IBM Security Access Manager
Version 9.0.7
June 2019

Web Reverse Proxy Configuration topics

IBM Security Access Manager
Version 9.0.7
June 2019

Web Reverse Proxy Configuration topics

Contents

Chapter 4. Session State 279

Chapter 5. Distributed session cache 327

Chapter 6. Authorization. 355

Figures

Tables

Chapter 1. Administration

With the centralized policy management solution for distributed applications in IBM Security Access Manager, you can build secure and well-managed intranets for Internet-based applications. You must understand how WebSEAL protects your resources so that you can successfully administer the WebSEAL server.

IBM Security Access Manager WebSEAL overview

WebSEAL is a high performance, multi-threaded web server that applies fine-grained security policies to the Security Access Manager protected web object space. Use WebSEAL so that you can manage access to your private and internal resources.

Introduction to IBM Security Access Manager

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

In addition to state-of-the-art security policy management, IBM Security Access Manager provides authentication, authorization, data security, and centralized resource management capabilities.

IBM Security Access Manager offers the following features:

- Authentication

  Provides a wide range of built-in authenticators and supports external authenticators.
- Authorization

  Provides permit and deny decisions for protected resources requests in the secure domain through the authorization API.
- Data security and centralized resource management

  Manages secure access to private internal network-based resources by using the public Internet's broad connectivity and ease of use with a corporate firewall system.

WebSEAL introduction

WebSEAL is a resource manager that protects web-based information and resources. It can provide single sign-on solutions and incorporate back-end web application server resources into its security policy.

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

WebSEAL provides the following features:

- Supports multiple authentication methods.
- Integrates with Security Access Manager authorization service.
- Accepts HTTP and HTTPS requests.

- Integrates and protects back-end server resources through WebSEAL junction technology.

  Provides a unified view of a combined protected object space.
- Manages fine-grained access control for the local and back-end server resources.

  Supported resources include URLs, URL-based regular expressions, CGI programs, HTML files, Java™ servlets, and Java class files.
- Performs as a reverse web proxy.

  WebSEAL appears as a web server to clients and appears as a web browser to the junctioned back-end servers it is protecting.
- Provides single sign-on capabilities.

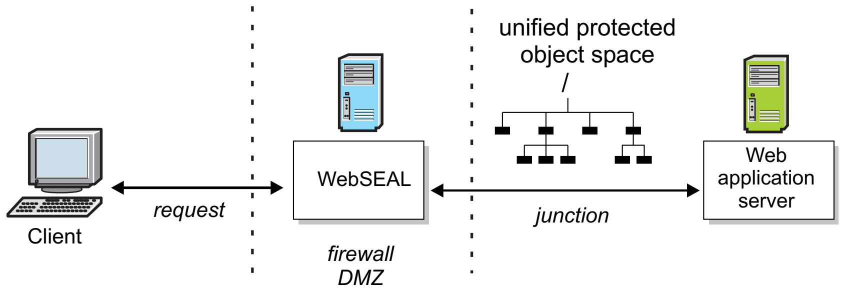


Figure 1. Protecting resources with WebSEAL

IBM Security Access Manager appliance

The IBM Security Access Manager appliance provides access and authentication management for user to web application sessions and helps protect applications from threats.

The appliance uses WebSEAL to enable scalability and protection of the applications from invalid access and threats. WebSEAL is located between the users and the back-end application servers.

There are two appliance offerings:
- Virtual offering
- Hardware offering

The appliance has a local management interface for administrative purposes. You can use the it to manage the WebSEAL instances in your environment. For more information about using the local management interface, see the Administration topics in the IBM Knowledge Center.

WebSEAL functionality on the appliance

The appliance web reverse proxy includes most of the features offered by a standard software installation of WebSEAL. However, there are some differences, as detailed in this section.

Table 1. WebSEAL features that the appliance does not support

Feature	Description
Custom libraries, including CDAS and EAS	The appliance does not support custom CDAS modules. As a result, the appliance does not support the following authentication mechanisms: • IP address • HTTP header • Post password change WebSEAL does not provide CDAS modules for these mechanisms. **Note:** The appliance does support the IBM Security Identity Manager Password Synchronization Plug-in. For more information, see the **[itim]** stanza in the Web Reverse Proxy Stanza Reference..
Local junctions	The following limitations apply to local junction support on the appliance: • The appliance can support a single fixed file system path for the local junction of a WebSEAL instance. • Local junctions on the appliance cannot run any CGI scripts.
Hardware Based Cryptography	The appliance does not support any hardware-based cryptography. However, the hardware appliance does include AES-NI support in the i7-2600 processor, which can handle cryptographic operations.
Application Response Measurement (ARM)	WebSEAL software includes ARM to monitor transactions throughout the request and response processing stream. The appliance does not include ARM.
Tivoli® Common Directory Logging	The Tivoli Common Directory Logging feature stores all log files for IBM® Security software applications in a common file system directory. The appliance does not support this common logging. Logging for the appliance is managed through the LMI.
Auditing to a pipe	The appliance cannot send audit records directly to a pipe. It can however, use an intermediate Security Access Manager authorization server to indirectly send audit records to the destinations. The appliance can also send audit data to remote syslog.
ARS (web service)	The IBM Security Access Manager ARS web service can send request information to an external ARS server for authorization. ARS is not available on the appliance.

Security concepts for a WebSEAL deployment

You must understand the security model concepts so that you can successfully deploy WebSEAL and protect your resources. Basic concepts include protected object space, access control lists, and protected object policies.

Secure domain overview

The computing environment in which Security Access Manager enforces security policies for authentication, authorization, and access control is called a *secure domain*.

The initial secure domain, called the *management domain*, is created when you install and configure the following systems:

Policy server
: Maintains the master authorization database for the management domain. In addition, it updates authorization database replicas and maintains location information about other Security Access Manager servers.

Registry
: Provides a database of the user identities that are known to Security Access Manager. It also provides a representation of groups in Security Access Manager roles that are associated with users.

These core systems must exist for Security Access Manager to complete fundamental operations, such as permitting or denying user access to protected objects (resources). All other Security Access Manager services and components are built on this base.

You can deploy Security Access Manager on multiple systems to configure and use the management domain on one stand-alone system. A single system setup is useful only when prototyping a deployment or developing and testing an application.

After you configure the policy server and registry server, you can set up more systems in the management domain. For example, you could set up an authorization server or application development system. You can also create more secure domains (if you use an LDAP registry) to securely partition data into separate, logical groupings. For information about creating multiple domains, see the Administering topics in the IBM Knowledge Center.

The role of the user registry and master authorization database in security

The user registry and the master authorization database are two key security structures that govern and maintain the security policy of a Security Access Manager secure domain.

A user registry, such as IBM Tivoli Directory Server or Microsoft Active Directory, contains all users and groups who can participate in the Security Access Manager environment. This environment is the secure domain.

The authorization database contains a representation of all resources in the domain. The security administrator can dictate any level of security by applying rules to the resources that require protection. These rules are known as access control list (ACL) policies and protected object policies (POPs).

The process of authentication proves the identity of a user to WebSEAL. A user can participate in the secure domain as authenticated or unauthenticated.

Authenticated users must have an account in the user registry. Using ACLs and POPs, the security administrator can ensure:

- Certain resources are publicly available to unauthenticated users

- Other resources are available only to certain authenticated users

When a user successfully authenticates, WebSEAL creates a set of identification information that is known as a credential. The credential contains the user identity, any group memberships, and any special extended security attributes.

A user requires a credential to fully participate in the secure domain. The Security Access Manager authorization service enforces security policies by comparing a user's authentication credentials with the policy permissions assigned to the requested resource. The authorization service passes the resulting recommendation to the resource manager, for example, WebSEAL, which completes the response to the original request.

The protected object space and system resource

The protected object space is a hierarchical representation of resources that belong to a Security Access Manager secure domain. The system resource is the actual physical file or application.

The authorization service, Web Portal Manager, and other Security Access Manager management utilities use the protected object space.

You can attach policies to objects in the object space so that resources are protected. The authorization service makes authorization decisions that are based on these policies.

The combined installation of Security Access Manager base and WebSEAL provides the following object space categories:

Web objects

Represent any resource that can be addressed by an HTTP URL. These objects can include static web pages and dynamic URLs that are converted to database queries or some other type of application. The WebSEAL server is responsible for protecting web objects.

Management objects

Represent the management activities that administrators can perform through policy administration. The objects represent the tasks that define users and set security policy. Security Access Manager supports delegation of management activities and can restrict an administrator's ability to set security policy to a subset of the object space.

User-defined objects

Represent customer-defined tasks or network resources that are protected by applications that access the authorization service through the Security Access Manager authorization API.

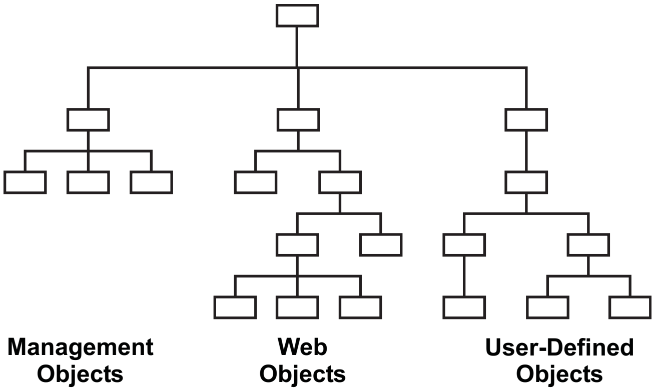


Figure 2. Protected object space

Access control lists and protected object policies

Security administrators define and apply the access control list (ACL) and protected object policy (POP) to protect resources in the Security Access Manager system. These rules are applied to the object representations of the resources in the protected object space.

The Security Access Manager authorization service authorizes decisions that are based on the policies that are applied to these objects. When a requested operation on a protected object is permitted, the application responsible for the resource implements it.

One policy can dictate the protection parameters of many objects. Any change to the rule affects all objects to which the ACL or POP is attached.

Access control list policies

An access control list policy, or ACL policy, controls what operations a user can perform on the resource and who can perform them.

It is a set of rules or permissions that specify the conditions that are necessary to perform certain operations on a resource.

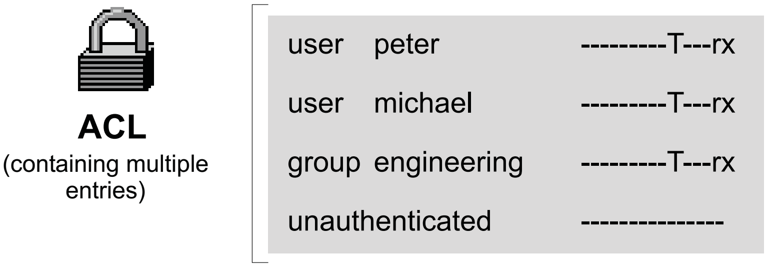
ACL policy definitions are components of the security policy that establish the secure domain. ACL policies define organizational security requirements to the resources in the protected object space.

ACL policies also provide the authorization service with information to make a `yes` or `no` determination for a request to access a protected object.

An ACL policy controls:

- What operations can be performed on the resource
- Who can perform these operations

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



user	peter	---------T---rx
user	michael	---------T---rx
group	engineering	---------T---rx
unauthenticated		---------------

Figure 3. ACL policy

Protected object policies

Protected object policies, or POPs, are policies that contain extra conditions on the requests that are sent to the Security Access Manager and WebSEAL along with the `yes` ACL policy decision from the authorization service.

The Security Access Manager and the resource manager enforce the POP conditions.

