This will force the single-sign-on process to always use the MAS, allowing the MAS to detect all hosts that sign on across the e-communities. This supports customers who wish to construct their own eCSSO Single Sign Off solution.<br><br>Disabling eCSSO cookies may provide extra security by ensuring that any compromise of a user's session in a slave domain will not result in impersonation of that user in either the master domain or other slave domains.<br><br>Default value: `no`<br><br>Example: `disable-ec-cookie = no` |
| `no-mas-logout-uri = /pkmslogout-nomas` | |

| | |
|---|---|
| | This configuration entry is used to define the URI of a new form of the /pkmslogout page. This page will operate identically to /pkmslogout, except it does not redirect to the MAS's /pkmslogout page after logging out of the current host.<br><br>Instead it simply performs the normal logout success process and returns the page, as defined by the `no-mas-logout-success` configuration entry. This supports customers wanting to use alternate methods of signing out all the hosts at the MAS.<br><br>Instead of using the technique by which each host to be signed out is visited sequentially by the MAS, all hosts could be visited simultaneously using features of HTML like iframes. A single page of iframes, one for each host to sign out, could be generated at the MAS.<br><br>Each form would access /pkmslogout-nomas of each host. If they accessed /pkmslogout of each host, then each iform would be redirected back to the MAS, making it difficult to control the ensuing /pkmslogout recursion.<br><br>This single page signout method would be more robust in the case of a single host failing to respond. If a single host failed using the sequential sign out process then the sign out sequence would halt possibly leaving some hosts signed in.<br><br>The default value is commented out by default: `/pkmslogout-nomas`<br><br>Example: `no-mas-logout-uri = /pkmslogout-nomas` |
| `no-mas-logout-success = logout_success.html` | |
| | This configuration entry is used to define the action which is taken by the server after the client has been successfully logged out via the `no-mas-logout-uri`. The entry should correspond to either a macro HTML file, which is relative to the translated PDWebPI HTML directory (for example, `/opt/pdwebpi/nls/html/C/utf-8`), or a valid redirect URI.<br><br>The redirect URI can be either absolute or server-relative, and can also contain macros. Be aware however that some clients impose restrictions on the maximum length of a URI; care should be taken to include only those URI elements that are required.<br><br>Default value: `logout_success.html`<br><br>Example: `no-mas-logout-success = logout_success.html` |

## [ecsso-domain-keys]

e-Community single sign on uses keys to encrypt and decrypt the data exchanged between participating servers in an e-community. This stanza defines those keys.

The name of this stanza is derived from the module name assigned to **pdwpi-ecsso-module** in the **[modules]** stanza. It is of the form [*ecsso-module-name*-domain-keys].

This stanza can be specified for particular virtual hosts by creating a stanza of the form, [*ecsso-module-name*-domain-keys:*virtual-host*].

| **[ecsso-domain-keys] stanza** |
|---|
| *domain name = key file* |

| | |
|---|---|
| | Defines the keys to use when communicating with participants from the specified domains within an e-community.<br><br>Configuration of the MAS involves defining the keys for each domain for which it is the master. Configuration of e-community members other than the MAS involves defining the key for the domain and for the MAS. You must specify the fully qualified domain names for the servers and the absolute path names for the key file locations.<br><br>There is no default value for this entry.<br><br>Example:<br>`ibm.com =/abc/xyz/ibm-tivoli.key`<br>`lotus.com =/abc/xyz/lotus-tivoli.key`<br>`tivoli =/abc/xyz/tivoli.key` |

## [ecsso-incoming-attributes]

This stanza defines the attributes to accept from incoming eCSSO vouch-for tokens.

The name of this stanza is derived from the module name assigned to **pdwpi-ecsso-module** in the **[modules]** stanza. It is of the form [*ecsso-module-name*-incoming-attributes].

This stanza can be specified for particular virtual hosts by creating a stanza of the form, [*ecsso-module-name*-incoming-attributes:*virtual-host*].

| **[ecsso-incoming-attributes] stanza** | |
|---|---|
| *attribute pattern* = {preserve \| refresh} | |
| | This stanza defines the sets of attributes to be accepted and those to be rejected from incoming eCSSO vouch-for tokens. Unlike the outgoing attributes configuration, incoming attributes cannot be configured for a specific peer or domain.<br><br>Only one set of attribute patterns can be configured, and these patterns are applied to incoming tokens regardless of source.<br><br>This processing only takes place if the default SSO token create and consume libraries (defined in the **[authentication-mechanisms]** stanza) are in use.<br><br>Attributes in eCSSO vouch-for tokens that match a 'refresh' entry are removed from the token before the CDMF library is called to map the remote user into the local domain. Attributes matching a 'preserve' entry, or matching none of the entries, are kept. If no entries are configured, then all attributes are kept.<br><br>There are no defaults for this entry.<br><br>Example:<br>`attrprefix_* = preserve`<br>`*_aatrsuffix = refresh` |

## [ecsso-token-attributes]

This stanza specifies the credential attributes to include in eCSSO vouch-for tokens.

The name of this stanza is derived from the module name assigned to **pdwpi-ecsso-module** in the **[modules]** stanza. It is of the form [*ecsso-module-name*-token-attributes].

This stanza can be specified for particular virtual hosts by creating a stanza of the form, [*ecsso-module-name*-token-attributes:*virtual-host*].

| **[ecsso-token-attributes] stanza** | |
|---|---|
| *domain name = pattern 1, pattern 2, ... , pattern-n* | |
| | Entries in this stanza define those credential attributes to be included in eCSSO vouch-for tokens. The credential attributes to include are specified by peer or domain.<br><br>This processing only takes place if the default SSO token create and consume libraries (defined in the **[authentication-mechanisms]** stanza) are in use.<br><br>There are no default values for this entry.<br><br>Example:<br>`ibm.com= attrprefix_*, *name*`<br>`tivoli.com = *_attrsuffix, some_exact_attribute` |

## [error-pages]

This stanza contains a mapping of plug-in error codes to either a macro-enabled file or URL. The error codes can be obtained from the *IBM Security Access Manager for Web: Error Message Reference*.

The values specified can be either:

- A macro HTML file located on the translated plug-in HTML directory, *install_path* /nls/html/*lang* /*charset*
- A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.

See "Macro support" on page 9 for information on plug-in supported macros.

The pages displayed during account management operations such as changing user passwords, and logging out are configured using the **[acctmgmt]** stanza.

| **[error-pages] stanza** | |
|---|---|
| 0x35F02188 = *error page* | |
| | The page displayed when an account is locked.<br><br>Default value: acct_locked.html<br><br>Example: 0x35F02188 = http://www.organization.com/TAM/errors/acc_lock_error_page.html |
| 0x35F021BE = *error page* | |
| | The page displayed when an account is temporarily disabled due to failed login attempts.<br><br>Default value: retry_limit_reached.html<br><br>Example: 0x35F021BE = http://www.organization.com/TAM/errors/retry_limit_error_page.html |

| 0x35F0205D = *error page* | |
|---|---|
| | The page displayed when the user successfully authenticates using a forms login URI.<br><br>Default value: `login_success.html`<br><br>Example: `0x35F0205D = http://www.organization.com/TAM/errors/login_succes_page.html` |
| 0x35F02421 = *error page* | |
| | The page displayed when the concurrent session limit has been reached for the user.<br><br>Default value: `session_limit_reached.html`<br><br>Example: `0x35F02421 = http://www.organization.com/TAM/errors/sees_lim_error_page.html` |
| default = *error page* | |
| | The page displayed when an error condition has been detected that does not match any other configured error response.<br><br>Default value: `azn_srv_error.html`<br><br>Example: `default = http://www.organization.com/TAM/errors/unspec_error_page.html` |

## [ext-auth-int]

The external authentication interface module allows a credential to be created based on information supplied by a back-end application's authentication information.

| **[ext-auth-int] stanza** | |
|---|---|
| auth-url = *url* | |
| | This entry specifies the URL to which a user is redirected to for authentication. This URL should not be protected by the plug-in and should be allowed to pass through to the external authentication application.<br><br>The values specified can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* /nls/html/*lang* /*charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>See "Macro support" on page 9 for information on plug-in supported macros.<br><br>There is no default value for this entry.<br><br>Example: `auth-url = /eai/app?orig-url=%URL%` |
| trigger-url = *url* | |

| | |
|---|---|
| | Once the client has authenticated, using the URL configured in the `auth-url` entry, the client is redirected to the trigger URL configured in this entry. This trigger URL indicates that the response should be used to generate a credential.<br><br>The configured URL can contain the following special characters which are used when matching the request URL:<br>1. ?: match any single character;<br>2. *: match any number of characters;<br>3. \: turn off the special meaning of the character that follows;<br>4. [ ] : match any one of the enclosed characters.<br><br>There is no default value for this entry.<br><br>Example:<br>`trigger-url = uri = /eai/page.asp*login*`<br>`trigger-url = uri = /cgi-bin/*` |
| `use-redirect-url-first = {true \| false}` | |
| | This Boolean value controls whether the redirect URL, provided in the authentication response from the EAI application, will take precedence over the originally requested URL in the post-authentication redirect.<br><br>Default value: `false`<br><br>Example: `use-redirect-url-first = false` |
| The plug-in examines the corresponding server response for the following headers. These are the headers containing authentication data used to authenticate the user. | |
| | `redirect-url-hdr-name = am-eai-redir-url`<br>`pac-hdr-name = am-eai-pac`<br>`pac-svc-id-hdr-name = am-eai-pac-svc`<br>`user-id-hdr-name = am-eai-user-id`<br>`user-auth-level-hdr-name = am-eai-auth-level`<br>`user-qop-hdr-name = am-eai-qop`<br>`user-ext-attr-list-hdr-name = am-eai-xattrs` |

## [failover]

This stanza contains the configuration details for the failover cookies authentication and post-authorization modules.