The following tables list the available attributes for a POP:

Enforced by Security Access Manager	
POP Attribute	**Description**
Name	Name of the policy. This attribute becomes the *<pop-name>* argument in the **pdadmin pop** commands.
Description	Descriptive text for the policy. This attribute appears in the **pop show** command.
Warning Mode	Provides administrators a means to test ACL and POP policies.
Audit Level	Specifies the type of auditing: all, none, successful access, denied access, errors.
Time-of-Day Access	Day and time restrictions for successful access to the protected object.

Enforced by Resource Manager (WebSEAL)	
POP Attribute	**Description**
Quality of Protection	Specifies the degree of data protection: none, integrity, privacy.
IP Endpoint Authentication Method Policy	Specifies the authentication requirements for access from members of external networks.
Document Cache Control	Specifies the caching instructions for the handling of specific documents.

Explicit and inherited policy

Policies can be explicitly applied or inherited. The protected object space supports inherited and explicit ACL and POP attributes.

Inheritance is an important management feature for the security administrator. The administrator must apply explicit policies only at points in the hierarchy where the rules must change.

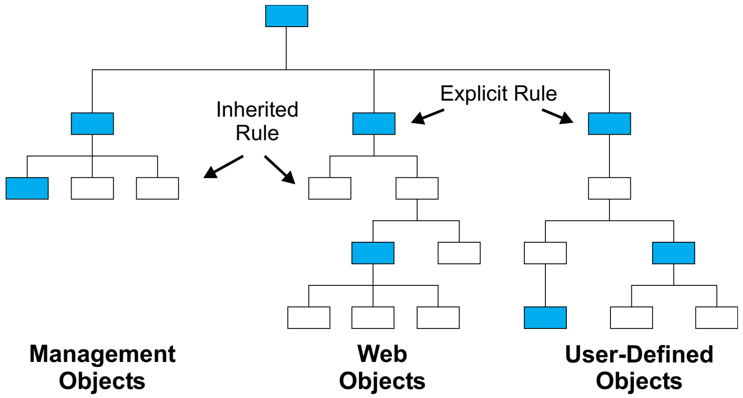


Figure 4. Explicit and inherited policies

Policy administration: The Web Portal Manager

The Web Portal Manager is a web-based graphical application that manages the security policy in a Security Access Manager secure domain. Use the Web Portal Manager to manage the user registry, the master authorization policy database, and the Security Access Manager servers.

You can add and delete users and groups and apply ACLs and POPs to network objects.

The **pdadmin** command-line utility provides the same administration capabilities as the Web Portal Manager, plus some commands that are not supported by the Web Portal Manager.

To access Web Portal Manager from the appliance, go to **Secure Web Settings** > **Manage** > **Policy Administration**.

Authorization process

The authorization process determines whether an authenticated user has the right to perform an operation on a specific resource in a secure domain.

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

In security systems, authorization is distinct from authentication. Authentication can validate the identity of a user, but says nothing about the user's right to perform operations on a protected resource.

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

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

The resources in a secure domain receive a level of protection as dictated by the security policy for the domain. The security policy defines the legitimate participants of the secure domain and the degree of protection that surrounds each resource that is being protected.

The authorization process consists of the following basic components:

- A **resource manager** is responsible for implementing the requested operation when authorization is granted. WebSEAL is a resource manager.

  A component of the resource manager is a **policy enforcer** that directs the request to the authorization service for processing.

  Note: Traditional applications bundle the policy enforcer and resource manager into one process. Examples of this structure include WebSEAL and third-party applications.
- An **authorization service** performs the decision-making action on the request.

The following diagram illustrates the complete authorization process:

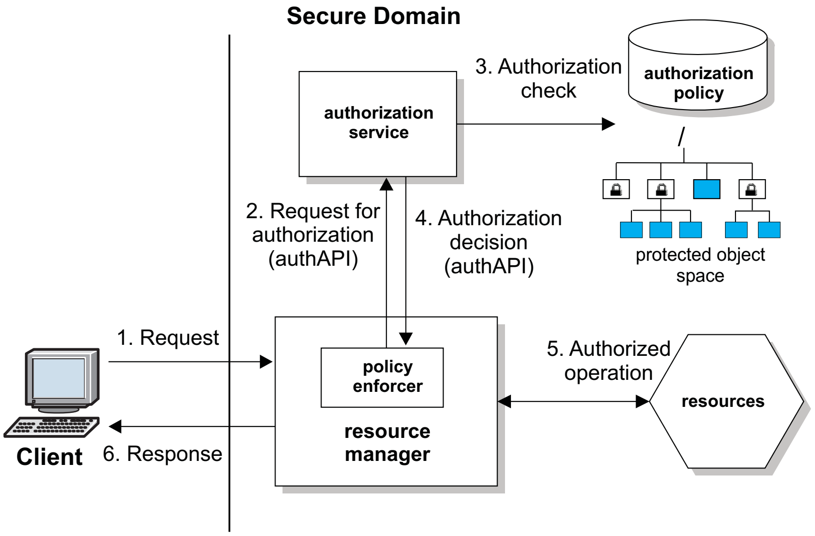


Figure 5. Web space protection

1. A request for a resource from an authenticated user is directed to the resource manager and intercepted by the policy enforcer process.

   The resource manager can be WebSEAL (for HTTP, HTTPS access) or a third-party application.
2. The policy enforcer process uses the Security Access Manager authorization API to call the authorization service for an authorization decision.
3. The authorization service performs an authorization check on the resource, represented as an object in the protected object space.
   a. Security Access Manager POPs are checked first.
   b. Next the ACL policy that is attached to the object is checked against the client's credentials.

   c. Finally, POPs enforced by the resource manager are checked.
4. The decision to accept or deny the request is returned as a recommendation to the resource manager (through the policy enforcer).
5. If the request is finally approved, the resource manager passes the request on to the application responsible for the resource.
6. The user receives the results of the requested operation.

Security policy planning and implementation

A corporate security policy for web resources identifies the web resources that require protection and the level of protection. You can implement the security policy by applying the appropriate security mechanisms to the objects requiring protection.

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.

Security mechanisms include:

- **Access control list (ACL) policies**

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

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

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

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

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

Content types and levels of protection

As the security administrator of your web space, you must correctly identify the types of content available to various user types. Some content must be highly protected and available only to specific users; other content is for general public view.

Each security scenario demands different protection requirements and an associated WebSEAL configuration.

You must take note of the following responsibilities.

- Know your web content
- Identify the types of users that require access to this content

- Understand the strengths and weaknesses of the available WebSEAL configuration options for securing this content

Protection of web content falls into three broad categories:

1. **Public content** – access requires no protection
   - Unauthenticated users can access resources by using HTTP.
   - An unauthenticated credential is used for access control to resources.
   - Basic WebSEAL configuration requirements provide protection.
2. **Public content** – access requires privacy (encryption)
   - Unauthenticated users can access resources by using HTTPS.
   - Encryption, which is required by the application server, is used to protect sensitive data (such as credit card numbers and user account information).
   - An unauthenticated credential is used for access control to resources.
   - WebSEAL configuration needs to stipulate privacy.
3. **Private content** – access requires authentication
   - Authenticated clients can access resources by using HTTP or HTTPS.
   - The administrator determines the need for encryption.
   - An authenticated credential is used for access control to resources; each user must have an account that is defined in the Security Access Manager user registry.
   - WebSEAL configuration is complex and all options must be considered carefully to determine the impact of the security policy.

WebSEAL authentication

Authentication is the method of identifying an individual process or entity that is attempting to log in to a secure domain. WebSEAL can enforce a high degree of security in a secure domain by requiring each user to provide proof of its identity.

The following conditions apply to the WebSEAL authentication process:

- WebSEAL supports several authentication methods by default and can be customized to use other methods.
- When both server and client require authentication, the exchange is known as mutual authentication.
- The WebSEAL server process is independent of the authentication method.
- The result of successful authentication to WebSEAL is a Security Access Manager user identity.
- WebSEAL uses this identity to build a credential for that user.
- The authorization service uses this credential to permit or deny access to protected objects after it evaluates the ACL permissions and POP conditions that govern the policy for each requested resource.

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

For a complete overview of WebSEAL authentication concepts, see “Authentication overview” on page 117.

Standard WebSEAL junctions

A WebSEAL junction is a TCP/IP connection between a front-end WebSEAL server and a back-end server.

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

The connection between a WebSEAL server and a back-end web application server is known as a standard WebSEAL junction.

Note: WebSEAL also supports virtual hosting through another form of junctions called virtual host junctions.

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

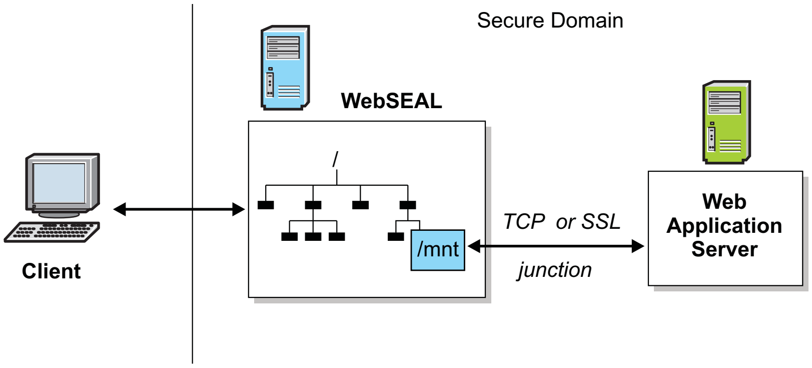


Figure 6. Junctions connect WebSEAL with back-end resources

A junction allows WebSEAL to provide protective services on behalf of the back-end server. WebSEAL authenticates and authorizes all requests before it passes those requests on to the back-end server. If the back-end server requires fine-grained access control on its objects, you must perform additional configuration steps, by using the `query_contents` CGI program, to describe the third-party web space to the Security Access Manager security service.

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

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

The client never needs to know the physical location of a web resource. WebSEAL translates logical URL addresses into the physical addresses that a back-end server expects. Web objects can be moved from server to server without affecting the way that the client can access those objects.

A unified web space simplifies the management of all resources for the system administrator. Additional administrative benefits include scalability, load balancing, and high availability.

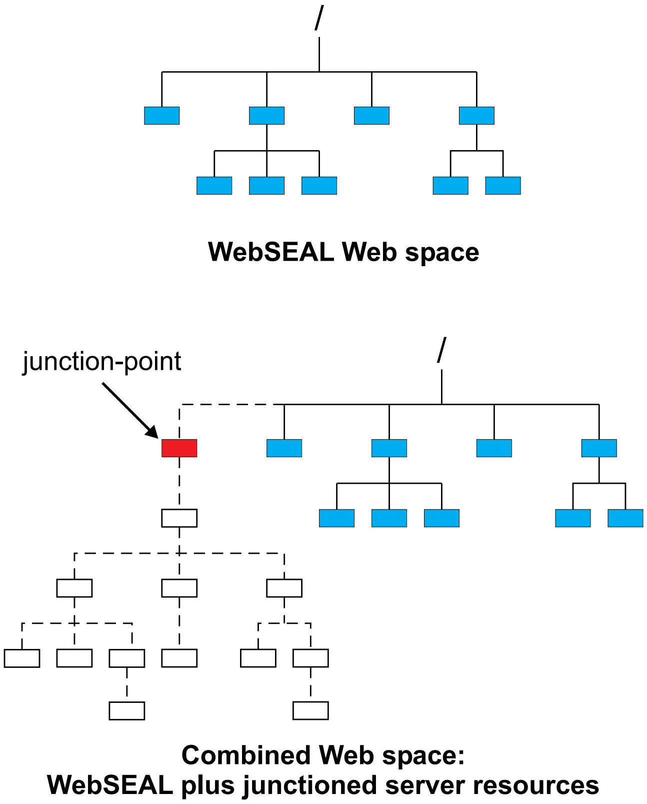


Figure 7. WebSEAL junction results in a unified web space

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

With WebSEAL junctions, you can create single signon solutions. A single signon configuration allows a user to access a resource, regardless of the resource's location, by using only one initial login. Any further login requirements from back-end servers are handled transparently to the user.

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

Web space scalability

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

Additional servers can be added for the following reasons:

- To extend the web space with additional content.
- To duplicate existing content for load balancing, failover capability, and high availability.

Replicated front-end WebSEAL servers

Junction support for back-end servers starts with at least one front-end WebSEAL server. Replicated front-end WebSEAL servers provide the site with load balancing during periods of heavy demand.

The load balancing process is handled by a third-party device such as Cisco Local Director.

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

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

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

Junctioned back-end servers

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

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

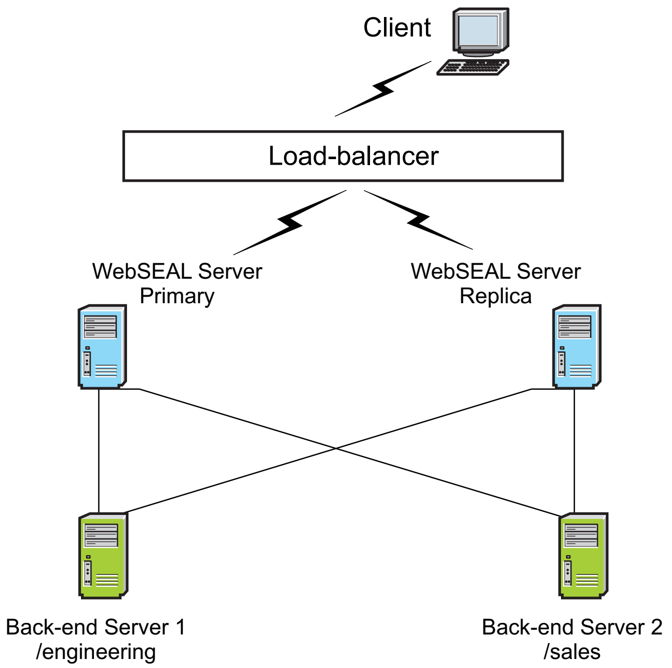


Figure 8. Junctioned back-end servers

The combined web spaces of the two back-end servers are transparent to the user and allows for centralized management.

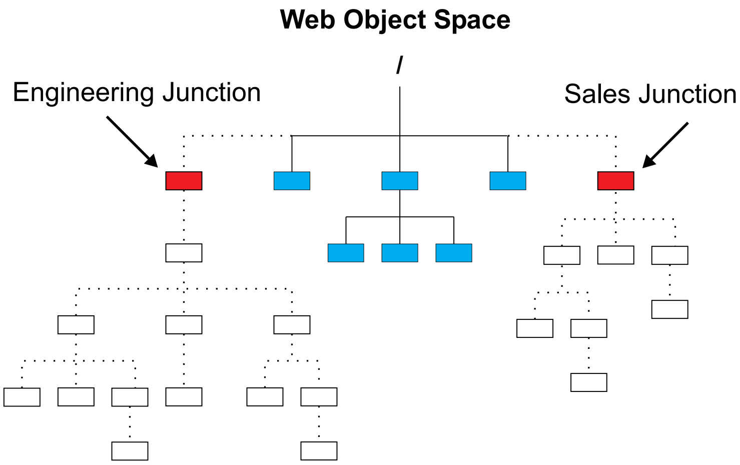


Figure 9. Unified web space

Replicated back-end servers

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

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

WebSEAL also correctly fails over when a server is down and starts reusing that server after it is restarted.

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

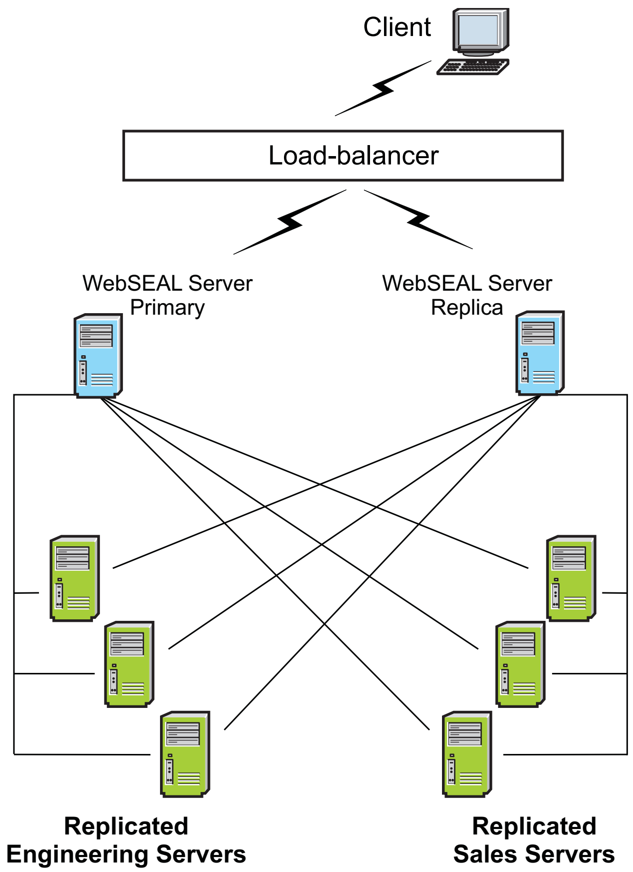


Figure 10. Replicated back-end servers

Server administration

You must understand the different server management tasks so that you can successfully administer WebSEAL. You can manage instances, deploy updates, synchronize data, configure cluster support, and access log files.

WebSEAL instance management

You can use the Reverse Proxy management page of the LMI to manage the WebSEAL instances on the appliance.

From the LMI top menu, select **Secure Web Settings** > **Manage** > **Reverse Proxy**. The Reverse Proxy management page displays.

You can use this page to manage each WebSEAL instance. Select the WebSEAL instance that you want to manage from the list of available instances. The available options for managing the WebSEAL instances are detailed in the following table:

Table 2. WebSEAL instance management

Option	Description
New	Create a WebSEAL instance.
Edit	Complete basic configuration.
Delete	Delete a WebSEAL instance.

Table 2. WebSEAL instance management (continued)

Option	Description
Start	Start a WebSEAL instance.
Stop	Stop a WebSEAL instance.
Restart	Restart a WebSEAL instance.
Refresh	Refresh the list of WebSEAL instances and the associated details.
Manage	Select from the following menu of management options: **Configuration** Provides advanced configuration options. You can modify the WebSEAL configuration file directly from this menu. **Troubleshooting** Includes troubleshooting resources such as trace and statistics. **Management Root** Provides access to the WebSEAL root files. **Junction Management** Includes junction creation and management. **Logging** Provides access to the WebSEAL log files.

Deploying WebSEAL updates in the LMI

When there are pending changes, a warning message is displayed at the top of the main pane. You can deploy or roll back the pending changes.

About this task

Any changes that affect running reverse proxy instances require a restart of the effected instances before the changes can take effect.

Procedure

1. Click the **Click here to review the changes or apply them to the system** link in the warning message.
2. In the Deploy Pending Changes page:
   - To view the details of changes that are made to a particular module, click the link to that module.
   - To deploy the changes, click **Deploy**.
   - To abandon the changes, click **Roll Back**.
   - To close the pop-up page without any actions against the changes, click **Cancel**.

Synchronization of WebSEAL data across multiple servers

You can use the WebSEAL **server sync** command to synchronize the configuration of one WebSEAL server with another.

Note: You can synchronize servers of the **same** type only. The WebSEAL server *type* is either a:

- WebSEAL running on an appliance.
- WebSEAL running on a standard operating system.

You can use the following list of server task commands for various tasks, including:

- Synchronizing one replicated WebSEAL server with another of the same type.
- Migrating one WebSEAL environment to another (for example, from test to production).

server sync
Used to synchronize the configuration of the supplied WebSEAL server to the current WebSEAL server. The **server sync** command invokes the other commands on this list for a complete synchronization operation. The data that can be synchronized includes configuration entries, the junction database, and selected data files, but not the object space or policy. Configuration entries and data files to be synchronized can be customized in the WebSEAL configuration file. For details, see "server task server sync" on page 658.

server restart
Used to restart the WebSEAL instance. For details, see "server task server restart" on page 656.

The following list describes the flow of communication for the **server sync** command:

1. The **server sync** command is issued from the administration console.
2. The request for data is issued from the WebSEAL server as a new server task command.
3. The source WebSEAL server gathers the data for synchronization and sends it to the target WebSEAL server.
4. The target WebSEAL server applies the data retrieved.

The request for data is issued from the WebSEAL server that is processing the **server sync** task. Data is pulled from one WebSEAL server to another with authorization automatically applied by the Security Access Manager **server task** framework. By using an existing communication channel, there is no need to open up more ports for the WebSEAL server.

Automating synchronization

You can configure more than one WebSEAL server in your environment. The term *cluster* refers to a group of WebSEAL servers that are configured to work together. You can use WebSEAL clusters to automate the synchronization of configuration information between different WebSEAL servers. Clustering can also improve system availability and performance.

Note: All cluster members must be the **same** server type. The WebSEAL server *type* for the cluster is either a:

- WebSEAL running on appliances.
- WebSEAL running on standard operating systems.

In a clustered environment, you must nominate a *master* server that is responsible for all cluster configuration changes. The remaining servers in the cluster are designated as *slaves*.

Whenever a slave is restarted, it checks the configuration information about the cluster master. If the slave's internal configuration information is not up to date then the slave automatically synchronizes with the master. You can synchronize all servers in the cluster by using the **cluster restart** server task command.

Cluster restart:

To synchronize the various WebSEAL servers, first select a master server. For example, `default-webseald-master.ibm.com` houses the generic configuration. Any configuration changes that you want made available to all servers must first be manually changed for the master server that includes the configuration files and junction definitions.

About this task

You must only modify the configuration information on the master. If you modify the configuration on a slave, you risk losing information during the next restart of the server when the slave synchronizes its configuration information with the master.

When you configure a WebSEAL cluster in your environment, you can issue a **`cluster restart`** server task command from the master server to apply any configuration changes and restart the updated servers. You can use the **`-ripple`** option to specify whether to restart the clustered WebSEAL servers in parallel or in sequence. There is also a **`-status`** option that you can use to monitor the progress of a cluster restart. See "server task cluster restart" on page 633.

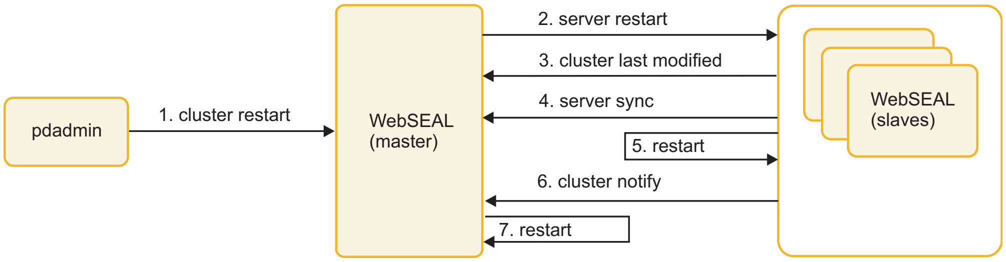


Figure 11. Cluster Support

Figure 11 shows the cluster restart process that involves the following high-level steps:

Procedure

1. Run the **`cluster restart`** server task command on the master server to initiate the restart.
2. The master issues a server restart command on each of the slaves in the cluster.