| **[failover] stanza** | |
|---|---|
| `failover-cookies-keyfile = path to key file` | |
| | Declares the path to the key file used to encrypt and decrypt credential data in the failover cookie.<br><br>Default value: `failover.key`<br><br>Example: |
| `failover-cookie-lifetime = time period in minutes` | |
| | The lifetime of a failover cookie in minutes.<br><br>Default value: `30`<br><br>Example: `failover-cookie-lifetime = 40` |
| `enable-failover-cookie-for-domain = {true \| false}` | |

| | |
|---|---|
| | Enables the failover cookie for the whole domain by setting a domain HTTP cookie.<br><br>Default value: `false`<br><br>Example: `enable-failover-cookie-for-domain = true` |
| `failover-update-cookie = number` | |
| | The following entry defines how often the failover cookie activity timestamp is updated. Set to 0, the failover cookie is updated every request. If set to a positive integer, the failover cookie is updated after that length of time, in seconds, has elapsed. If set to a negative integer, the failover cookie will only be updated when authentication occurs, or when the credential is refreshed.<br><br>Default value: `-1`<br><br>Example: `failover-update-cookie = 30` |
| `failover-require-lifetime-timestamp-validation = {true \| false}` | |
| | This entry determines whether lifetime timestamp validation is required for failover authentication to succeed.<br><br>Set to *true*, the lifetime timestamp is required. If it is missing or invalid, failover authentication will fail. Set to *false*, the timestamp is not required, but if it exists and is invalid, failover authentication will fail.<br><br>If the plug-in needs to accept failover cookies generated by an earlier version of the plug-in or WebSEAL, this option needs to be set to *true*.<br><br>Default value: `false`<br><br>Example: `failover-require-lifetime-timestamp-validation = true` |
| `failover-require-activity-timestamp-validation = {true \| false}` | |
| | This entry determines whether activity timestamp validation is required for failover authentication to succeed.<br><br>Set to *true*, the activity timestamp is required. If it is missing or invalid, failover authentication will fail. Set to *false*, the timestamp is not required, but if it exists and is invalid, failover authentication will fail.<br><br>If the plug-in needs to accept failover cookies generated by an earlier version of the plug-in or WebSEAL, this option needs to be set to *true*.<br><br>Default value: `false`<br><br>Example: `failover-require-activity-timestamp-validation = true` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables UTF8 string encoding for the failover cookie. This entry needs to be set to *false* when the plug-in needs to accept failover cookies generated by an earlier version of the plug-in or WebSEAL.<br><br>Default value: `true`<br><br>Example: `use-utf8 = false` |
| `use-same-cookie = {true \| false}` | |

| | |
|---|---|
| | This entry specifies whether the HTTP and HTTPS protocols use the same session or not.<br><br>If set to *true*, the cookie to be used for both HTTP and HTTPS is defined in the `http-cookie-name` entry.<br><br>Default value: `false`<br><br>Example: `use-same-cookie = false` |
| `http-cookie-name = PD-ID` | |
| | Name of the cookie to use for cookies established over HTTP or - if `use-same-cookie` is set to false - over both HTTP and HTTPS.<br><br>Default value: `PD-ID`<br><br>Example: `http-cookie-name = PD-ID` |
| `https-cookie-name = PD-ID` | |
| | Name of the cookie to use for cookies established over HTTPS.<br><br>If `use-same-cookie` is enabled then this value is ignored and the cookie name specified by the `http-cookie-name` parameter is used for cookies created for either HTTP or HTTPS.<br><br>Default value: `PD-ID`<br><br>Example: `https-cookie-name = PD-ID` |

## [failover-add-attributes]

This stanza defines the attributes to add from the original credential into the failover cookie.

This stanza can be defined for specific virtual hosts by creating it in the format, `[failover-add-attributes:`*`virtual-host`*`]`.

| **[failover-add-attributes] stanza** | |
|---|---|
| *attribute pattern* = add | |
| | These entries define which attributes the plug-in should transfer from the original credential into the failover cookie.<br><br>There can be multiple attribute entries.<br><br>The standard plug-in pattern matching rules apply except that character comparisons are **not** case sensitive.<br><br>Rules that appear earlier in the stanza take precedence over those that appear later in the stanza. If an attribute does not match any of the entries then it is not included in the failover cookie. If an attribute is not present in the credential then it will not be added to the failover cookie, regardless of the entries in this stanza.<br><br>The **AUTHENTICATION_LEVEL** and **AZN_CRED_AUTH_METHOD** attributes are always added to the failover cookie regardless of the entries in this stanza.<br><br>There are no default values for this stanza.<br><br>Example: `my_value_* = add` |

## [failover-restore-attributes]

This stanza defines the attributes requiring transfer from the failover cookie to the credential during user authentication with a failover cookie.

This stanza can be defined for specific virtual hosts by creating it in the format, `[failover-restore-attributes:virtual-host]`. The values in such a stanza — defined for a specific virtual host — override the values in the standard **[failover-restore-attributes]** stanza.

| **[failover-restore-attributes] stanza** | |
|---|---|
| *attribute pattern* = {preserve \| refresh} | |
| | Specifies those attributes to preserve from the failover cookie to the credential and those to refresh when a user authenticates using a failover cookie.<br><br>There can be multiple attribute entries.<br><br>The standard plug-in pattern matching rules apply except that character comparisons are **not** case sensitive.<br><br>Rules that appear earlier in the stanza take precedence over those that appear later in the stanza. If an attribute does not match any of the entries then it is not included in the failover cookie. If an attribute is not present in the failover cookie then it will not be preserved regardless of the configuration of this stanza.<br><br>There are no defaults for this entry.<br><br>Example:<br>`myvalue_failover_* = refresh`<br>`my_value_* = preserve` |

## [forms]

This stanza contains the configuration details for the forms-based login module.

| **[forms] stanza** | |
|---|---|
| login-form = *form name* | |
| | The form displayed to prompt the user for authentication information.<br><br>An entry can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* /nls/html/*lang* /*charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>See "Macro support" on page 9 for information on plug-in supported macros.<br><br>Default value: `login.html`<br><br>Example: `login-form =http://www.organization.com/TAM/forms/login_form.html` |
| login-uri = *uri* | |

| | |
|---|---|
| | The URI that receives submitted login details.<br><br>The login details must be submitted to this URI with the user's name specified in the POST data attribute, **username**, and the user's password specified in the POST data attribute, **password**.<br><br>Default value: `/pkmslogin.form`<br><br>Example: `login-uri = /pkmslogin.form` |
| `create-ba-hdr = {yes \| no}` | |
| | Set to *yes*, the supplied username and password is provided in a BA header to the destination application. Set to *no*, the supplied username and password is **not** provided in a BA header to the destination application<br><br>Default value: `no`<br><br>Example: `create-ba-hdr = yes` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables UTF8 string encoding for BA headers.<br><br>Default value: `true`<br><br>Example: `use-utf8 = false` |

## [fsso]

Lists the login forms to be intercepted by the Forms Single Sign-On module.

| **[fsso] stanza** |
|---|
| `login-page-stanza = form name` |
| One or more entries can be specified that point to other stanzas containing the details of each login form to be intercepted.<br><br>Default value: `login-form-1`<br><br>Example:<br>`login-page-stanza = login-form-1`<br>`login-page-stanza = login-form-2`<br><br>The example configuration above requires two extra stanzas, `[login-form-1]` present by default and `[login-form-2]` which would need to be created. |

## [http-hdr]

This stanza contains the entries for the `http-hdr` authentication and session modules.

| **[http-hdr] stanza** |
|---|
| `header = header type` |

| | |
|---|---|
| | All supported header types must be configured in this entry. These are the header types passed to the external authentication mechanism for authentication.<br><br>There are no default values for this entry.<br><br>Example: `header = entrust-client` |
| `auth-source = header` | |
| | Controls whether the configured authentication data (header) is retrieved from a HTTP header or from a cookie. The default will be to retrieve the data from a HTTP header.<br><br>Syntax: `auth-source = [header\|cookie]`<br><br>Default: `header`<br><br>Example: `auth-source = header` |

## [http-method-perms]

This stanza is where you define the permissions required to perform a request using a particular HTTP method.

This stanza can be specified for specific virtual hosts by creating a stanza of the form, `[http-method-perms:`*`virtual-host`*`]`.

See "Plug-in ACL permissions" on page 130 for an explanation of plug-in ACL permissions.

| **[http-method-perms] stanza** | |
|---|---|
| *http method = permission* | |
| | The `<default>` entry defines the permissions required for any methods not explicitly specified elsewhere in this stanza.<br><br>Default values:<br>`<default> = [PDWebPI]r`<br>`OPTIONS = [PDWebPI]r`<br>`GET = [PDWebPI]r`<br>`HEAD = [PDWebPI]r`<br>`POST = [PDWebPI]r`<br>`PUT = [PDWebPI]m`<br>`TRACE = [PDWebPI]r`<br>`PROPFIND = [PDWebPI]R`<br>`PROPPATCH = [PDWebPI]M`<br>`MKCOL = [PDWebPI]N`<br>`COPY = [PDWebPI]r`<br>`MOVE = [PDWebPI]rd` |

## [ihs]

This stanza contains the configuration entries specific to Security Access Manager IBM HTTP Server Plug-in.

| **[ihs] stanza** |
|---|
| `query-contents =` *`query contents program`* |

| | |
|---|---|
| | Specifies the query contents program to use for browsing the IBM HTTP Server Web space by the **pdadmin> object list** command. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[ihs:*branch*]**. For example, [ihs:/PDWebPI/lotus.com].<br><br>Default value: /opt/pdwebpi/bin/wpi_ihs_ls<br><br>Example: query-contents = /opt/pdwebpi/bin/wpi_ihs_ls |
| query-log-file = *log file location* | |
| | Location of log file for errors encountered by the query contents program.<br><br>Default value: *install-path*/log/msg__pdwebpi-ls.log<br><br>Example (Windows): query-log-file = C:\PROGRA~1\Tivoli\POLICY~1\log\msg_pdwebpi-ls.log |
| doc-root = *apache-branch-doc-root* | |
| | Specifies the documentation root that provides the Web space browse capability needed for performing **pdadmin> object list** commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[ihs:*branch*]** stanza, for example [ihs:/PDWebPI/lotus.com].<br><br>There is no default value for this entry.<br><br>Example (UNIX): doc-root =/usr/local/ibm.com/doc/root |

## [iis]

This stanza contains the configuration entries for the IIS Web server.

| **[iis] stanza** | |
|---|---|
| query-contents = *query contents program* | |
| | Specifies the query contents program for browsing the IIS Web space by **pdadmin**. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[iis:*branch*]**, for example, [iis:/PDWebPI/lotus.com]<br><br>Default value: *install-path*/bin/pdwpi-iis-ls.exe<br><br>Example: query-contents = /opt/pdwebpi/bin/wpi_iis_ls |
| query-log-file = *path to query log file* | |
| | Location of log file for errors encountered by the query contents program.<br><br>Default value: query-log-file = *install-path*/log/msg_pdwebpi-ls.log<br><br>Example (Windows): query-log-file = C:\PROGRA~1\Tivoli\POLICY~1\log\msg_pdwebpi-ls.log |
| log-file = *log file name* | |

| | |
|---|---|
| | Defines the log file for error and trace messages from the IIS plug-in, that are kept separate from the Authorization Server's log file in order to ensure consistency of the files. If specified as a relative path, the location is relative to the log sub-directory of the installation directory. If specified as an absolute path, the absolute path is used.<br><br>Default value: `msg_pdwebpi-iis.log`<br><br>Example: `log-file = msg_pdwebpi-iis.log` |
| `use-error-pages = {yes \| no}` | |
| | In certain situations the authorization server will need to send an error code to the client; for example, in the case of a request for authorization information - 401.<br><br>The IIS server can be configured to send back a specific body for these error codes.<br><br>This parameter controls whether the IIS configured pages for the error code are sent back to client, or whether some simple constructed pages are sent instead. By default the simple constructed pages are sent back instead of the configured IIS error pages. System performance may be effected if the IIS error pages are chosen.<br><br>Default value: `no`<br><br>Example: `use-error-pages = yes` |
| `iis5-always-in-data-stream = {yes \| no}` | |
| | For IIS 5 (Windows 2000), some applications also implemented as IIS filters generate additional Web server requests while performing their own processing. In order for these requests to be intercepted, the Security Access Manager Plug-in for Microsoft IIS must be configured to always remain in the data stream.<br><br>For performance reasons, the Security Access Manager Plug-in will normally remove itself from the data stream when it has finished processing a request. To force the plug-in to remain in the data stream and ensure that requests generated by other filters during their processing are intercepted, you must enable this configuration parameter.<br><br>This parameter may NOT be overridden on a per-virtual host basis. Any changes to this parameter require IIS to be restarted before they will take affect.<br><br>By default, the Security Access Manager Plug-in removes itself from the data stream when it has finished processing a request.<br><br>This configuration parameter is specific to IIS 5 (Windows 2000) and is ignored for later versions.<br><br>Default value: `no`<br><br>Example: `iis5-always-in-data-stream = no` |
| `authenticate-by-redirect = {yes \| no}` | |