   Note: This is either done in parallel or in sequence depending on whether the **`-ripple`** option was used in the cluster restart command.
3. The slave retrieves the timestamp that indicates when the master configuration information was last modified.
4. If the configuration data stored locally on the slave is not up to date, then the slave synchronizes with the master configuration data.
5. If the slave configuration data was updated in step 4, then the slave server restarts. If the local configuration database on the slave did not need updating, then the slave does not restart.
6. The slave notifies the master that its operation is complete.

7. Once all slaves are updated and restarted where required, the master server restarts.

Configure WebSEAL for cluster support:

Use the `[cluster]` configuration stanza entries so that you can configure each of the WebSEAL servers in your clustered environment.

You must identify one server to be the master of the WebSEAL cluster. The **`is-master`** configuration entry is set to `yes` for the master server. You can also configure a **`max-wait-time`** on the master that represents the maximum number of seconds that the master server waits for each slave server to be restarted.

All other servers in the cluster must have the **`is-master`** configuration entry set to `no`. These servers are the slaves in the cluster. You must configure an additional configuration entry called **`master-name`** on each of the slaves. The **`master-name`** configuration entry specifies the authorization server name of the master server. For example, `default-webseald-master.ibm.com`.

For more information about **`[cluster] stanza`**, see the the Reference topics in the IBM Knowledge Center.

Note: For failover considerations in a clustered environment, see "Failover to a new master" on page 282.

Error message logging

WebSEAL uses a routing file to control the display of error messages and the logging of message data.

WebSEAL server error messages are formatted by using a standard format. By default, WebSEAL uses a single log file to record server error message data.

The WebSEAL routing file controls default message routing. The routing file can be modified for customized message logging.

For complete information about the WebSEAL routing file and error message logging, see the Troubleshooting topics in the IBM Knowledge Center.

WebSEAL server activity auditing

The Security Access Manager event logging mechanism can capture common events and activity that is generated by WebSEAL. Event logging provides a structured hierarchy for capturing information for auditing purposes. Event logging also supports the use of alternative destinations for logging output, such as files and remote servers.

To implement event logging, you use the **logcfg** event logging stanza entry to define one or more log agents (loggers) that gather a specific category of audit information from the event pool and direct this information to a destination.

The **logcfg** stanza entries are entered in the **[pdaudit-filter]** stanza of the `pdaudit.conf` configuration file. The `pdaudit.conf` configuration file is a component of the Common Auditing and Reporting Service (CARS).

WebSEAL still supports event logging configuration through the **[aznapi-configuration]** stanza in the WebSEAL configuration file. The format of the

logcfg event logging stanza entries remains the same whether entered in the **[pdaudit-filter]** stanza of the `pdaudit.conf` configuration file or the **[aznapi-configuration]** stanza of the WebSEAL configuration file.

For complete information about the event logging mechanism, see the Auditing topics in the IBM Knowledge Center.

Note: WebSEAL also supports traditional auditing and logging of HTTP events. Use this traditional mechanism only for situations that require compatibility with older installations of Security Access Manager. See “Traditional auditing and logging of HTTP events” and “Traditional auditing mechanism.”

Traditional auditing mechanism

Traditional auditing is configured by supplying a value for each the following stanza entries in the **`[aznapi-configuration]`** stanza of the WebSEAL configuration file.

```
[aznapi-configuration]
logaudit
auditcfg
logsize
logflush
```

Use of this method is comparable to the event logging method when directing output to a file. Note, however, that the event logging method provides more control over buffer size and event queues. Also, traditional auditing does not support output to consoles, pipes, or remote servers.

For more information about traditional auditing configuration settings for authentication, authorization, and HTTP events, see the Auditing topics in the IBM Knowledge Center.

Traditional auditing and logging of HTTP events

Configure the **[logging]** stanza of the WebSEAL configuration file so that you can audit and log HTTP events.

WebSEAL maintains three traditional HTTP log files that record HTTP activity:

- `request.log`
- `agent.log`
- `referer.log`

The request log can be customized by adding a `request-log-format` entry to the `[logging]` stanza. For more information about customized logging and traditional auditing and logging of HTTP events, see the Auditing topics in the IBM Knowledge Center.

HTTP resource access audit events can be filtered by document MIME-types and response codes by using the following stanza entries in the **[logging]** stanza of the WebSEAL configuration file:

```
[logging]
audit-mime-types
audit-response-codes
```

Configuration data log file

Use the configuration data so that you can troubleshoot problems with WebSEAL installation and operation.

Each WebSEAL instance records the complete contents of its configuration file in a log file every time that instance is started. Many WebSEAL problems are often the result of incorrect or incomplete configuration file settings. The logged configuration data provides an accurate snapshot of the current configuration settings for any WebSEAL instance.

When asked to troubleshoot a WebSEAL-related problem, IBM Support can rely on the information that is contained in this logged configuration data. The accuracy and completeness of this information increases the efficiency of the troubleshooting process. Alternatively, you can use this logged configuration data to perform your own diagnosis of a WebSEAL problem.

Configuration data log file location

Use the appliance local management interface to access the configuration data log files.

Select **Secure Web Settings** > **Manage** > **Reverse Proxy**. Select the appropriate WebSEAL instance and click **Manage** > **Logging**.

Notes on configuration data log file growth

Take note of the following details about the size of log files.

- Only 1 log file is created when a WebSEAL instance is started for the first time.
- Data from the complete WebSEAL configuration file is recorded in the log file each time that WebSEAL instance is started.
- The configuration file data for each subsequent startup is appended to the same log file.
- Each entry in the log file begins with a timestamp. The timestamp entry distinguishes each data entry from the others.
- The log file continues to grow in size with each new entry. You can use the local management interface to control the size of this file.

Configuration data log file format

In the WebSEAL configuration file, a default value for a stanza entry is provided by WebSEAL during initial installation and configuration. If an administrator later modifies a value in the configuration file, that custom value becomes non-default.

In the configuration data log file, a special marker (**`[default]`**) identifies any stanza entry value that is default. Non-default values do not contain this marker. For example:

```
...
https = yes
https-port = [default] 443
http = yes
http-port = [default] 80
...
```

The information in the configuration data log file is grouped according to the same stanzas that occur in the actual WebSEAL configuration file. The following partial view of a configuration data log file shows stanza entries and values that are grouped by stanzas. This example also includes a sample timestamp line:

```
============================================
Configuration Data Logged: Fri Jul 23 15:37:02 2004
...
[gso-cache]
gso-cache-enabled = [default] no
gso-cache-size = [default] 1024
```

```
gso-cache-entry-lifetime = [default] 900
gso-cache-entry-idle-timeout = [default] 120

[ltpa-cache]
ltpa-cache-enabled = [default] yes
ltpa-cache-size = [default] 4096
ltpa-cache-entry-lifetime = [default] 3600
ltpa-cache-entry-idle-timeout = [default] 600

[ba]
ba-auth = [default] https

[forms]
forms-auth = [default] none


[certificate]
accept-client-certs = [default] never
cert-cache-max-entries = [default] 1024
cert-cache-timeout = [default] 120


...
```

Messages relating to the configuration data log file

A serviceability message is entered in the WebSEAL log file that specifies the location of the log file every time a configuration data log file is created.

```
"The configuration data for this WebSEAL instance has been logged in %s"
```

The **%s** macro is replaced by the string containing the name of the log file.

If for any reason the configuration file data cannot be logged during a WebSEAL startup, the WebSEAL startup proceeds without interruption and an error message is recorded in the WebSEAL log file:

```
"An error occurred trying to log the WebSEAL configuration data at startup."
```

Statistics

WebSEAL provides a series of built-in software modules to monitor specific server activity and collect information about those activities. You can view the statistics from any active module. You can also direct the statistics information to the log files.

Each active module captures statistics information. You can configure WebSEAL to capture statistics at regular intervals and view a snapshot of this information at any time.

The appliance collects flow data. Flow data provides performance and throughput statistics for WebSEAL. The appliance records the statistics in a separate database for each WebSEAL instance. You can use the local management interface to retrieve these statistics.

Use the **[flow-data]** and **[user-agent]** stanzas to configure the flow data statistics for the appliance. For more information, see the Administration topics in the IBM Knowledge Center.

Trace utility

The **trace** utility captures information about error conditions and program control flow in Security Access Manager, WebSEAL, and Plug-in for Web Servers. This information is stored in a file and used for debugging and problem determination purposes.

The **trace** utility is provided primarily to assist support personnel in diagnosing problems that occur with the functions of the Security Access Manager. As a user, you might find some of the Security Access Manager, WebSEAL, and Plug-in for Web Servers tracing components useful. However, most of the components are of little benefit unless you are diagnosing complex problems with the assistance of technical support personnel.

Trace data is intended primarily for use by IBM Tivoli Software Support personnel and might be requested as part of diagnosing a reported problem. However, experienced product administrators can also use trace data to diagnose and correct problems in the Security Access Manager environment.

You can use the appliance local management interface to access the trace files. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**. Select the appropriate WebSEAL instance and click **Manage** > **Troubleshooting** > **Tracing**.

Chapter 2. Configuration

Web server configuration

Configure the web server capabilities of WebSEAL such as content caching, communication protocol, LDAP directory server, worker thread, and HTTP data compression.

Specifying the WebSEAL host name

Typically, the name of the WebSEAL host computer is automatically determined when this information is required. There are situations, such as with virtual host junctions, where the WebSEAL host can use several names. On systems with many host names, interfaces, or WebSEAL instances, the automatic determination might not be correct for a specific situation. You can specify the correct one.

Before you begin

You must understand the WebSEAL server name in the following situations.

In the configuration file

The name uniquely identifies a WebSEAL server process. You can install and configure multiple WebSEAL servers on one computer system. Therefore, each WebSEAL server process must have a unique name. Each WebSEAL server process is known as an *instance*.

Each WebSEAL instance has its own configuration file. The `server-name` stanza entry in the `[server]` stanza of the configuration file for each WebSEAL instance specifies the unique name for that WebSEAL instance.

The `server-name` stanza entry is a combination of the host name of the physical computer where WebSEAL is installed and the WebSEAL instance name. Both names are specified during WebSEAL configuration.

```
[server]
server-name = host_name-instance_name
```

A computer host name always has a fully qualified name, for example, `abc.ibm.com`). It also can have a short name, for example, `abc`. When prompted for the host name during WebSEAL configuration, you can specify either the fully qualified name or the short name. In the following example, the WebSEAL instance name `web1` is on a computer with a fully qualified host name of `abc.ibm.com`, as specified during WebSEAL configuration.

```
[server]
server-name = abc.ibm.com-web1
```

The initial WebSEAL server is automatically assigned an instance name of `default`, unless you modify this name during WebSEAL configuration. For example:

```
[server]
server-name = abc.ibm.com-default
```

In the `pdadmin server list` command

The instance name also affects how the WebSEAL server is listed with the **`pdadmin server list`** command. Because the **`pdadmin`** command serves the entire Security Access Manager family, a product component name is required in the command syntax. The component name for WebSEAL is `webseald`. For the **pdadmin server list** command, the WebSEAL server name has the following format.

```
instance_name-webseald-host_name
```

The following example shows the output from **pdadmin server list** for the instance `web1` installed on the host `abc.ibm.com`:

```
web1-webseald-abc.ibm.com
```

The following **pdadmin server list** command output displays an initial `default` WebSEAL server and a second WebSEAL instance named `web1`:

```
pdadmin> server list
web1-webseald-abc.ibm.com
default-webseald-abc.ibm.com
```

In the protected object space

Each WebSEAL instance is represented as a member of the `/WebSEAL` container object in the protected object space. Two WebSEAL instances (`default` and `web1`), on the host `abc.ibm.com`, appear in the protected object space in the following format:

```
/WebSEAL/abc.ibm.com-web1
/WebSEAL/abc.ibm.com-default
```

About this task

You can manually specify the host name in the `web-host-name` stanza entry in the `[server]` stanza of the WebSEAL configuration file. The value must be the fully qualified name. This manual setting resolves any conflicts in determining the host name that is used, for example, by WebSEAL HTTP/HTTPS responses and authentication mechanisms in a traditional junction environment.