| | |
|---|---|
| | This parameter controls whether or not redirects are used to trigger Web server authentication (**web-server-authn** authentication module) or client certificate authentication (**cert** authentication module).<br><br>By default redirects are used. This ensures that all application types that may be hosted by IIS can be protected by these Security Access Manager authentication modules.<br><br>If redirects are not used, the type of application that may not be protected by Security Access Manager are those that are implemented as IIS filters.<br><br>Since these authentication methods require interaction with the Security Access Manager IIS extension, an IIS filter application that handles the request (typically based on URL pattern matching) may completely handle the request prior to control being passed to the extension. To ensure that the extension gains control, the client is redirected to a URL handled only by the extension.<br><br>If this is not necessary for your application, a performance improvement can be gained by disabling `authenticate-by-redirect`. This applies particularly to applications where the initial (or all) requests are POST requests with a significant amount of data in the request body.<br><br>When redirects are used in this authentication process, the POST body data must be cached, which involves transfer of the data to and from the authorization server. Disabling redirects removes the need to cache this data, since user agents will typically not submit the POST data until any HTTP 401 authentication exchanges have completed.<br><br>Changes to this parameter require restarting IIS before taking effect.<br><br>This parameter may be qualified by the IIS Web site name. For example, the following specifies that for all Web sites except "Web Site 2" redirects will not be used:<br>`[iis]`<br>`authenticate-by-redirect = no`<br>`[iis:Web Site 2]`<br>`authenticate-by-redirect = yes`<br><br>Default value: `yes`<br><br>Example: `authenticate-by-redirect = yes` |
| `fallback-to-server-port = {true\| false}` | |
| | When following redirects to non-standard ports, some browsers do not include the non-standard port in the Host header of the redirected request. This causes any subsequent redirects to target the standard port (80 for HTTP, 443 for HTTPS) rather than the non-standard port. This may cause these requests to fail.<br><br>Normally, the information in the Host header should be used to target redirects back to this server. Setting this configuration parameter to *true* overrides this and, if the Host header does not include port information, will fallback to the server's listening port as the default (rather than standard port for the protocol).<br><br>Changes to this parameter will not take affect until the Web server has been restarted.<br><br>This is a global parameter and may not be set on a per-virtual host basis nor on a per-branch basis.<br><br>Default value: `false`<br><br>Example: `fallback-to-server-port = false` |

## [iv-headers]

This stanza contains configuration entries for the iv-headers authentication and post-authorization module.

| **[iv-headers] stanza** | |
|---|---|
| `accept = {all \| iv-creds \| iv-user \| iv-user-l \| iv-remote-address \| iv-remote-address-ipv6}` | |
| | Specifies the headers to accept as proof of authentication from a proxy.<br><br>Default value: `all`<br><br>Example: `accept = all` |
| `generate = {all \| iv-creds \| iv-user \| iv-user-l \| iv-groups \| iv-remote-address \| iv-remote-address-ipv6 \| server-name}` | |
| | Specifies the headers to generate when forwarding a request from a proxy.<br><br>Default value: `all`<br><br>Example: `generate = iv-creds` |
| `server-name-header = header name` | |
| | The name of the header to use when server-name is present in the list of values to generate.<br><br>Default value: `iv-server-name`<br><br>Example: `server-name-header = iv-server-name` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables UTF8 string encoding for iv-headers.<br><br>Default value: `true`<br><br>Example: `use-utf8 = false` |

## [ldap]

This stanza contains the configuration entries for LDAP.

| **[ldap] stanza** | |
|---|---|
| `ldap-server-config` | |
| | Indicates the location of the `ldap.conf` file.<br><br>This entry is set by the configuration. |
| `bind-dn` | |
| | Indicates the Distinguished Name of the daemon.<br><br>This entry is set by the configuration. |
| `bind-pwd` | |
| | Indicates the password for the daemon.<br><br>This entry is set by the configuration. |
| `ssl-enabled = {yes \| no}` | |

| | |
|---|---|
| | Indicates whether SSL is enabled.<br><br>This entry is set by the configuration. |
| `ssl-keyfile` | |
| | Indicates path/filename of the SSL keyfile.<br><br>This entry is set by the configuration. |
| `ssl-keyfile-dn` | |
| | Indicates the certificate label in the SSL keyfile, if present.<br><br>This entry is set by the configuration. |
| `ssl-keyfile-pwd` | |
| | Indicates the SSL keyfile password.<br><br>This entry is set by the configuration. |
| `cache-enabled = {yes \| no}` | |
| | Enable and disable the local LDAP cache.<br><br>Default value: `yes`<br><br>Example: `cache-enabled = yes` |
| `cache-user-size = number of entries` | |
| | The number of entries in the LDAP user cache.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `256`<br><br>Example: `cache-user-size = 256` |
| `cache-group-size = number of entries` | |
| | The number of entries in the LDAP group.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `64`<br><br>Example: `cache-group-size = 64` |
| `cache-policy-size = number of entries` | |
| | The number of entries in the LDAP policy.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `20`<br><br>Example: `cache-policy-size = 20` |
| `cache-user-expire-time = number of seconds` | |
| | The amount of time (in seconds) until a user entry in the cache is considered stale and is discarded.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `30`<br><br>Example: `cache-user-expire-time = 30` |
| `cache-group-expire-time = number of seconds` | |

| | |
|---|---|
| | The amount of time (in seconds) until a group entry in the cache is considered stale and is discarded.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `300` (5 minutes)<br><br>Example: `cache-group-expire-time = 300` |
| `cache-policy-expire-time = number of seconds` | |
| | The amount of time (in seconds) until a policy entry in the cache is considered stale and is discarded.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `30`<br><br>Example: `cache-policy-expire-time = 30` |
| `cache-group-membership = {yes \| no}` | |
| | Indicates whether group membership information should be cached.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `yes`<br><br>Example: `cache-group-membership = yes` |
| `cache-use-user-cache = {yes \| no}` | |
| | Indicates whether to use the user cache information or not.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `yes`<br><br>Example: `cache-use-user-cache = yes` |
| `cache-return-registry-id = {yes \| no}` | |
| | Indicates whether to cache the user identity as it is stored in the registry, or cache the value as entered during authentication.<br><br>This optional value is ignored if `cache-enabled` is set to *no*.<br><br>Default value: `no`<br><br>Example: `cache-return-registry-id = no` |
| `prefer-readwrite-server = (yes \| no}` | |
| | Indicates whether to select writable LDAP server when available.<br><br>Default value: `no`<br><br>Example: `prefer-readwrite-server = yes` |
| `auth-using-compare = {yes \| no}` | |
| | Indicates whether to perform authentication using LDAP bind or comparing passwords.<br><br>Default value: `yes`<br><br>Example: `auth-using-compare = no` |
| `default-policy-override-support = {yes \| no}` | |

| | |
|---|---|
| | When set to `yes`, no user Policy will be checked, only the default Policy is checked (saves some LDAP searches).<br><br>This option is disabled by default.<br><br>Example: `default-policy-override-support = yes` |
| `user-and-group-in-same-suffix = {yes \| no}` | |
| | Indicates whether the groups are defined in the same LDAP suffix as the user.<br><br>This entry is commented by default.<br><br>Example: `user-and-group-in-same-suffix = yes` |
| `login-failures-persistent = {yes \| no}` | |
| | Indicates whether the tracking of login failures is persistent (maintained in the registry) or done in local process cache.<br><br>If not set, the default is: `no`<br><br>Example: `login-failures-persistent= yes` |

## [login-form-1]

The stanza name comes from that configured in the **[fsso]** stanza.

Login forms are identified by a 2 stage pattern matching process. Forms single sign-on (FSSO) intercepts all pages matching the regular expression configured in the `login-page` entry. Login forms within those pages are located by matching the action attribute of HTML form elements against the regular expression configured in the `login-form-action` entry.

| **[login-form-1] stanza** | |
|---|---|
| `login-page = login page regular expression` | |
| | Specifies a pattern, using a regular expression, that uniquely identifies requests for an application's login page when using the plug-in's forms single sign-on functionality. The configured pattern is compared against the request URI.<br><br>Default value: `*login.html`<br><br>Example: `login-page =/cgi-bin/getloginpage*` |
| `login-form-action = action attribute regular expression` | |
| | Specifies a pattern, using a regular expression, that identifies the action attribute within the login form when using the plug-in's forms single sign-on functionality. If there are multiple matches then the first is used.<br><br>Default value: `/cgi-bin/*login*`<br><br>Example: `llogin-form-action = *` |
| `argument-stanza = stanza name` | |

| | |
|---|---|
| | This entry specifies the stanza that lists the fields and data required for completing the login form.<br><br>Default value: `auth-data`<br><br>The entries for the default are discussed under the heading `[auth-data]` earlier in this chapter. See "[auth-data]" on page 215.<br><br>Example: `argument-stanza = form1-data` |

## [login-redirect]

This stanza contains details for the **login-redirect** pre-authorization module. For this module to work correctly it *must* be configured before the account management pre-authorization module.

| **[login-redirect] stanza** | |
|---|---|
| `redirect-uri = redirect uri` | |
| | Defines the URI to which a user is redirected upon successful authentication. The specified URI can either be a relative URI or an absolute URI.<br><br>The URI can also contain macros. Refer to "Macro support" on page 9 for information on plug-in supported macros.<br><br>There is no default value for this entry. |

## [ltpa]

This stanza contains details for the LTPA cookie based post authorization and authentication modules. This module is designed to allow single sign-on capability with a WebSphere server.

| **[ltpa] stanza** | |
|---|---|
| `ltpa-keyfile = full path of keyfile` | |
| | Full path name of the LTPA key file.<br><br>There is no default value for this entry. |
| `ltpa-stash-file = password stash file location` | |
| | Location of the password stash file. This entry takes priority over the `ltpa-password` entry. If no stash file is present then this entry should commented out.<br><br>There is no default value for this entry. |
| `ltpa-password = password in lieu of stash file` | |
| | The password to use in lieu of stash file.<br><br>There is no default value for this entry. |
| `ltpa-cookie-name = name of the cookie containing the LTPA token` | |
| | The name of the cookie that contains the LTPA token.<br><br>The default value is **LtpaToken**.<br><br>Example: `ltpa-cookie-name = myLTPACookie` |

| ltpa-lifetime = *lifetime of the LTPA cookie in seconds* | |
|---|---|
| | The lifetime in seconds of the LTPA cookie.<br><br>There is no default value for this entry.<br><br>Example: `ltpa-lifetime = 30` |

## [ltpa2]

The `[ltpa2]` module of the configuration file deals with the **LtpaToken2** cookie, which is used by IBM WebSphere Application Server.

| **[ltpa2] stanza** | |
|---|---|
| ltpa-keyfile = *full path of keyfile* | |
| | Full path name of the LTPA key file.<br><br>There is no default value for this entry. |
| ltpa-stash-file = *password stash file location* | |
| | Location of the password stash file. This entry takes priority over the `ltpa-password` entry. If no stash file is present then this entry should commented out.<br><br>There is no default value for this entry. |
| ltpa-password = *password in lieu of stash file* | |
| | The password to use in lieu of stash file.<br><br>There is no default value for this entry. |
| ltpa-cookie-name = *name of the cookie containing the LTPA token* | |
| | The name of the cookie that contains the LTPA token.<br><br>The default value is **LtpaToken2**.<br><br>Example: `ltpa-cookie-name = myLTPA2Cookie` |
| ltpa-lifetime = *lifetime of the LTPA cookie in seconds* | |
| | The lifetime in seconds of the LTPA cookie.<br><br>There is no default value for this entry.<br><br>Example: `ltpa-lifetime = 30` |

For more details, see "Handling LtpaToken2 cookies" on page 100.

## [modules]

This stanza declares all available authentication methods with their associated shared library names. This includes the methods used for session identification, pre-authorization and post-authorization processing.

| **[modules] stanza** |
|---|
| *module name = shared library name* |

| | |
|---|---|
| | The shared libraries declared in this stanza must exist in the `pdwebpi/lib` directory. Shared library names are specified without any operating-system-specific prefix (such as `lib`) and any operating-system-specific suffix (such as `dll`). An alternative to the default searching path for library files can be defined in the **[module-mgr]** stanza.<br>The module entries relate to the following module types:<br>acctmgmt — Account Management<br>BA — Basic Authentication<br>cert — Certificate<br>failover — Failover<br>forms — Forms<br>fsso — Forms Single Sign-On<br>ip-addr — IP Address<br>iv-headers — IV Headers<br>session-cookie — session Cookie<br>ssl-id — SSL ID<br>tag-value — Tag Value<br>http-hdr— HTTP Header<br>token— Token<br>ltpa — LTPA<br>ecsso — e-Community Single Sign-On<br>cdsso — Cross Domain Single Sign-On<br>login-redirect— Login redirect<br>ntlm — NTLM<br>spnego — SPNEGO<br>web-log — Web Log<br>boolean-rules — Boolean Rules<br>switch-user — Switch User<br>dynurl — Dynamic URL<br>cred-refresh— Credential Refresh<br>web-server-authn — Web server authentication<br>ext-auth-int — External Authentication Interface<br>dsess — Distributed Session<br><br>Default values:<br>`acctmgmt = pdwpi-acct-mgmt-module`<br>`BA = pdwpi-ba-module`<br>`cert = pdwpi-certificate-module`<br>`failover = pdwpi-failovercookie-module`<br>`forms = pdwpi-forms-module`<br>`fsso = pdwpi-fsso-module`<br>`ip-addr = pdwpi-ipaddr-module`<br>`iv-headers = pdwpi-iv-headers-module`<br>`session-cookie = pdwpi-sesscookie-module`<br>`ssl-id = pdwpi-sslsessid-module`<br>`tag-value = pdwpi-tag-value-module`<br>`http-hdr = pdwpi-httphdr-module`<br>`token = pdwpi-token-module`<br>`ltpa = pdwpi-ltpa-module`<br>`ltpa2 = pdwpi-ltpa2-module`<br>`ecsso = pdwpi-ecsso-module`<br>`cdsso = pdwpi-cdsso-module`<br>`login-redirect = pdwpi-loginredirect-module`<br>`spnego = pdwpi-spnego-module`<br>`ntlm = pdwpi-ntlm-module`<br>`web-server-authn = pdwpi-websvrauth-module`<br>`web-log = pdwpi-web-log-module`<br>`boolean-rules = pdwpi-boolean-rules-module`<br>`switch-user = pdwpi-su-module`<br>`dynurl = pdwpi-dynurl-module`<br>`cred-refresh = pdwpi-cred-refresh-module`<br>`ext-auth-int = pdwpi-ext-auth-int-module`<br>`dsess = pdwpi-dsess-module` |

## [module-mgr]

This stanza contains details for the proxy module manager.

| **[module-mgr] stanza** | |
|---|---|
| path = *directory location* | |
| | This entry specifies the location for shared libraries. More than one path entry can be specified. The plug-in library directory is always searched last.<br><br>There is no default entry for this value.<br><br>Example for UNIX: `path = /opt/pdwebpi/lib` |
| verify-step-up-user = {true \| false} | |
| | This entry determines whether in the event of a step-up operation the new user ID must match any pre-existing user ID.<br><br>When set to *true* and a different authenticated user ID is used, a generic authorization server error page is returned to the user. This page can be customized if so desired.<br><br>Default value: `false`<br><br>Example: `verify-step-up-user = false` |

## [ntlm]

The [ntlm] stanza holds the configuration entries for the NTLM authentication module available on Windows platforms.

This stanza holds the configuration entries for the NTLM authentication module available on Windows platforms.

| **[ntlm] stanza** | |
|---|---|
| use-pre-windows-2000-logon-name = {true \| false} | |
| | By default, the **ntlm** module uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the username portion of the username@domain.com logon name.<br><br>This entry permits pre-Windows 2000 logon names to represent the authenticated user in Security Access Manager. This is the username portion of the DOMAIN\USERNAME logon name. This parameter is ignored if Security Access Manager uses Active Directory as its registry. When Active Directory is used the user's Security Access Manager user name is always the username portion of the username@domain.com logon name.<br><br>Default value: `false`<br><br>Example: `use-pre-windows-2000-logon-name = true` |
| use-single-authentication-connection = {true \| false} | |

| | |
|---|---|
| | Set to *true* for the parallel NTLM authentications to succeed. Setting this value to *false* is not advisable unless the first and second phases of the NTLM authentication require different connections.<br><br>Default value: `true`<br><br>Example: `use-single-authentication-connection = true` |

## [p3p-header]

This stanza specifies the P3P compact policy that applies to all HTTP cookies set.

This stanza can be specified for specific virtual hosts by creating a stanza of the form, `[p3p-header:`*`virtual-host`*`]`.

| **[p3p-header] stanza** | |
|---|---|
| `p3p-element =` *`element`* | |
| | This entry is used to specify elements to add to the P3P header besides the compact policy configured with the other configuration items in this stanza. A reference to a full XML policy can be supplied.<br><br>This entry is disabled by default.<br><br>Example: `p3p-element = policyref="/w3c/p3p.xml"` |
| `access = {none \| all \| nonident \| contact-and-other \| ident-contact \| other-ident}` | |
| | This entry specifies the access the user has to the information contained within the cookie and the information linked to the cookie.<br><br>Default value: `none`<br><br>Example: `access = all` |
| `disputes = {true \| false}` | |
| | Specifies whether the full P3P policy contains some information regarding disputes over the information contained within the cookie.<br><br>Default value: `false`<br><br>Example: `disputes = true` |
| `remedies = {correct \| money \| law}` | |
| | Specifies the possible remedies for disputes. If not specified, no remedy information is included in the policy.<br><br>This entry is disabled by default.<br><br>Example: `remedies = money` |
| `non-identifiable = {true \| false}` | |
| | When set to *true*, this parameter specifies that no information in the cookie, or information linked to by the cookie, personally identifies the user in any way. Valid values are *true* or *false*.<br><br>This parameter is disabled by default.<br><br>Example: `non-identifiable = false` |

| | |
|---|---|
| purpose = {urrent \| admin \| develop \| tailoring \| pseudo-analysis \| pseudo-decision \| individual-analysis \| individual-decision \| contact \| historical \| telemarketing \| other-purpose} | |
| | This entry specifies the purpose of the information in the cookie and linked to by the cookie. For all values except current, an additional specifier may be configured. The possible values are always, opt-in, and opt-out.<br><br>Default:<br>purpose = current<br>purpose = other-purpose:opt-in<br><br>Example: purpose = current:always |
| recipient = {ours \| delivery \| same \| unrelated \| public \| other-recipient} | |
| | This entry specifies the recipients of the information in the cookie, and linked to by the cookie.<br><br>Default value: ours<br><br>Example: recipient = delivery |
| retention = {no-retention \| stated-purpose \| legal-requirement \| business-practices \| indefinitely} | |
| | This entry specifies how long the information in the cookie or linked to by the cookie is retained.<br><br>Default value: no-retention<br><br>Example: retention = indefinitely |
| categories = {physical \| online \| uniqueid \| purchase \| financial \| computer \| navigation \| interactive \| demographic \| content \| state \| political \| health \| preference \| location \| government \| other-category} | |
| | This entry specifies the type of information stored in the cookie or linked to by the cookie.<br><br>Default value: uniqueid<br><br>Example: categories = physical |

## [pdweb-plugins]

This stanza defines the virtual hosts the plug-in will protect as well as other global configuration parameters.

| **[pdweb-plugins] stanza** | |
|---|---|
| virtual-host = *virtual host name* | |
| | The virtual-host configuration parameter defines a new virtual host. The value of the parameter corresponds to the name of the stanza used to specify the configuration of that virtual host<br><br>There are no default values for this entry.<br><br>Example:<br>virtual-host = foo.com<br>virtual-host = Default Web Server<br>virtual-host = foo.com - HTTP only |
| web-server = {apache \| ihs \| iis} | |

| | |
|---|---|
| | This entry specifies the Web server type in use.<br><br>There is no default value for this entry.<br><br>Example: `web-server = apache` |
| `windows-file-system = {true\|false}` | |
| | This entry indicates to the Authorization Server that precautions should be taken to avoid security issues related to URIs representing Windows file system resources. When enabled, any access to a URI with path elements that look like Windows short path names are forbidden. In particular path elements ending with ~digit are rejected.<br><br>On Windows systems this parameter is set to *true* by default. On UNIX systems it is set to *false*.<br><br>Example: `windows-file-system = true` |
| `case-sensitive = {true \| false}` | |
| | This entry indicates to the Authorization Server that URIs of differing case are to be treated differently. When not set, URIs are converted to lower case when constructing the corresponding Access Manager object name against which an authorization decision is to be made.<br><br>On UNIX systems this parameter is set to *true*. On Windows systems it is set to *false*.<br><br>Example: `case-sensitive = true` |
| `late-lockout-notification = {yes \| no}` | |
| | The plug-in returns an error page (`acct_locked.html`) that notifies the user of the penalty for reaching or exceeding the maximum value set by the `max-login-failures` policy. This stanza entry specifies whether this notification occurs when the user reaches the `max-login-failures` limit, or at the next login attempt after reaching that limit.<br><br>The default setting for new installations is no. The default setting for migrated installations is *yes*.<br><br>Example: `late-lockout-notification = no` |
| `terminate-on-reauth-lockout = {true \| false}` | |
| | The entry controls whether the login session will be terminated in the event that:<br>• The user registry policy setting, **`max-login-failures`**, is set and<br>• The maximum number of authentication login failures is reached.<br><br>Default value: `true`<br><br>Example: `terminate-on-reauth-lockout = true` |
| `log-file = absolute or relative pathname` | |