By default, `web-host-name` is not enabled and has no value. When required, WebSEAL attempts to automatically determine the host name.

Procedure

1. Stop the WebSEAL server process.
2. Manually edit the WebSEAL configuration file to provide a value for the stanza entry.
3. Uncomment the line.
4. Restart WebSEAL.

Example

```
[server]
web-host-name = abc.ibm.com
```

Notice the difference in syntax between the **server-name** and the **web-host-name** values. For example:

```
[server]
server-name = abc.ibm.com-default
web-host-name = abc.ibm.com
```

Modifying the configuration file settings

The operation of the WebSEAL server is controlled by using the WebSEAL configuration file and a corresponding obfuscated file that is used for sensitive data. Use the local management interface to modify the configuration file.

About this task

For details on the stanza entries that you can use in the WebSEAL configuration file, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.

Configuration file organization

The configuration file contains sections that control specific portions of WebSEAL. Each section contains further divisions that are called stanzas.

Stanza labels appear in brackets.

```
[stanza_name]
```

For example, the **[ssl]** stanza defines the SSL configuration settings for use by the WebSEAL server.
Each stanza in a Security Access Manager configuration file contains one or more *stanza entries*. A stanza entry consists of a *key value pair*, which contains information that is expressed as a paired set of stanza entries.
Each stanza entry has the following format:
key = value
The initial installation of WebSEAL establishes many of the default values. Some values are static and never change; other values can be modified to customize server function and performance.
The ASCII-based text file can be edited with a common text editor.

Configuration file name and location

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

```
webseald-instance_name.conf
```

The administrator can use the local management interface to configure more WebSEAL instances and specify each new *instance_name*.
The configuration utility uses the specified *instance_name* to name the new WebSEAL configuration file. For example, if you name the new WebSEAL instance **webseal2**, the following configuration file is created:

```
webseald-webseal2.conf
```

For more information about WebSEAL instance configuration, see "WebSEAL instance deployment" on page 583.

Procedure

1. Log in to the local management interface.
2. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select the appropriate WebSEAL instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.

   Note: To complete basic configuration file updates, select the WebSEAL instance and click **Edit**.
5. Make the required changes to the configuration.
6. Save your changes. The local management interface displays a warning message that states about an undeployed change.
7. Click **Click here to review the changes or apply them to the system**. Review the change and click **Deploy**. A System Warning displays to indicate that the deployment is complete and a restart is required.

8. Restart the WebSEAL instance from the **Reverse Proxy Management** page for these changes to take effect.

Content caching

WebSEAL can cache static web contents to increase the response time of a transaction. You must understand the key concepts, configuration variables, and conditions that affect content caching and the impact of HTTP headers. You can flush all caches and set cache control for specific documents.

Key caching concepts

Users can often experience extended times for network access and file download due to poor web document retrieval performance. Poor performance can occur because the WebSEAL server is waiting for documents that are retrieved from junctioned back-end servers.

Caching web content gives you the flexibility of serving documents locally from WebSEAL rather than from a back-end server across a junction. With the content caching feature, you can store commonly accessed web document types in the WebSEAL server memory. Clients can experience much faster response to follow up requests for documents that were cached in the WebSEAL server.

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

Caching is based on MIME type. When you configure WebSEAL for content caching, identify the following settings:

- Document MIME type
- Type of storage medium
- Size of storage medium
- Maximum age if expiry information is missing from the original response

Configuration of content caching

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

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

Variable	Description
mime-type	Represents any valid MIME type conveyed in an HTTP `Content-Type:` response header. This value can contain an asterisk (*). A value of `*/*` represents a default object cache that holds any object that does not correspond to an explicitly configured cache. **Note:** The asterisk is a wildcard only for a MIME-type directory and its contents. The asterisk is not a wildcard for regular expression matching.
cache-type	Specifies the type of storage medium to use for the cache. This release of Security Access Manager supports only "memory" caches.

Variable	Description
cache-size	Specifies the maximum size in KB that the specified cache can grow before objects are removed according to a Least Recently Used algorithm.
def-max-age	Specifies the maximum age (in seconds) if expiry information is missing from the original response. If no value is provided, a default maximum age of 3600 (1 hour) is applied.

Conditions affecting content caching configuration

The content caching mechanism observes the following conditions:

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

Impact of HTTP headers on WebSEAL content caching

The following table describes how specific

- HTTP Request headers from the client affect whether WebSEAL uses the cache to produce the requested resource or send the request to the destination server.
- HTTP Response headers from the junction affect whether WebSEAL allows caching of the resource,

Note: WebSEAL does not process **<meta>** tags with **http-equiv** attributes for Response headers.

Response Header	Impact of HTTP Response headers in the request	Impact of HTTP Request Headers in the request
Accept-encoding	Not applicable	Allows the response to come from the cache if the value matches the cached encoding type.
Age	Calculates whether data in the cache is fresh enough to use.	Not applicable
Authorization	Not applicable	If the **document-cache-control=public** POP value is not set, stops the response from the cache unless the junction has the **-b** filter set. See Cache control for specific documents.

Response Header	Impact of HTTP Response headers in the request	Impact of HTTP Request Headers in the request
Age	Calculates whether data in the cache is fresh enough to use.	Not applicable
Cache-control	• The **no-cache**, **no-store**, or **private** values stop the resource from being stored in the content cache. • The **public** value stores the resource in the content cache even if user-identifying data is passed to the junction.	• The **no-cache** stops the response from the cache. • The **no-store** value stops the response from the cache and being stored in the cache. • The **max-age**, **max-stale**, or **min-fresh** values determine whether the cache is used for the response.
Content-range	Stops the resource from being stored in the content cache.	Not applicable
Date **Note:** If the value of **Date** is greater than the value of **Expires**, the resource is not stored in the cache, which complies with HTTP/1.0 specification.	Calculates whether data in the cache is fresh enough to use.	Not applicable
Expires	• Stores the resource in the content cache even if user-identifying data is passed to the junction. • Calculates whether data in the cache is fresh enough to use.	Not applicable
Last date	Calculates whether data in the cache is fresh enough to use.	Not applicable
Pragma	The **non-cache** value stops the resource from being stored in the content cache.	The **non-cache** value stops the response from the cache.
Range	Not applicable	Stops the response from the cache and the response from being stored in the cache.
Transfer-encoding	WebSEAL strips the `TE:` header from the request sent to the junction, so it does not expect a `Transfer-Encoding:` header to be sent back. However, if a `Transfer-Encoding:` header is sent back, WebSEAL stops the resource from being stored in the content cache.	Not applicable

Response Header	Impact of HTTP Response headers in the request	Impact of HTTP Request Headers in the request
Vary	Stops the resource from being stored in the content cache.	Not applicable

Impact of other conditions on WebSEAL content caching

Other conditions affect WebSEAL content caching.

- WebSEAL does not cache content if the response from the junction does not have the status `200 OK`.
- WebSEAL does not cache content if the request URL contains a query string.
- WebSEAL flushes a cache entry if a **PUT**, **POST**, or **DELETE**is used in the URL.
- WebSEAL returns only values from the cache for **HEAD** and **GET** requests.
- WebSEAL does not cache the response from junctions to**HEAD** requests.
- WebSEAL does not cache the response if the junction has `-b gso`, `-b supply`, `C`, or -cset, unless one of the following conditions exists:
  - There is a POP on the object with value `document-cache-control=public`. See Cache control for specific documents.
  - The response has an **Expires:** header.
  - The response has **Cache-Control:** public set.
- WebSEAL does not cache the response if there is a POP on the object with the value `document-cache-control=no-cache`. See Cache control for specific documents.
- You cannot override the calculations that are based on date headers (**Date**, **Age**, **Last-modified**, **Expires**, and the related **Cache-control** header values.
- You cannot override all other headers.

Flushing all caches

You can use the **pdadmin** utility to flush all configured content caches. You cannot flush individual caches with **pdadmin**.

Note: You must log in to the secure domain as the Security Access Manager administrator `sec_master` before you can use **pdadmin**.

To flush all content caches, use the following **pdadmin** command:

```
pdadmin> server task instance_name-webseald-host_name cache flush all
```

Cache control for specific documents

You can control caching for specific documents by attaching a special protected object policy (POP) to those objects. This POP must contain an extended attribute called **document-cache-control**.

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

Value	Description
no-cache	The **no-cache** value instructs WebSEAL not to cache this document. All children of the object with the POP also inherit the POP conditions.

Value	Description
public	The **public** value allows WebSEAL to cache the document by ignoring the fact that the junction was created with a **–c** or **–C** option. In addition, this value also allows caching of this document when the request is sent with an authorization header (such as Basic Authentication). This condition also includes a request where WebSEAL inserts BA information on behalf of the client (such as with GSO or **–b supply** junctions). Normally, proxy servers do not cache the response documents to requests that include authorization headers.

Use the **pdadmin pop create**, **pdadmin pop modify**, and **pdadmin pop attach** commands to set a POP on a protected object.

The following example illustrates creating a POP called "doc-cache" (with the **document-cache-control** extended attribute) and attaching it to an object (`budget.html`):

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

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

Details about the **pdadmin** command-line utility can be found in the Command Reference topics in the IBM Knowledge Center.

Communication protocol configuration

You can configure the WebSEAL communication protocols to control how WebSEAL handles requests and creates connections. There are many stanza entries available to configure the communication protocols.

Configuring WebSEAL HTTP requests

Configure the WebSEAL HTTP requests so that you can define how WebSEAL handles HTTP requests from unauthenticated users.

About this task

It is typical to allow anonymous users read-only access to selected documents on the public section of your website. Enable or disable access to selected documents by editing the stanza entries for handling HTTP requests over TCP in the **[server]** stanza of the WebSEAL configuration file.

Procedure

1. Log in to the local management interface.
2. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select your instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.
5. Add the following line under the **[server]** stanza.

```
[server]
http = {yes|no}
```

6. Set the HTTPS access port value by adding the following line under the **[server]** stanza. IBM HTTP Server, WebSphere® Application Server, and WebSEAL all use port 80 as the default port.

```
[server]
http-port = 80
```

7. Save and deploy your changes.

Configuring WebSEAL HTTPS requests

Configure WebSEAL HTTPS requests to determine how WebSEAL handles HTTPS requests from unauthenticated users.

Procedure

1. Log in to the local management interface.
2. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select your instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.
5. Enable or disable HTTPS access by adding the following line under the **[server]** stanza.

   ```
   [server]
   https = {yes|no}
   ```

6. Set the HTTPS access port value by adding the following line under the **[server]** stanza.

   The default port for HTTPS access is 443:

   ```
   [server]
   https-port = 443
   ```

   To change to port 4343, for example, set:

   ```
   [server]
   https-port = 4343
   ```

Restrictions on connections from specific SSL versions

The stanza entries that control connections for specific SSL and TLS versions are in the **[ssl]** stanza of the WebSEAL configuration file. By default SSL version 2 is disabled. All other SSL and TLS versions are enabled by default.

You can independently enable and disable connectivity for the following communication protocol versions:

- Secure Sockets Layer (SSL) version 2
- SSL version 3
- Transport Layer Security (TLS) version 1
- TLS version 1.1
- TLS version 1.2

```
[ssl]
disable-ssl-v2 = {yes|no}
disable-ssl-v3 = {yes|no}
disable-tls-v1 = {yes|no}
disable-tls-v11 = {yes|no}
disable-tls-v12 = {yes|no}
```

Persistent HTTP connections

At the HTTP communication layer, WebSEAL maintains persistent connections between the client browser and WebSEAL, and between junctioned web servers and WebSEAL.

Client connections are controlled by the following entries in the **[server]** stanza of the WebSEAL configuration file:

max-idle-persistent-connections
This entry controls the maximum number of idle client persistent connections.

persistent-con-timeout
This entry controls the maximum number of seconds that WebSEAL holds an HTTP persistent connection open for a new request before the connection is shut down.

disable-timeout-reduction
This entry determines whether WebSEAL reduces the timeout duration to help control the number of active worker threads. By default, WebSEAL automatically reduces the timeout duration for threads as the number of in-use worker threads increases.

Junction connections are controlled by the following entries in the **[junction]** stanza of the WebSEAL configuration file:

max-cached-persistent-connections
This entry controls the maximum number of persistent connections that will be stored in the cache for future use.

persistent-con-timeout
This entry controls the maximum number of seconds a persistent connection can remain idle in the cache before WebSEAL closes the connection.

Note: WebSEAL supports HTTP/1.1 persistent connections. HTTP/1.0 persistent connections are not supported.

WebSEAL configuration for handling HTTPOnly cookies

To help reduce the risk of cross-site scripting, an `HTTPOnly` attribute was added to cookies, preventing them from being accessed through client-side scripts.

Cross-site scripting is among the most common security problems for web servers and can expose sensitive information about the users of a website. WebSEAL includes the option to enable WebSEAL to add the `HTTPOnly` attribute to the `Set-Cookie` headers it uses for sessions, failover, and LTPA cookies. WebSEAL can also be configured to pass the `HTTP-only Set-Cookie` header attribute from back-end junction servers to web browsers.

To configure WebSEAL to add the `HTTPOnly` attribute to Session, Failover and LTPA Set-Cookie headers, change the value of **`use-http-only-cookies`** in the **`[server]`** stanza of the WebSEAL configuration file to `yes`. The default value is `no`.

```
[server]
use-http-only-cookies = yes
```

To configure WebSEAL to pass the `HTTPOnly` attribute from Set-Cookie headers sent by junctioned servers, change the value of **`pass-http-only-cookie-attr`** in the **`[junction]`** stanza of the WebSEAL configuration file to `yes`. The default value is `no`.

```
[junction]
pass-http-only-cookie-attr = yes
```

For more information about these entries, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.

Configuring WebSEAL connection timeout settings

Edit the WebSEAL configuration file so that you can configure the timeout settings for HTTP and HTTPS communication.

About this task

The following flow diagram shows where the timeout settings affect an example request and response exchange. The number of fragments that are indicated for the request and the response are for sample purposes only.

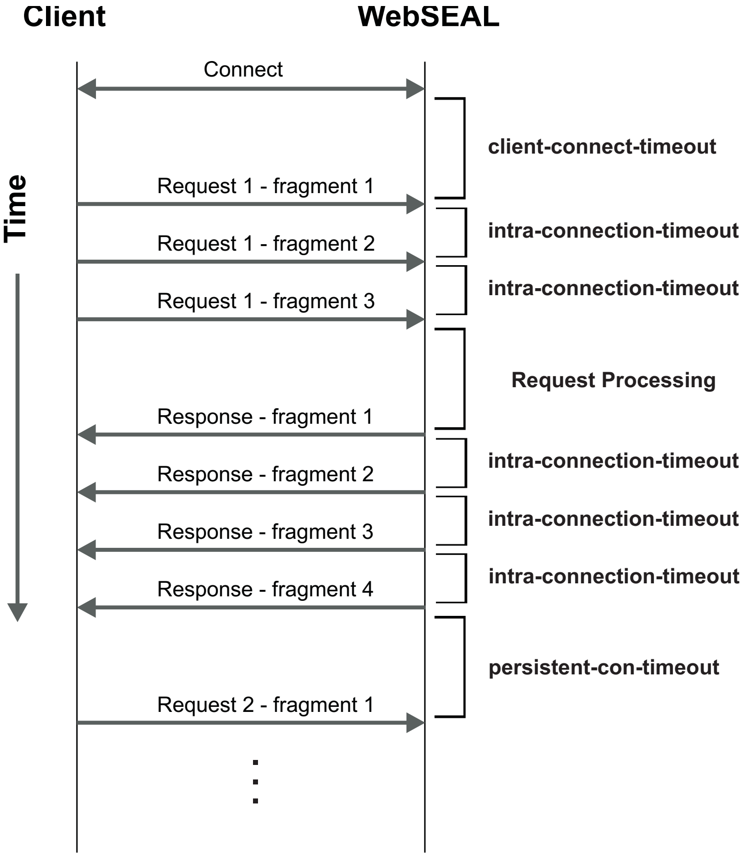


Figure 12. Timeout settings for HTTP and HTTPS communication

The stanza entries for timeout settings are in the `server` stanza of the WebSEAL configuration file.

client-connect-timeout

Specifies how long WebSEAL holds the connection open for the initial HTTP or HTTPS request after the initial connection handshake. The default value is 120 seconds.

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

intra-connection-timeout

Specifies the number of seconds, between each request data fragment when request and response data is sent as two or more fragments.

The stanza entry also governs the timeout between response data fragments after the first data fragment is returned by WebSEAL. The default value is 60 seconds.

```
[server]
intra-connection-timeout = 60
```

If the value of this stanza entry is 0, connection timeouts between data fragments are governed by the **client-connect-timeout** stanza entry. The exception to this rule occurs for responses that are returned over HTTP or TCP. In this case, there is no timeout between response fragments.

If a connection timeout occurs on a non-first data fragment due to the **intra-connection-timeout** setting, a TCP reset packet is sent.

persistent-con-timeout

Controls the maximum number of seconds that WebSEAL holds an HTTP persistent connection open for a new client request before the connection is shut down.

The HTTP persistent connection opens after the HTTP request and server response exchange is complete. The default value is 5 seconds.

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

If the value of this stanza entry is 0, the connection does not remain open for future requests. A value of zero causes WebSEAL to set the `Connection: close` header and then close the connection on every response.

Note: The timeout setting on the junction side is set by the **persistent-con-timeout** entry in the `[junction]` stanza.

Procedure

1. Log in to the local management interface.
2. From the top menu, select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select your instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.
5. Edit the stanza by using the information in About this task.
6. Save and deploy your changes.

WebSEAL server timeout settings

The WebSEAL server timeout settings are set in the WebSEAL configuration file.

Stanza Entry	Description	Default Value

`[junction]`
`http-timeout`
	The timeout value for sending to and reading from a back-end server over a TCP junction.	120

`[junction]`
`https-timeout`
	The timeout value for sending to and reading from a back-end server over an SSL junction.	120

Stanza Entry	Description	Default Value
[junction] ping-time	WebSEAL performs a periodic background ping of each junctioned server to determine whether it is running. This value sets the time interval between these pings. To turn off the ping, set this entry to zero. If this entry is set to zero, the recovery-ping-time must be set.	300
[junction] recovery-ping-time	This optional entry sets the interval, in seconds, between pings when the server is determined **not** to be running. If this entry is not set, the value defaults to the ping-time setting.	300

Optionally, you can also customize the **http-timeout** and **https-timeout** settings for any particular WebSEAL junction by manually creating a junction-specific stanza for example, [junction:/WebApp] in the WebSEAL configuration file.

The stanza can be customized for a particular junction by adding the adjusted configuration items to a [junction:*junction_name*] stanza, where junction_name refers to the junction point for a standard junction (including the leading '/'), or the virtual host label for a virtual host junction. If specified, the following settings override the default settings that are indicated in the previous table.

Stanza Entry	Description	Default Value (seconds)
[junction:*junction_name*] http-timeout	The timeout value for sending to and reading from a back-end server over a specific TCP junction.	120
[junction:*junction_name*] https-timeout	The timeout value for sending to and reading from a back-end server over a specific SSL junction.	120

WebDAV support

Web-based Distributed Authoring and Versioning (WebDAV) is a set of extensions to the HTTP protocol that users can use to edit and manage files on remote web servers.

WebSEAL supports the use of WebDAV methods in client requests for junctioned applications. WebSEAL does not support WebDAV methods when it acts as a web server that serves local content. Authorization for all WebDAV methods is controlled through the *r* ACL bit.

WebSEAL forwards requests to junctioned applications by using the following WebDAV methods:

- ACL

- BASELINE-CONTROL
- BCOPY
- BDELETE
- BIND
- BMOVE
- BPROPPATCH
- CHECKIN
- CHECKOUT
- COPY
- LABEL
- LINK
- LOCK
- MERGE
- MKACTIVITY
- MKCOL
- MKREDIRECTREF
- MKWORKSPACE
- MOVE
- NOTIFY
- ORDERPATCH
- PATCH
- POLL
- PROPFIND
- PROPPATCH
- REBIND
- REPORT
- SEARCH
- SUBSCRIBE
- SUBSCRIPTIONS
- UNBIND
- UNCHECKOUT
- UNLINK
- UNLOCK
- UNSUBSCRIBE
- UPDATE
- UPDATEDIRECTREF
- VERSIONCONTROL

Microsoft RPC over HTTP support

Microsoft Remote Procedure Call over HTTP is a protocol that is used to communicate RPC traffic over an HTTP connection. Microsoft Outlook clients use this protocol to access Microsoft Exchange servers over HTTP.

WebSEAL does not natively support Microsoft RPC over HTTP operations. WebSEAL does support the forwarding of RPC over HTTP version 2 methods to junctioned applications. Authorization for all RPC over HTTP methods is controlled through the *r* ACL bit.

WebSEAL forwards requests to junctioned applications by using the following RPC over HTTP methods:

- RPC_IN_DATA
- RPC_OUT_DATA

Chunked transfer coding support

WebSEAL supports chunked transfer coding as defined by the HTTP/1.1 specification.

Chunks from the client are transmitted as they are received to junctioned web servers.

IPv4 and IPv6 overview

Beginning with Tivoli Access Manager for Web version 6.0, WebSEAL supports Internet Protocol version 6 (IPv6).

IPv6 improves upon IPv4 in the following ways:

- IPv6 allocates 128 bits for the address space; IPv4 only allocates 32 bits for the address space.
- IPv6 can decrease the size of static, non-default routing tables, used to route packets through the Internet backbone.
- IPv6 provides end-to-end security by requiring adherence to the IP Security protocols (IPSec).

The primary format of an IPv4 address is a 32-bit numeric address that is written as four numbers that are separated by periods. For example:

```
x.x.x.x
```

The valid range for each number is zero to 255. For example:

```
1.160.10.240
```

One primary format of an IPv6 address is a 128-bit numeric address that is written as eight numbers that are separated by colons. For example:

```
x:x:x:x:x:x:x:x
```

The valid range for each number is zero to ffff (hexadecimal). For example:

```
fec0:fff:0000:0000:0000:0000:0000:1
```

The IPv6 address can be expressed in an abbreviated form by collapsing the contiguous fields that contain only zeros. For example, `0009:0000:0000:0000:0000:0008:0007:0006` can be represented as `9::8:7:6`.