| | |
|---|---|
| | This entry specifies the name of the log file that captures status and error messages. Plug-in auditing is configured in the `[aznapi-onfiguration]` stanza.<br><br>The value for *log-file* can be specified either as an absolute or relative pathname. When logging to Tivoli Common Directory is enabled, relative pathnames are applied relative to the product's common logging directory, determined by appending **AMZ/logs** to the value of the `tivoli_common_dir` entry in the `pd.conf` file.<br><br>If logging to Tivoli Common Directory is not enabled, relative pathnames are applied relative to `/var/pdwebpi/log` on UNIX systems or the product's installation directory on Windows systems.<br><br>Default value: `msg_pdwebpi.log`<br><br>Example: `log-file = msg_pdwebpi.log` |
| `logs = number` | |
| | Specifies the number of log files to create before re-using the first log file. To enable log file rollover both this entry and `log-entries` must be set to some positive integer.<br><br>When the authorization server is restarted the log file generation number is reset to 1.<br><br>This entry cannot be specified for specific virtual hosts.<br><br>Default value: `0`<br><br>Example: `logs = 5` |
| `log-entries = number` | |
| | This entry controls the number of log entries written to a log file before rolling over to a new log. To enable log file rollover both this entry and `logs` must be set to some positive integer.<br><br>This entry cannot be specified for specific virtual hosts.<br><br>Default value: `0`<br><br>Example: `log-entries = 500` |
| `mpa-enabled = {true \| false}` | |
| | This entry enables or disables MPA plug-in functionality.<br><br>This entry can be specified for specific virtual hosts by adding it to a [*virtual-hosts*] stanza.<br><br>Default value: `false`<br><br>Example: `mpa-enabled = true` |
| `mpa-protected-object = object name` | |
| | This entry defines the object against which the MPA authorization decision is made. If the configured object name is not an absolute object name (that is, it does not commence with a `/`) it is prepended with the root object space configured for the virtual host.<br><br>This entry can be specified for specific virtual hosts by adding it to a [*virtual-hosts*] stanza.<br><br>Default value: `/PDWebPI`<br><br>Example: `mpa-protected-object = /PDWebPI` |

| user = *process name* | |
|---|---|
| | This entry specifies the user name for the manager and proxy processes.<br><br>This entry is not used on Windows systems.<br><br>Default value: pdwebpi<br><br>Example: user = pdwebpi |
| group = *process name* | |
| | This entry specifies the group name(s) for the manager and proxy processes. More than one group may be specified.<br><br>This entry is not used on Windows systems.<br><br>Default value: pdwebpi,ivmgr<br><br>Example: group = pdwebpi,ivmgr |
| use-accept-language-header = {true \| false} | |
| | This entry determines whether the 'accept-language' HTTP header is used when attempting to locate the language for the generated HTML response.<br><br>This entry can be specified for specific virtual hosts by defining it in a [virtual-host] stanza.<br><br>Default value: true<br><br>Example: use-accept-language-header = true |
| use-accept-charset-header = {true \| false} | |
| | This entry determines whether the 'accept-charset' HTTP header is used when attempting to locate the charset in which to decode elements of a HTTP request, or generate a HTML response. The default value (if not found within this configuration file) is true.<br><br>This entry can be specified for specific virtual hosts by defining it in a [virtual-host] stanza.<br><br>Default value: false<br><br>Example: use-accept-charset-header = false |
| max-cached-http-body = *HTTP body data cached* | |
| | This entry specifies the maximum amount of HTTP body data cached for any given request. If the amount of body data exceeds the configured maximum, all of the body data is discarded.<br><br>The worker-size entry within the [proxy-if] stanza controls the amount of memory allocated for any given request. The max-cached-http-body size, at a minimum, should conform to the following algorithm: (max-cached-http-body * 4/3 * 2 + 3000) <= worker-size This algorithm assumes that 3000 bytes is enough memory to hold:<br>• The request less any POST data;<br>• The returned form less the cached POST data.<br><br>If the size of the request plus the size of the returned form is likely to exceed 3000 bytes you must either increase the worker-size entry or decrease the max-cached-http-body entry.<br><br>Default value: 2500<br><br>Example: max-cached-http-body = 2500 |

| | |
|---|---|
| `send-p3p-header = {true \| false}` | |
| | This entry controls the addition of a P3P header containing a compact policy statement to any HTTP responses in which it has set cookies.<br><br>Before enabling this entry, configure the entries within the `[p3p-header]` stanza to match your organization's privacy policy.<br><br>This entry can be specified for specific virtual hosts by defining it in a `[virtual-host]` stanza.<br><br>Default value: `false`<br><br>Example: `send-p3p-header = true` |
| `tag-value-prefix =` | |
| | This entry specifies the optional prefix added to credential attribute names used for tag value HTTP headers. The tag-value module will search for credential attributes using this prefix; the session ID credential attribute will be added with this prefix; and the su module will add this prefix to the credential attribute it uses to store the administrator's username.<br><br>This entry can be specified for specific virtual hosts by defining it in a `[virtual-host]` stanza.<br><br>There is no default value for this entry.<br><br>Example: `tag-value-prefix =tag-value` |
| `use-uri-encoded-session-id = {true \| false}` | |
| | This entry controls whether or not the session ID specified in the *terminate session* administration task should be URI encoded.<br><br>Default value: `true`<br><br>Example: `use-uri-encoded-session-id = true` |
| `remove-headers = {true \| false}` | |
| | This entry specifies whether any headers that the *tag-value* module may set should be removed from the request before tag-value processing. Removing these headers ensures that they cannot be inserted by a malicious user agent to spoof the values derived from the credential.<br><br>WARNING!<br>Disabling the remove header capability may introduce a security vulnerability to your web site. Applications relying on tag-value headers not being removed from the request prior to tag-value processing must be reimplemented to avoid the possibility of tag-value http headers being spoofed by malicious user agents.<br>WARNING!<br><br>This entry is provided for the purposes of backwards compatibility with plug-in version 4.1 and earlier.<br><br>This entry can be specified for specific virtual hosts by defining it in a `[virtual-host]` stanza.<br><br>Default value: `true`<br><br>Example: `remove-headers = true` |
| `pass-thru-authn-error = {true \| false}` | |

| | |
|---|---|
| | This entry controls whether any authentication system errors are passed through to the user.<br><br>This entry can be specified for specific virtual hosts by defining it in a `[virtual-host]` stanza.<br><br>Default value: `false`<br><br>Example: `pass-thru-authn-error = true` |
| `ipv6-support = {true \| false}` | |
| | This entry controls support for IPv6, determining whether or not credential attributes containing IPv6 addresses are added.<br><br>This entry can be over-ridden on a per-virtual host basis by defining it in a `[virtual-host]` stanza.<br><br>Default value: `false`<br><br>Example: `ipv6-support = false` |
| `compat-ipv4-support = {true \| false}` | |
| | This entry controls backwards compatibility support for IPv4, determining whether or not credential attributes containing IPv4 addresses are added.<br><br>Enabling this option implies a slight performance hit due to additional credential attributes. Disabling this option improves performance, but breaks backward compatibility.<br><br>This entry can be over-ridden on a per-virtual host basis by defining it in a `[virtual-host]` stanza.<br><br>Default value: `false`<br><br>Example: `compat-ipv4-support = false` |
| `resend-pdwebpi-cookies = {true \| false}` | |
| | This entry defines whether Web Plug-In cookies should be sent with each request.<br><br>Default value: `false`<br><br>Example: `resend-pdwebpi-cookies = true` |
| `fips-140-2-mode-enabled = {true \| false}` | |
| | Enables or disables the FIPS 140-2 compliant cryptographic operations within the plug-in.<br><br>Default value: `false`<br><br>Example: `fips-140-2-mode-enabled = false` |
| `config-data-log = msg__pdwebpi_cfgdata.log` | |
| | The plug-in can be configured to dump configuration data at startup, highlighting non-default values and any inheritance to virtual-host qualified stanzas.<br><br>The entries in this stanza specify the location of the file to which the configuration data is logged. Dumps in this file are prefixed and suffixed with separators and prefixed with a timestamp. |
| `httponly-pdwebpi-cookies = false` | |

| | |
|---|---|
| | Defines whether Web Plug-In security cookies should be sent with the **HttpOnly** attribute set.<br><br>This parameter may be overridden on a per-virtual host basis by specifying it in the appropriate [*virtual-host*] stanza. It can also be overloaded for any module which creates a security cookie by specifying it in the appropriate [*module*] stanza. This specification will take precedence over a specification in the [*virtual-host*] stanza.<br><br>Default value: false<br><br>Example: `httponly-pdwebpi-cookies = false` |

## [performance]

This stanza contains entries for fine tuning the performance of the plug-in. This stanza can be defined for specific virtual hosts by creating a `[performance:virtual-host]` stanza.

| **[performance] stanza** | |
|---|---|
| `enable-pop = {true \| false}` | |
| | This entry enables or disables the enforcement of Protected Object Policies (POPs). For customers not requiring the authorization functionality provided by POPs a performance improvement can be achieved by them with this entry.<br><br>POPs are required for all of the following features:<br>• Step-up Authentication<br>• Multi-Factor Authentication<br>• Forced Reauthentication<br>• IP Address based authorization<br>• QoP based authorization<br>• Warning mode<br><br>Default value: `true`<br><br>Example: `enable-pop = true` |
| `add-session-id-to-cred = {true \| false}` | |
| | This entry enables or disables the addition of the session ID to the session credential. This feature should be set to *true* if the session ID is to be inserted into a request header using tag-value.<br><br>Default value: `true`<br><br>Example: `add-session-id-to-cred = false` |

## [proxy-if]

This stanza contains general configuration entries that relate to communication between the plug-in and the Authorization Server.

| **[proxy-if] stanza** |
|---|
| `id = ID or shared memory file` |

| | |
|---|---|
| | The following entry defines the ID (or shared memory file name) for the proxy interface. This ID must match that which is used by the plug-ins.<br><br>There is no default value for this entry. |
| `number-of-workers = number of worker threads` | |
| | The number of worker threads that handle plug-in requests.<br><br>Default value: `10`<br><br>Example: `number-of-workers = 10` |
| `worker-size = amount of memory` | |
| | The amount of memory pre-allocated for each worker thread.<br><br>Default value: `10000`<br><br>Example: `worker-size = 10000` |
| `cleanup-interval = period of time in seconds` | |
| | Time in seconds between each clean-up of shared memory.<br><br>Default value: `300`<br><br>Example: `cleanup-interval = 420` |
| `max-session-lifetime = period of time in seconds` | |
| | The time in seconds that the plug-in waits for a response from the Authorization Server before timing out.<br><br>Default value: `300`<br><br>Example: `max-session-lifetime = 300` |

## [sessions]

This stanza contains configuration entries common to all session modules.