Refer to the RFC 2373 standard to determine what constitutes a valid representation of an IPv6 address. IBM Security Access Manager supports any of the valid forms for an IPv6 address that is described in Section 2.2 of RFC 2373. IBM Security Access Manager does not support prefix notation for a netmask.

Configuring WebSEAL for IPv6 and IPv4 requests

By default Security Access Manager WebSEAL, version 6.0 or later, supports IPv6 networks. You can configure it to support either IPv4 only or both IPv4 and IPv6.

About this task

You determine the supported network by setting the following values in the **ipv6-support** stanza.

yes IPv6 and IPv4 networks are supported. This value is the default setting.

When both IPv6 and IPv4 networks are supported, WebSEAL listens for incoming HTTP and HTTPS requests that are IPV4 protocol-based or IPV6 protocol-based.

If the host name for a junction maps only to an IPv6 address, then IPv6 is used. If the host name maps to an IPv4 address, then an IPv4 address is used.

If the DNS name maps to both an IPv6 and an IPv4 address, then the choice is arbitrary. To specify a particular protocol, use the IPv6 or IPv4 address of the host when you create the junction.

If no IPV6 interface is available, WebSEAL listens only for IPv4 requests. If there are both IPv6 and IPv4 interfaces available, by default WebSEAL listens for requests on both protocols. If you do not want WebSEAL to accept IPv6 connections, set `ipv6-support = no`.

no Only IPv4 networks are supported.

Procedure

1. Log in to the local management interface.
2. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select your instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.
5. Edit the **ipv6-support** stanza.
   ```
   [server]
   ipv6-support = {yes|no}
   ```
6. Save and deploy your changes.

IPv6: Compatibility support

Before you enable IPv6 support, you must understand how IP version compatibility is maintained for previous versions of Security Access Manager.

- If the **ipv6-support** is set to `no`, then WebSEAL provides support for IPv4 as it did in previous releases, but does not support IPv6.

  For example, the **iv-remote-address-ipv6** HTTP header and `XAUTHN_IPADDR_IPV6` identifier, which is used with the external authentication C API are not available. For more information, see the Reference topics in the IBM Knowledge Center.
- If **ipv6-support** is set to `yes`, IPv6 is supported. Attributes containing IPv4 addresses continue to hold IPv4 addresses. Custom modules that are written for previous releases still continue to work.

  However, if WebSEAL passes an IPv6-only address to an older custom module that is not written to support IPv6 format, the older module might require updating to handle the IPv6 address format.

  The address range for IPv6 is larger than the range available to IPv4. With older modules, WebSEAL maps an IPv6 address to an IPv4 format when possible. For example, the IPv6 address `::c0a8:1` maps to the IPv4 address `192.168.0.1`.

If the IPv6 address exceeds the range for IPv4, WebSEAL maps the address by default to `0.0.0.0` in IPv4 format. For example, the IPv6 address `fec0::1` has no IPv4 equivalent and therefore is mapped to the IPv4 address `0.0.0.0`.

IPv6: Upgrade notes

When you upgrade to Security Access Manager WebSEAL version 7.0 from a previous version, IPv6 support is automatically disabled.

The upgrade module sets `ipv6-support = no` in the instance-specific configuration file. After an upgrade, you can enable support for IPv6 by manually editing the WebSEAL configuration to change the value of **ipv6-support** to `yes`.

For more information, see "Configuring WebSEAL for IPv6 and IPv4 requests" on page 39.

IP levels for credential attributes

Network information can be stored as an extended attribute in a user's credential. You can control the amount of network information that is stored in a credential by specifying the required IP level.

There are different format structures available to hold IPv4 and IPv6 information. Adding attributes to credentials can affect WebSEAL performance.

The following values are available in the **ip-support-level** stanza entry in the **[server]** stanza of the WebSEAL configuration file.

- **displaced-only**

  WebSEAL generates the IPv4 attribute only when it builds user credentials and when authenticating users through external authentication C API modules.

  This value is the default for migrated WebSEAL installations (`ipv6-support=no`):

  ```
  [server]
  ip-support-level = displaced-only
  ```

  This value is not permitted when `ipv6-support=yes`.

- **generic-only**

  WebSEAL generates new generic attributes that support both IPv4 and IPv6 only when it builds user credentials and when authenticating users through external authentication C API modules.

  This value is the default for new WebSEAL installations (`ipv6-support=yes`):

  ```
  [server]
  ip-support-level = generic-only
  ```

- **displaced-and-generic**

  Both sets of attribute types (produced by **displaced-only** and **generic-only**) are used when it builds user credentials and when authenticating users through external authentication C API modules.

  ```
  [server]
  ip-support-level = displaced-and-generic
  ```

LDAP directory server configuration

When Security Access Manager is configured to use an LDAP-based user registry, such as IBMTivoli Directory Server, WebSEAL must be configured as an LDAP client so it can communicate with the LDAP server.

The location of the LDAP server and its configuration file `ldap.conf` is provided during Security Access Manager runtime configuration. A combination of stanza entries and values from the `ldap.conf` and the WebSEAL configuration file `webseald.conf` provides the appropriate information to WebSEAL as the LDAP client.

- WebSEAL determines that the configured user registry is an LDAP-based directory server.
- The following stanza entries in the **[ldap]** stanza of `webseald.conf` are valid:

  ```
  host
  port
  ssl-port
  max-search-size
  replica
  auth-using-compare
  cache-enabled
  prefer-readwrite-server
  ssl-enabled
  ssl-keyfile
  ssl-keyfile-dn
  timeout
  auth-timeout
  search-timeout
  default-policy-override-support
  user-and-group-in-same-suffix
  login-failures-persistent
  ```
- Additionally, the values for the following stanza entries in `ldap.conf` override any existing values in `webseald.conf`:

  ```
  host
  port
  ssl-port
  max-search-size
  replica
  ```

For information about the stanza entries, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.

WebSEAL worker thread configuration

The number of configured worker threads specifies the number of concurrent incoming requests that can be serviced by a server. You can set the number of threads available to service incoming connections to WebSEAL.

Other connections that arrive when all worker threads are busy are buffered until a worker thread is available.

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

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

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

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

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

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

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

WebSEAL worker threads

WebSEAL draws from its pool of worker threads to process multiple requests. Worker threads handle incoming requests to applications on multiple junctioned back-end servers.

However, the worker thread pool can be quickly drained if a particular back-end application is unusually slow when it processes a high volume of requests. A depletion of the worker thread pool by this one application renders WebSEAL incapable of responding to requests for services on the remaining junctioned application servers.

When no worker threads are available to handle incoming requests, users experience a WebSEAL server that is not responding.

You can adjust the worker threads value according to your WebSEAL implementation. The number of worker threads available to WebSEAL is specified by the **worker-threads** stanza entry in the WebSEAL configuration file.

The configuration mechanism maintains a fair distribution of worker threads across all junctions and prevents depletion of the worker thread pool by any one junction. This configuration also prevents any application from claiming more than its share of worker threads.

To configure the allocation of WebSEAL worker threads across multiple junctions, see “Global allocation of worker threads for junctions.”

To configure the allocation of WebSEAL worker threads for every junction, see “Per-junction allocation of worker threads for junctions” on page 44.

Global allocation of worker threads for junctions

You can modify the entries in the [junction] stanza of the WebSEAL configuration file to control the global allocation of worker threads across all junctions for a particular WebSEAL server.

The values that are used for these stanza entries are expressed as percentages within the range of 0 to 100.

- **worker-thread-soft-limit**

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

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

The default value is 90 percent.

- **worker-thread-hard-limit**

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

  For example, when **worker-threads = 50**, a setting of 80 (%) causes error messages to be displayed when the junction tries to use more than 40 worker threads. All requests that represent greater than 40 worker threads on the junction are returned with a **503 Service Unavailable** message.

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

These global settings apply equally to all configured junctions. When you configure these two stanza entries, it is logical to set the soft limit to a lower value than the hard limit.

Per-junction allocation of worker threads for junctions

Use the pdadmin command so that you can limit worker thread consumption on a per-junction basis.

Specify hard and soft worker thread limits on a specific junction by using the following options in the **pdadmin server task create** command.

- –l `percent_value`

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

- –L `percent_value`

  This option sets a value (percent) on the junction that defines the hard limit for consumption of worker threads. As in the global hard limit setting, this option causes warning messages to be displayed when the junction tries to use more worker threads than allowed by the setting. In addition, the user is sent a **503 Service Unavailable** message.

This example must be entered in one line.

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

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

Allocation view of worker threads for junctions

Use the allocation view when you want to determine the location of a junction that is absorbing more than its share of worker thread resources.

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

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

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

You must specify both soft and hard limit values, both –l and –L options, on a specific junction.

HTTP data compression

The WebSEAL servers can be configured to compress data that is transferred over HTTP between the WebSEAL server and the client.

WebSEAL uses the **gzip** compression algorithm that is described in RFC 1952. **Gzip** is supported by all major browsers.

HTTP compression in WebSEAL can be configured based on the following types.

- MIME-type
- browser type
- protected object policies or POPs

Compression based on MIME-type

You can create an entry in the WebSEAL configuration file for each MIME-type or group of MIME-types for which data compression is needed.

```
[compress-mime-types]
mime_type = minimum_doc_size[:compression_level]
```

The *mime-type* can specify one particular MIME type or can use wildcard characters to specify a class of MIME types. Each mime-type declaration is a separate entry in the **[compress-mime-types]** stanza. The wildcard character (*) is limited to entries of one collection of MIME types. For example, `text/*`. Any MIME-type not listed in the stanza is not compressed. Order is important. The first entry that matches a returned document is used for that document.

The *minimum_doc_size* value specifies the policy on the size of documents that are compressed. This value is an integer. Valid values are shown in the following list.

- **-1**

  When the minimum size is **-1**, documents of the specified MIME-type are never compressed.
- 0

  When the minimum size is 0, documents of the specified MIME-type are always compressed.
- Integer greater than zero

  When the minimum size is greater than zero, documents of the specified MIME-type are compressed when the number of bytes in the response to WebSEAL exceeds this integer value.

Any negative number other than **-1** generates an error message.

When WebSEAL receives a request from a browser, the server examines the content-length field in the HTTP header to determine the size of the incoming data. However, not all HTTP responses contain the content-length field. When the content-length field is not found, WebSEAL compresses the document unless the applicable MIME-type is configured to never be compressed. For example, *minimum_doc_size* of **-1**.

The *compression_level* is an optional setting that specifies the data compression level. Valid values are integers 1 - 9. The larger the integer, the greater the amount of compression that takes place. The greater the amount of compression, the greater the load that is placed on the CPU. The value of increased compression must be weighed against any performance impacts. When the *compression_level* is not specified, a default level of 1 is used.

The following example compresses all documents of a size greater than 1000 bytes:

```
[compress-mime-type]
*/* = 1000
```

The following set of entries:

- disables compression for all images
- disables compression for CSS files
- enables compression at level 5 for all PDF documents
- enables compression for HTML documents of size greater than 2000 bytes
- enables compression for all other text documents, regardless of size

```
,
[compress-mime-type]

image/* = -1
text/css = -1
application/pdf = 0:5
text/html = 2000
text/* = 0
```

Compression based on user agent type

WebSEAL returns compressed data to user agents that request them. A user agent is a client that initiates a request. Use the WebSEAL configuration file to explicitly enable or disable compression for various browsers.

Examples of user agents include browsers, editors, spiders (web-traversing robots), or other tools. WebSEAL does not return compressed data to user agents that do not request it. However, some user agents request compressed data but do not know how to handle the data properly.