| **[sessions] stanza** | |
|---|---|
| `max-entries = number of entries` | |
| | Defines the maximum number of sessions which may be stored within a single instance of a session module.<br><br>Default value: `4096`<br><br>Example: `max-entries = 4096` |
| `timeout = period of time in seconds` | |
| | Defines the maximum lifetime of a session in seconds.<br><br>Default value: `3600`<br><br>Example: `timeout = 3000` |
| `inactive-timeout = period of time in seconds` | |
| | Defines the length of idle time in seconds required for a session before it will time out.<br><br>Default value: `600`<br><br>Example: `inactive-timeout = 720` |

| reauth-lifetime-reset = {yes \| no} | |
|---|---|
| | If set to *yes* then the credential lifetime timer will be reset upon successful reauthentication.<br><br>Default value: no<br><br>Example: reauth-lifetime-reset = yes |
| reauth-grace-period = 0 | |
| | Specifies the amount of time in seconds the client has as a grace period within which to successfully perform reauthentication if the credential would otherwise have expired.<br><br>Default value: 0<br><br>Example: reauth-grace-period = 30 |
| session-id-cred-attribute = tagvalue_session_index | |
| | Controls the name of the credential attribute that records the session ID in the credential.<br><br>For correct integration with the Command Audit and Reporting Service (CARS), this attribute name must be tagvalue_session_index. Upgrading or migrating users should migrate from the previous user_session_id attribute name totagvalue_session_index.<br><br>Default value: tagvalue_session_index<br><br>Example: session-id-cred-attribute = tagvalue_session_index |

## [session-cookie]

This stanza contains the configuration entries for the session cookie based session module. The session cookie module allows session state information to be maintained by a cookie.

| **[session-cookie] stanza** | |
|---|---|
| use-same-cookie = {yes \| no} | |
| | Specifies whether the HTTP and HTTPS protocols should use the same session.<br><br>Default value: no<br><br>Example: use-same-cookie = yes |
| http-cookie-name = *cookie name* | |
| | Specifies the name of the cookie to use for sessions established over HTTP. If use-same-cookie is enabled then this entry is ignored and the cookie name specified by the https-cookie-name configuration entry is used for sessions created as either HTTP or HTTPS.<br><br>Default value: PDWPI-SESSION-COOKIE<br><br>Example: http-cookie-name = PDWEBPI-SESSION-COOKIE |
| https-cookie-name = *cookie name* | |

| | |
|---|---|
| | Specifies the name of the cookie to use for sessions established over HTTPS. If `use-same-cookie` is enabled then this entry is used for sessions created as either HTTP or HTTPS.<br><br>Default value: PDWPI-SSL-SESSION-COOKIE<br><br>Example: `https-cookie-name = PDWEBPI-SSL-SESSION-COOKIE` |

## [spnego]

The `[spnego]` stanza holds the configuration entries for the SPNEGO module.

| **[spnego] stanza** | |
|---|---|
| `spnego-krb-service-name = *service name*` | |
| | This mandatory entry stores the service name to which the plug-in authenticates during initialization of the spnego authentication module. The service name must match the template NAME[@*hostname*] where hostname is the fully qualified DNS domain name of the plug-in host.<br><br>Default value: `HTTP`<br><br>Example: `spnego-krb-service-name = HTTP@host.tivoli.com` |
| `spnego-krb-keytab-file = *path*` | |
| | The path name of the Kerberos configuration file (`krb5.keytab`). This is used only on UNIX platforms.<br><br>Default value: `/etc/krb5.keytab`<br><br>Example: `spnego-krb-keytab-file = /etc/krb5.keytab` |
| `use-domain-qualified-name = (true \| false)` | |
| | SPNEGO authentication provides a principal name of the form *shortname@domain.com*. By default, Security Access Manager uses only the *shortname* as the user ID. If this parameter is set to *true*, then Security Access Manager will include the domain as part of the Security Access Manager user ID.<br><br>This configuration option has no effect if Active Directory Multi Domain is being used as the Security Access Manager user registry.<br><br>Default value: `false`<br><br>Example: `use-domain-qualified-name = false` |
| `use-single-authentication-connection = (true \| false)` | |
| | When running on a Windows platform, one of the protocols that may be negotiated when using SPNEGO is NTLM. Set to *true* for the parallel NTLM authentications to succeed. Setting this value to *false* is not advisable unless the first and second phases of the NTLM authentication require different connections.<br><br>Default value: `true`<br><br>Example: `use-single-authentication-connection = true` |

## [switch-user]

This stanza contains the definitions for the switch user pre-authorization module.

| **[switch-user] stanza** | |
|---|---|
| switch-user-form = *file* | |
| | Specifies the name of the HTML file which is returned to the client upon the request to 'su'.<br><br>An entry can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* /nls/html/*lang* /*charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>Refer to "Macro support" on page 9 for information on plug-in supported macros.<br><br>Default value: switchuser.html<br><br>Example: switch-user-form = http://www.organization.com/TAM/su/su_form.html |
| switch-user-uri = *URI* | |
| | This entry holds the URI used to invoke the switch user function. Note that the standard authorization policy is not applied to this URI (that is, there is no ACL checking). Instead, group based authorization checking is conducted.<br><br>Default value: /switchuser.html<br><br>Example: switch-user-uri = /switchuser.html |
| switch-user-post-uri = *uri* | |
| | The following entry contains the name of the URI which the 'su' form is submitted to.<br><br>Default value: /pkmssu.form<br><br>Example: switch-user-post-uri = /pkmssu.form |

## [tag-value]

This stanza contains all of the details for the tag/value based post-authorization module.

| **[tag-value] stanza** | |
|---|---|
| cache-definitions = {yes \| no} | |
| | This entry indicates whether to cache the tag-value definitions which are attached to the object space. When set to *yes*, the proxy will need to be restarted to pick up any changes to the tag-value definitions.<br><br>Default value: yes<br><br>Example: cache-definitions = yes |
| cache-refresh-interval = 60 | |

| | |
|---|---|
| | Defines the refresh interval in seconds for the cache of definitions.<br><br>Default value: `60`<br><br>Example: `cache-refresh-interval = 120` |
| `use-utf8 = {true \| false}` | |
| | Enables or disables UTF8 string encoding for tag-value data.<br><br>Default value: `true`<br><br>Example: `use-utf8 = false` |
| `use-uri-encoding = {true \| false}` | |
| | The following entry defines whether to perform URI encoding of the tag-value data.<br><br>Default value: `true`<br><br>Example: `use-uri-encoding = false` |
| `default-header-value = NOT_FOUND` | |
| | The following entry defines a default value to use for headers when no corresponding attribute is found in the user's credential.<br><br>If no default header value is specified, no header will be added. If a default header value is specified any headers without matching credential attributes will be set to the default value.<br><br>Default value: `NOT_FOUND`<br><br>Example: `default-header-value = NOT_FOUND` |

## [token]

This stanza contains entries for the token-card module.

| **[token] stanza** | |
|---|---|
| `token-login-form = login form` | |
| | Specifies the name of the token-card login page.<br><br>An entry can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* `/nls/html/`*lang* */charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>Refer to "Macro support" on page 9 for information on plug-in supported macros.<br><br>Default value: `tokenlogin.html`<br><br>Example: `token-login-form = http://www.organization.com/TAM/token-card/login.html` |
| `next-token-form = next token form` | |

| | |
|---|---|
| | This entry specifies the token-card form to request the next token.<br><br>An entry can be either:<br>• A macro HTML file located on the translated plug-in HTML directory, *install_path* /nls/html/*lang* /*charset*<br>• A valid redirect URI. The redirect URI can be either absolute or server relative and can also contain macros.<br><br>Refer to "Macro support" on page 9 for information on plug-in supported macros.<br><br>Default value: `nexttoken.html`<br><br>Example: `next-token-form = http://www.organization.com/TAM/token-card/next-token.html` |

## [unprotected-resource-cache]

This stanza contains configuration entries related to the unprotected resource cache. Unprotected resource cache configuration is global and cannot be specified for a particular branch.

| **[unprotected-resource-cache] stanza** | |
|---|---|
| `enabled = {yes \| no}` | |
| | Enables or disables the unprotected resource cache.<br><br>Default value: `no`<br><br>Example: `enabled = yes` |
| `max-poll-interval = time period in seconds` | |
| | The maximum time period (in seconds) that will elapse before unprotected resource cache policy changes are propagated from the authorization server to the plug-ins. Valid values are cardinal numbers >= 5.<br><br>Default value: `30`<br><br>Example: `max-poll-interval = 60` |

## [user-agent]

This stanza creates mappings between user agents, as defined in the user-agent HTTP header, and a specific locale.

| **[user-agent] stanza** |
|---|
| *user-agent-pattern = locale-string* |

| | |
|---|---|
| | These entries define the mapping from user-agent to language and character set. The user-agent header is only used when either or both accept-language and accept-charset headers are not found, or if the use of those headers is disabled.<br><br>The stanza includes a list of patterns that are matched, in the order specified, to the contents of the user-agent header. For a list of the available wildcard characters refer to Appendix F, "Special characters allowed in regular expressions," on page 295.<br><br>If a match is found then the directory for the corresponding language and charset is used. In addition to specifying a language and character set for a given user-agent pattern, it is also possible to specify a directory. In this case the directory name specified is used rather than the one for the charset when sending Security Access Manager pages. This directory must be located under the specified language directory.<br><br>There are no default values for this entry.<br><br>Example: `KDDIOTS22* = ja,sjis` |

## [web-log]

This stanza contains the configuration entries for the web-log transaction module. This module specifies the information to be included in the Web server access log file, and for the Sun Java System, IHS and Apache Web servers, the REMOTE_USER CGI variable.

| **[web-log] stanza** | |
|---|---|
| `format-string = string value` | |
| | Format string which is used to construct the Web log user name, and for the Sun Java System, IHS and Apache web servers, the REMOTE_USER CGI variable.<br><br>This string can also contain control sequences; that is:<br>• %u- access manager user name<br>• %d- access manager user DN<br>• %w- Web server user name<br><br>Default value: `%u`<br><br>Example: `format-string = AM User: %u(%d)` |
| `unauth-user-string = string value` | |
| | The string which is used to denote an unauthenticated Security Access Manager user (%u) within the Web server access log file.<br><br>Default value: `-`<br><br>Example: `unauth-user-string = Unauth user` |
| `unauth-server-user-string = string value` | |
| | The string which is used to denote an unauthenticated Web server user (%w) within the Web server access log file.<br><br>If unset, this entry defaults to the value set for `unauth-user-string`.<br><br>Example: `unauth-server-user-string = Unauth Web server user` |

## [web-server-authn]

This stanza contains configuration entries for the **web-server-authn** authentication module which is available on Windows only.

| **[web-server-authn] stanza** | |
|---|---|
| use-pre-windows-2000-logon-name = {true \| false} | |
| | By default, the web-server-authn module uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. This is the *username* portion of the *username*@domain.com logon name.<br><br>This entry allows the pre-Windows 2000 logon name to represent the authenticated user in Security Access Manager.<br><br>This entry is ignored when the Security Access Manager registry is configured to be Active Directory. With Active Directory the user's Security Access Manager user name is always the 'username' portion of the 'username@domain.com' logon name.<br><br>Default value: false<br><br>Example: use-pre-windows-2000-logon-name = true |

## [wpiconfig]

This stanza contains information set by the configuration program to aid in the unconfiguration. There is no reason to modify the entries in this stanza.

# Appendix D. Module quick reference

Authentication is the method of identifying an individual process or entity that is attempting to log on to a secure domain. The authentication method by which an individual or process uses to access a plug-in protected domain can take one of many forms. IBM IBM Security Access Manager Plug-in for Web Servers supports a number of authentication methods. These authentication methods are listed with an appropriate description in the following tables.

*Table 31. Plug-in authentication method/module reference*

| Authentication method/module | Description |
|---|---|
| BA<br>`pdwpi-ba-module` | Basic Authentication authentication module.<br><br>May also be configured as a session and post-authorization module. |
| forms<br>`pdwpi-forms-module` | HTML Forms authentication module.<br><br>Authenticates using a username and password submitted through a form.<br><br>When in use, this module must also be configured as a pre-authorization module. |
| ip-addr<br>`pdwpi-ipaddr-module` | Client IP Address authentication module.<br><br>Provides authentication based solely on the client's IP address. An http-request authentication mechanism must be provided by the customer to map the IP address information to a Security Access Manager principal.<br><br>May also be configured as a session module. |
| http-hdr<br>`pdwpi-httphdr-module` | HTTP Header authentication module.<br><br>Provides authentication based solely on the value of a nominated HTTP header in the request. An http-request authentication mechanism must be provided by the customer to map the header information to a Security Access Manager principal.<br><br>May also be configured as a session module. |
| token<br>`pdwpi-token-module` | Token authentication module.<br><br>IBM Security Access Manager Plug-in for Web Servers supports authentication using a token passcode supplied by the client. This authentication uses a two factor logon based on RSA SecureID jobs.<br><br>When in use, must also be configured as a post-authorization module. |

*Table 31. Plug-in authentication method/module reference (continued)*

| Authentication method/module | Description |
|---|---|
| cert<br>`pdwpi-certificate-module` | Client certificate authentication module.<br><br>The subject DN of the client certificate is mapped by the cert-ssl authentication mechanism to a Security Access Manager principal name. The cert-ssl authentication mechanism requires that the subject DN of the client certificate map directly to the DN of a Security Access Manager user in the user registry.<br><br>This module ignores requests to authenticate requests that did not arrive over an SSL session and so can be safely configured for virtual hosts that handle authorization of both HTTP and HTTPS requests. |
| failover<br>`pdwpi-failovercookie-module` | Failover Cookie authentication module.<br><br>This module accepts a failover cookie to authenticate a user.<br><br>When in use, this module must also be configured as a post-authorization module. |
| iv-headers<br>`pdwpi-iv-headers-module` | IV Headers authentication module.<br><br>Provides authentication based on the values of the iv-user, iv-user-l, iv-creds, or iv-remote-address HTTP header in the request. This is useful for using single-signing on to IBM Security Access Manager Plug-in for Web Servers when a user has already authenticated to a front-end proxy server.<br><br>In order to be trusted, the request must have arrived using an authenticated session with a front-end proxy server (for example a WebSEAL junction). The proxy must authenticate as a user with Proxy ([PDWebPI]p) permission on the protected object space branch of the virtual host being accessed.<br><br>For authentication using iv-remote-address header, a http-request authentication mechanism must be provided by the customer to map the IP address information to a Security Access Manager principal.<br><br>This module may also be configured as a post-authorization module and session module. |
| ecsso<br>`pdwpi-ecsso-module` | e-Community Single sign-on authentication module.<br><br>This module must be configured as an authentication module for virtual hosts other than the master authentication server that is participating in the e-community.<br><br>When in use, this module must also be configured as a pre-authorization module. |
| unauth<br>`pdpwi-unauth-module` | Unauthenticated user authentication module.<br><br>This module is listed here for completeness. It is implicitly always configured as the lowest precedence authentication module and is used to generate a credential for unauthenticated users. |

*Table 31. Plug-in authentication method/module reference (continued)*

| Authentication method/module | Description |
|---|---|
| ltpa<br>`pdwpi-ltpa-module` | LTPA authentication module<br><br>Accepts and authenticates users based on an LTPA cookie. The LTPA cookie can either be provided by WebSEAL or by a WebSphere server. |
| spnego<br>`pdwpi-spnego-module` | SPNEGO authentication module<br><br>Utilizes the standard SPNEGO authentication protocol within Windows LAN domains to achieve a Single Sign-on solution for plug-in implementations on IIS. |
| cdsso<br>`pdwpi-cdsso-module` | CDSSO authentication module<br><br>Allows cross domain single sign-on between different domains. |
| ext-auth-int<br>`pdwpi-ext-auth-int-module` | External authentication interface module.<br><br>Allows a credential to be created based on information supplied by a back-end application. |

*Table 32. Windows-specific authentication modules*

| Module | Description |
|---|---|
| ntlm<br>`pdwpi-ntlm-module` | NTLM authentication module.<br><br>NTLM is a Windows-specific authentication module that uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. |
| web-server-authn<br>`pdwpi-websvrauth-module` | Web server authentication module.<br><br>The Web server authentication modules is an authentication module for Windows platforms. The module uses the Windows 2000 logon name to represent the authenticated user in Security Access Manager. |

*Table 33. Plug-in session module reference*

| Module | Description |
|---|---|
| BA<br>`pdwpi-ba-module` | Basic Authentication session module.<br><br>Use the Basic Authentication Authorization header value as a session key.<br><br>When in use, must also be configured as an authentication module.<br><br>May also be configured as a post-authorization module. |
| ip-addr<br>`pdwpi-ipaddr-module` | IP Address session module.<br><br>Uses an authenticated client IP address as the session key.<br><br>When in use, it must also be configured as an authentication module. |

*Table 33. Plug-in session module reference (continued)*

| Module | Description |
|---|---|
| http-hdr<br>`pdwpi-httphdr-module` | HTTP Header session module.<br><br>Uses an authenticated HTTP header as the session key. |
| session-cookie<br>`pdwpi-sesscookie-module` | Session Cookie session module.<br><br>This module generates and accepts cookies for use in identifying sessions. Generally used only as a low-priority session identification mechanism. |
| ssl-id<br>`pdwpi-sslsessid-module` | SSL Session ID session module.<br><br>Uses the SSL Session ID as a session key. Note that although this module is provided in the Windows distribution of IBM Security Access Manager Plug-in for Web Servers, the Microsoft Internet Information Services Web Server does not provide SSL Session ID information to the plug-in so SSL Session IDs cannot be used as session keys for IIS. |
| iv-headers<br>pdwpi-iv-headers-module | IV headers session module<br><br>Uses IV headers to maintain session state. |
| ltpa<br>`pdwpi-ltpa-module` | LTPA session module.<br><br>Uses LTPA cookies to maintain session state. |

*Table 34. Plug-in pre-authorization module reference*

| Module | Description |
|---|---|
| boolean-rules<br>`pdwpi-boolean-rules-module` | Boolean Rules pre-authorization module. |
| switch-user<br>`pdwpi-switch-user-module` | Switch User pre-authorization module. |
| dynurl<br>`pdwpi-dynurl-module` | Dynamic URL pre-authorization module. |
| acctmgt<br>`pdwpi-acct-mgmt-module` | Account Management pre-authorization module.<br><br>This module provides the logout (/pkmslogout), change password (/pkmspasswd), help (/pkmshelp) features. |
| cred-refresh<br>`pdwpi-cred-refresh-module` | Credential Refresh pre-authorization module. |
| forms<br>`pdwpi-forms-module` | Forms pre-authorization module. |
| token<br>`pdwpi-token-module` | Token pre-authorization module.<br><br>IBM Security Access Manager Plug-in for Web Servers supports authentication using a token passcode supplied by the client. This authentication uses a two factor log on based on RSA SecureID fobs.<br><br>When in use, the token module must also be configured as an authentication module. |

*Table 34. Plug-in pre-authorization module reference (continued)*

| Module | Description |
|---|---|
| ext-auth-int<br>`pdwpi-ext-auth-int-module` | External authentication interface pre-authorization module. |
| login-redirect<br>`pdwpi-loginredirect-module` | Login Redirect pre-authorization module.<br><br>When performing a login using any of the plug-in supported methods, the user is redirected to a configured page upon successful authentication. |
| ecsso<br>`pdwpi-ecsso-module` | e-Community single sign-on pre-authorization module.<br><br>All virtual hosts participating in an e-community must have the ecsso module configured as a post-authorization module.<br><br>This module must also be configured as an authentication module for all participants other than the master authentication server. |

*Table 35. Plug-in post-authorization module reference*

| Module | Description |
|---|---|
| forms<br>`pdwpi-forms-module` | HTML Forms post-authorization module.<br><br>This module handles the submission of the form data during an HTML Forms-based logon.<br><br>When in use, it must also be configured as an authentication module.<br><br>This module may also set the BA header from the submitted username and password. |
| BA<br>`pdwpi-ba-module` | Basic Authentication post-authorization module.<br><br>Modifies the BA header seen by the Web server or by creating it from GSO lockbox data. |
| failover<br>`pdwpi-failovercookie-module` | Failover Cookie post-authorization module.<br><br>This module generates a failover cookie for the client.<br><br>When in use, the failover cookie module must also be configured as an authentication module. |
| iv-headers<br>`pdwpi-iv-headers-module` | IV Headers post-authorization module.<br><br>This module inserts user identity information as IV headers in to the request before allowing the request to be handled by the Web server. This is useful for providing single sign on to an application hosted by the Web server. The headers that can be added are iv-user, iv-user-l, iv-groups, iv-creds, and iv-remote-address.<br><br>This module may also be configured as an authentication module and session module. |

*Table 35. Plug-in post-authorization module reference (continued)*

| Module | Description |
|---|---|
| tag-value<br>`pdwpi-tag-value-module` | Tag/Value post-authorization module.<br><br>This module inserts additional extended attributes from the users credential as HTTP headers in the request before allowing the request to be handled by the Web server. These extended attributes typically correspond to user attributes from the user registry. |
| ltpa<br>`pdwpi-ltpa-module` | LTPA Cookie post-authorization module.<br><br>This module inserts a WebSphere Application Server Lightweight Third Party Authentication (LTPA) cookie into the request before allowing the request to be handled by the Web server. This provides single sign on to a WebSphere Application Server being hosted by the Web server. |
| cdsso<br>`pdwpi-cdsso-module` | CDSSO post-authorization module. |
| boolean-rules<br>`pdwpi-boolean-rules-module` | Boolean Rules post-authorization module. |
| fsso<br>`pdwpi-fsso-module` | Forms single sign-on module. |

*Table 36. Response module reference*

| Module | Description |
|---|---|
| fsso<br>`pdwpi-fsso-module` | Forms single sign-on response module. |
| ext-auth-int<br>`pdwpi-ext-auth-int-module` | External authentication interface response module. |

*Table 37. Transaction module reference*

| Module | Description |
|---|---|
| web-log<br>`pdwpi-web-log-module` | Web log post-authorization module. |

# Appendix E. Command quick reference

## pdwebpi_start

Starts, restarts, and stops the Security Access Manager Plug-in for Web Servers process on UNIX installations. Note that the Plug-in for Web Servers is automatically started and stopped when the Security Access Manager base product is started or stopped. Also, displays the status of all Web servers.

**Note:** If needed, the **pdwebpi_start** command can be used to control the Plug-in for Web Servers independently of the Security Access Manager base product.

### Syntax

**pdwebpi_start start**

**pdwebpi_start stop**

**pdwebpi_start restart**

**pdwebpi_start status**

### Parameters

**pdwebpi_start {start|stop|restart|status}** where:

**start**
Starts the Plug-in for Web Servers process on UNIX installations.

**stop**
Stops the Plug-in for Web Servers process on UNIX installations.

**restart**
Stops and then restarts the Plug-in for Web Servers process on UNIX installations.

**status**
Provides status information of the Plug-in for Web Servers on UNIX installations.

### Availability

This command is located in the default installation directory:
`/opt/pdwebpi/sbin/`

When an installation directory other than the default is selected, this utility is located in the `sbin` directory under the installation directory (for example, *install_dir*`/sbin/`).

### Return Codes

The following exit status codes can be returned:

**0** The command completed successfully.

**1** An error occurred.

# pdwebpi

Returns the current version of Security Access Manager Plug-in for Web Servers. Also, specifies whether to run Plug-in for Web Servers as a service or run it in the foreground.

## Syntax

**pdwebpi** [**–foreground**] [**–version**]

## Purpose

The **pdwebpi** utility returns the current version of Security Access Manager Plug-in for Web Servers. Also, specifies whether to run Plug-in for Web Servers as a daemon or run it in the foreground.

**Note:** When using Windows Remote Desktop Connection, you must run the plug-in as a service.

## Parameters

**–foreground**
Runs the Plug-in for Web Servers binary in the foreground as opposed to running as a daemon.

**–version**
Specifies the version information for the Plug-in for Web Servers installation.

## Availability

This utility is located in the following default installation directories:

- On UNIX:
  `/opt/pdwebpi/bin`
- On Windows:
  `C:\Program Files\Tivoli\pdwebpi\bin`

When an installation directory other than the default is selected, this utility is located in the `/bin` directory under the installation directory (for example, *`installation_directory`*`/bin`).

## Return Codes

**0** The utility completed successfully.

**1** The utility failed. When a utility fails, a description of the error and an error status code in hexadecimal format is provided (for example, `0x14c012f2`). Refer to the *IBM Security Access Manager for Web: Error Message Reference*. This reference provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

## pdwpi-version

Lists the version and copyright information for the Security Access Manager Plug-in for Web Servers installation.

### Syntax

**pdwpi-version [–h] [–V] [–l | binary [binary ... ]]**

### Parameters

**–h** Displays a help or usage message.

**–l** Specifies long list, which lists the versions of all binaries, not just the package version.

**–V** Displays the version information for the **pdwpi-version** binary.

**binary [***binary***]**
Displays version information for specified binaries, or for all files if no binary files are specified.

### Availability

This command is located in the following default installation directories:

- UNIX systems:
  `/opt/pdwebpi/bin/`
- On Windows systems:
  `C:\Program Files\Tivoli\pdwebpi\bin\`

When an installation directory other than the default is selected, this utility is located in the `bin` directory under the installation directory (for example, *install_dir*`\bin\`).

### Return Codes

The following exit status codes can be returned:

**0** The command completed successfully.

**1** An error occurred.

## pdwpicfg –action config

Configures the Security Access Manager Plug-in for Web Servers.

### Syntax

**pdwpicfg –action config –admin_id** *admin_id* **–admin_pwd** *admin_pwd* **–host –auth_port** *authorization_port_number* **–web_server {iis|ihs|apache} –version –iis_filter {yes|no} –web_directory** *server_install_directory* **–vhosts** *virtual_host_id* **–ssl_enable {yes|no} –keyfile** *keyfile* **–key_pwd** *key_password* **–key_label***key_label* **–ssl_port** *ssl_port_number*

**pdwpicfg –action config –interactive {yes|no}**

**pdwpicfg –action config –rspfile** *response_file*

**pdwpicfg –operations**

**pdwpicfg –help [** *options*]

**pdwpicfg –usage**

**pdwpicfg –?**

## Parameters

**–admin_id** *admin_id*
Specifies the administration user identifier (normally, sec_master).

**–admin_pwd** *admin_pwd*
Specifies the password for the administrative user *admin_id*.

**–auth_port** *authorization_port_number*
Specifies the port number of the authorization server. The default port number value is **7237**.

**–help [***options***]**
Lists the option name and a short description. If one or more options are specified, it lists each option and a short description.

**–host**
Specifies the host name of the server. The default value is the configured host name.

**–interactive {yes|no}**
Enables interactive mode for the command if **yes**; otherwise, disables interactive mode for the command. The default value is **yes**.

**–iis_filter {yes|no}**
Enables the Internet Information Server (IIS) filtering if **yes**; otherwise, disables the IIS filtering. The default value is **yes**.

**–keyfile** *keyfile*
Specifies the LDAP SSL key file. There is no default value. Specify this option when you are not running the command in interactive mode and when you have enabled SSL between the Plug-in for Web Servers and LDAP.

**–key_label** *key_label*
Specifies the LDAP SSL key label. There is no default value. Specify this option when you are not running the command in interactive mode and when you have enabled SSL between the Plug-in for Web Servers and LDAP.

**–key_pwd** *key_password*
Specifies the LDAP SSL key file password.

**–operations**
Lists each of the option names one after another with no description.

**–rspfile** *response_file*
Optionally provides the fully qualified path and file name for the Plug-in for Web Servers response file to use during silent installation. A response file can be used for configuration or unconfiguration. There is no default response file name. The response file contains stanzas and *option=value* pair stanza entries. To use response files, see the procedures in the *IBM Security Access Manager for Web: Installation Guide*.

**–ssl_enable {yes|no}**
Enables SSL communications with LDAP if **yes**; otherwise, disables SSL communications with LDAP. The default value is **no**.

**–ssl_port** *ssl_port_number*
Specifies the LDAP SSL port. The default port number value is **636**.

**–usage**
Displays the usage syntax for this command. Also displays an example.

**–vhosts** *virtual_host_id*
Specifies the virtual hosts that are to be protected. The value should be in the format of a comma separated list of virtual host IDs. There should be no spaces between the virtual host IDs.

**–web_directory** *server_install_directory*
Specifies the Web server installation directory. For the Sun Java System Web Server this is the base installation directory of the Web server. For Apache and IHS Web servers this is the directory containing the Web server's configuration file. This option is used only if the Web server being configured is `apache`, `ihs`. The defaults are:

```
apache:/usr/local/apache/conf
ihs [AIX]:/usr/HTTPServer/conf
ihs [Linux]:/usr/IBMHTTPServer/conf
ihs [Solaris]: /usr/IBMHTTPD/conf
```

**–web_server {iis|ihs|apache}**
Specifies the Web server type on which the Plug-in for Web Servers is to be installed. The choices are: `iis` for Internet Information Server, `ihs` for IBM HTTP Server , or `apache` for the Apache Server. This option defaults to the type and location of the configured Web server.

**–?** Displays the usage syntax for this command. Also displays an example.

## Availability

This command is located in the following default installation directories:

- UNIX systems:
  `/opt/pdwebpi/bin/`
- On Windows systems:
  `C:\Program Files\Tivoli\pdwebpi\bin\`

When an installation directory other than the default is selected, this utility is located in the `bin` directory under the installation directory (for example, *install_dir*`\bin\`).

## Return Codes

The following exit status codes can be returned:

**0** The command completed successfully.

**1** The command failed.

When a command fails, a description of the error and an error status code in hexadecimal format is provided (for example, `0x14c012f2`). Refer to the *IBM Security Access Manager for Web: Error Message Reference*. This reference provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

# pdwpicfg –action unconfig

Unconfigures the Security Access Manager Plug-in for Web Servers.

## Syntax

**pdwpicfg –action unconfig –admin_id** *admin_id* **–admin_pwd** *admin_pwd* **–force {yes|no} –remove {none|acls|objspace|all} –vhosts** *virtual_host_id*

**pdwpicfg –action unconfig –interactive {yes|no}**

**pdwpicfg –action unconfig –rspfile** *response_file*

**pdwpicfg –operations**

**pdwpicfg –help [** *options*]

**pdwpicfg –usage**

**pdwpicfg –?**

## Parameters

**–admin_id** *admin_id*
Specifies the administration user identifier (normally, sec_master).

**–admin_pwd** *admin_pwd*
Specifies the password for the administrative user *admin_id*.

**–force {yes|no}**
Forces the unconfiguration process to proceed even if the policy server cannot be contacted. The default value is **no**.

**–help [***options***]**
Lists the option name and a short description. If one or more options are specified, it lists each option and a short description.

**–interactive {yes|no}**
Enables interactive mode for the command if **yes**; otherwise, disables interactive mode for the command. The default value is **yes**.

**–operations**
Lists each of the option names one after another with no description.

**–remove {none|acls|objspace|all}**
Specifies whether to remove the object space or the ACLs or both as part of the unconfiguration process. The default value is **none**.

**–rspfile** *response_file*
Provides the fully qualified path and file name for the Plug-in for Web Servers response file to use during silent installation. A response file can be used for configuration or unconfiguration. There is no default response file name. The response file contains stanzas and *option=value* pair stanza entries. To use response files, see the procedures in the *IBM Security Access Manager for Web: Installation Guide*.

**–usage**
Displays the usage syntax for this command. Also displays an example.

**–vhosts** *virtual_host_id*
Specifies the identifiers of the virtual hosts that are to be unconfigured. The value can be in the format of a comma separated list of virtual host IDs. There should be no spaces between the virtual host IDs.

**–?** Displays the usage syntax for this command. Also displays an example.

## Availability

This command is located in the following default installation directories:

- UNIX systems:
  `/opt/pdwebpi/bin/`
- On Windows systems:
  `C:\Program Files\Tivoli\pdwebpi\bin\`

When an installation directory other than the default is selected, this utility is located in the `bin` directory under the installation directory (for example, *install_dir*`\bin\`).

## Return Codes

The following exit status codes can be returned:

**0** The command completed successfully.

**1** The command failed.

When a command fails, a description of the error and an error status code in hexadecimal format is provided (for example, `0x14c012f2`). Refer to the *IBM Security Access Manager for Web: Error Message Reference*. This reference provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

# Appendix F. Special characters allowed in regular expressions

The following table lists the special characters allowed in regular expressions used in the `pdwebpi.conf` configuration file.

| | |
|---|---|
| ***** | Matches zero or more characters |
| **?** | Matches any one character |
| \ | Escape character (for example, \? matches ?) |
| **[acd]** | Matches character a, c, or d (case-sensitive) |
| **[^acd]** | Matches any character except a, c, or d (case-sensitive) |
| **[a-z]** | Matches any character between a and z (lower case letter) |
| **[^0-9]** | Matches any character not between 0 and 9 (not a number) |
| **[a-zA-Z]** | Matches any character between a and z (lower case) or A and Z (upper case) |

# 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 might 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 U.S.A.

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.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 illustrate 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.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.


Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

# Index

## Special characters



## A



## B



## C

## D



## E



## F



## G



## H



## I



## K



## L

## M



## N



## O



## P



## Q



## R



## S

# T



# U



# V



# W

Printed in USA

SC23-6507-02