The following entry is the syntax.

```
[compress-user-agents]
user_agent_pattern = {yes|no}
```

The *user_agent_pattern* consists of wildcard patterns that match characters that are found in the user-agent header sent to WebSEAL. The value `yes` means to compress data that is returned to the browser. The value `no` means to return the data uncompressed.

When the user-agent header does not match any of the stanza entries in the WebSEAL configuration file, WebSEAL allows the accept-encoding header that is sent by the browser.

For example, the following entry enables compression for Internet Explorer 6, but disables compression for all other browsers:

```
[compress-user-agents]
*MSIE 6.0" = yes
* = no
```

Compression policy in POPs

Attach a protected object policy or POP to an object so that WebSEAL disables compression of that object.

You can specify a compression policy of *do not compress* in a POP. To specify this policy, add the following attribute to the POP:

```
document-compression = no
```

A POP without this attribute set, or with this attribute set to *any value* other than no, allows documents to be compressed.

The following example shows how to disable compression for the junction /appOne.

```
pdadmin> pop create appOnePop
pdadmin> pop modify appOnePop set attribute document-compression no
pdadmin> pop attach /WebSEAL/host/appOne appOnePop
```

To allow compression for a subdirectory beneath /appOne with an overriding POP, attach a different POP that does not have the document-compression attribute. For example:

```
pdadmin> pop create dataPop
pdadmin> pop attach /WebSEAL/host/appOne/data dataPop
```

This method of applying compression policy can be used with URLs. For example, to disable compression that is based on wildcard patterns that are applied to URLs, you can use dynurl. To disable compression for all requests to a junction that have a particular argument in the query string, you can create a dynurl.conf file with the following entries.

```
/disableCompression     /appOne/*\?want-response=text/xml
```

You can then attach a POP to /WebSEAL/host/disableCompression with the document-compression attribute set to no.

Data compression limitation

The transfer of compressed data between WebSEAL and back-end servers is not supported.

WebSEAL filters URLs for various purposes. However, WebSEAL does not filter compressed data.

Configuring data compression policy

Edit the WebSEAL configuration file so that you can specify data compression policy for communication between WebSEAL and client browsers.

Procedure

1. Log in to the local management interface.
2. Select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
3. Select your instance.
4. Select **Manage** > **Configuration** > **Edit Configuration File**.
5. Specify each MIME-type for which a data compression policy applies. Assign a value that enforces the policy.

   ```
   [compress-mime-types] mime_type = minimum_doc_size
   ```

   The default setting leaves all data uncompressed:

```
[compress-mime-types]
*/* = -1
```

For more information, see “Compression based on MIME-type” on page 45.

6. Specify each type of user agent (browser) for which a data compression policy applies. Enable data compression by assigning the value `yes`. Disable data compression by assigning the value `no`.

   ```
   [compress-user-agents]
   user_agent = {yes|no}
   ```

   No entries are set by default. When no entry matches the user-agent's accept-encoding header, the value in the accept-encoding header is allowed. For more information, see the **[compress-user-agents]** stanza in the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.
7. Optional: Specify compression policies in POPs, and apply the POPs to the appropriate objects in the protected object space. For more information, see “Compression policy in POPs” on page 47.

WebSEAL data handling by using UTF-8

WebSEAL implements multi-locale support by internally maintaining and handling all data by using UCS Transformation Format 8 byte (UTF-8) encoding. UTF-8 is a multi-byte code page with variable width.

WebSEAL adopts UTF-8 as the default code page for all internal data handling. This support enables WebSEAL to process data from multiple languages at the same time.

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

WebSEAL handles data internally in UTF-8 regardless of the locale in which the WebSEAL process is running. When locale-specific data is needed as input or output, the locale in which the WebSEAL process is running becomes important.

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

The appliance utilizes a UTF-8 code page. If WebSEAL needs to be run using a different code page, the LC_ALL environment variable, located in the **[system-environment-variables]** configuration stanza, should be set to the desired locale. For example:

```
[system-environment-variables]
LC_ALL = ja_JP.eucjp
```

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

For example, you might need to change default settings for systems that process client requests and forms data that uses non-UTF-8 local code pages. For example:

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

If you are running systems that need to provide true multi-locale support to handle users and data in multiple languages, review the following settings:

- The default WebSEAL multi-locale UTF-8 settings.

You can customize these configuration settings to best fit your deployment.

UTF-8 dependency on user registry configuration

For optimal multi-locale support, store all the users in one common user registry, regardless of which language they prefer.

Most user registries support UTF-8 by default. Some LDAP user registries, and their supporting databases, can optionally be configured to not support UTF-8. Ensure that the LDAP user registry and database that is used with Security Access Manager uses UTF-8.

IBM Tivoli Directory Server is by default that is configured to use UTF-8.

UTF-8 data conversion issues

By default, the appliance will use a UTF-8 code page when running WebSEAL. However, it is possible to configure WebSEAL so that it uses a non-UTF-8 code page. In this environment, WebSEAL needs to convert data upon data input and output.

When WebSEAL reads data in, it must convert the data from non-UTF-8 to UTF-8. When WebSEAL writes out data, it must convert the data from UTF-8 to non-UTF-8.

If conversion to a local code page is required, no data loss occurs when it runs in a UTF-8 locale.

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

WebSEAL generates logging and auditing data by using UTF-8. To prevent possible data loss, use UTF-8 to write the data to the appropriate logging and auditing files. When the local code page is non-UTF-8, data must be converted to non-UTF-8 before it can be written. In this case, the possibility of data loss exists.

All log audit files that are generated by WebSEAL are in the language that is specified by the locale in which the server runs. The code page that is used to write the messages is configurable in the WebSEAL routing file.

UTF-8 impact on authentication

The use of UTF-8 for internal data handling has impacts on the processing of authentication requests by WebSEAL.

The following list describes the impact of the usage of UTF-8 for internal data handling.

- UTF-8 logins over basic authentication are not supported.

  Use of UTF-8 with basic authentication login is not supported. UTF-8 logins with basic authentication cannot be supported because browsers transmit data in inconsistent ways. WebSEAL does not support multi-byte basic authentication logins because of browser inconsistency.

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

  In previous versions of WebSEAL, forms login data was always used by WebSEAL with the `auto` function. WebSEAL examined the login data to see whether it was in UTF-8 format. If the data was not in UTF-8 format, the data was processed as local code page.

  For WebSEAL version 5.1 and greater, this setting is configurable as described in "UTF-8 support in POST body information (forms)" on page 52.
- Cross-domain single signon, e-community single signon, and failover authentication

  Each of these authentication methods employs encoded tokens. The encoding of these tokens must be configured to use either UTF-8 encoding or non-UTF-8 encoding.

UTF-8 impact on authorization (dynamic URL)

WebSEAL restricts all requests that require authorization checks to requests that use UTF-8 or the locale setting of the WebSEAL host. All back-end servers are also bound by these settings. WebSEAL must enforce this restriction so it can apply security policy on known protected objects.

This restriction is important to consider when you enable the WebSEAL dynamic URL feature. WebSEAL dynamic URL processes data from POST bodies and from query strings. Data from both POST bodies and query strings need to be in a character encoding that is known to WebSEAL for successful mapping of character patterns to authorization objects.

By design, WebSEAL dynamic URL processes the query string portion of a request, where the dynamic data destined for the web application interface is located. The GET request standard uses this query string format. To support the query string requirement for dynamic URL, WebSEAL converts any data that is contained in the body of a POST request into the query string format.

When dynamic URL is enabled, WebSEAL maps the data from query strings to objects that require protection or access control. To securely map query strings to objects, the string data needs to use the same character set known to WebSEAL and the back-end application server. Otherwise, dynamic URL access control might be circumvented by a request that uses a character that is accepted by the back-end application, but not accepted by WebSEAL. If WebSEAL receives dynamic data (in a POST body or query string) by using characters that are not UTF-8 or from the character set in which WebSEAL runs, WebSEAL rejects the request and returns an error.

If WebSEAL (with dynamic URL enabled) is running in a non-UTF-8 environment, and request POST bodies or query strings contain UTF-8 characters, you can configure the **utf8-form-support-enabled** stanza entry in the **[server]** stanza of the WebSEAL configuration file to allow WebSEAL to decode the UTF-8 coding in these requests.

For more information, see "Dynamic URLs" on page 600.

Encoding type usage

WebSEAL requires that any URL presented for processing must contain only a single character encoding type such as UTF-8 or ShiftJIS.

When URLs contain multiple character encoding types, WebSEAL cannot guarantee the accuracy of the data in the request because the decoded value of the UTF-8 characters might not match the decoded value of the same characters in the local code page. This possible inaccuracy in the data might cause WebSEAL to mistakenly grant unauthorized users access to protected objects.

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

UTF-8 support for uniform resource locators

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

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

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

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

The three possible values are as follows:

- **yes**

  In this mode, WebSEAL recognizes only URI encoded UTF-8 data in URL strings and they are used without modification. These UTF-8 characters are then validated and taken into account when it determines access rights to the URL. WebSEAL supports both raw UTF-8 and URI encoded UTF-8 strings in URLs. In this mode, other encoding techniques are not accepted.

  This value is the default and is appropriate for most environments.

  Servers that run in a 7-bit ASCII English locale must use this value.
- **no**

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

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

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

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

- **auto**

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

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

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

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

The following list is a sample deployment strategy.

1. Unless required for content purposes, immediately check and set the **default-webseal** ACL on existing production deployments to NOT allow unauthenticated `r` access. This setting limits security exposure to users who have a valid account in the Security Access Manager domain.
2. Ensure that the **utf8-url-support-enabled** stanza entry is set to the default value of `yes`.
3. Test your applications. If they function correctly, use this setting.
4. If any applications fail with **Bad Request** errors, try the application with the **utf8-url-support-enabled** stanza entry set to `no`. If this step works, you can deploy with this setting. Ensure, however, that no junctioned web server is configured to accept UTF-8 encoded URLs.
5. If the application continues to have problems, try setting **utf8-url-support-enabled** to `auto`.

UTF-8 support in POST body information (forms)

Edit WebSEAL configuration file so that you can configure how WebSEAL processes data in POST bodies that contain information from forms.

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

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

The three possible values are as follows:

- **yes**

  WebSEAL recognizes only UTF-8 encoding in forms and the data is used without modification. These UTF-8 characters are then validated and taken into account when it processes the data. Other encoding techniques are not accepted.

  When **double-byte-encoding** is set to `yes`, Unicode of the form `%HH%HH` is supported. When a double-byte Unicode character is detected, the entire string must be double-byte encoded.

  This value is the default value and appropriate for most environments.
- **no**

  WebSEAL does not recognize UTF-8 encoding in forms. Used for local code page only. If the form data can be validated, it is converted to UTF-8 for internal use.
- **auto**

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

UTF-8 support in query strings

You can enable UTF-8 support in query strings by editing the WebSEAL configuration file.

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

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

The three possible values are as follows:

- **yes**

  WebSEAL recognizes only UTF-8 encoding in query strings and the data is used without modification. These UTF-8 characters are then validated and taken into account when it processes the data. Other encoding techniques are not accepted.

  Use this setting when your WebSEAL server must process query strings that use UTF-8.

  Servers that operate in a single-byte Latin locale, such as French, German, or Spanish, and process queries from an application that uses UTF-8, must use this setting. Servers that operate in a multi-byte locale and process only UTF-8 query strings can use this setting.
- **no**

  WebSEAL does not recognize UTF-8 encoding in query strings. Used for local code page only. If the form data can be validated, it is converted to UTF-8 for internal use.

  This setting is the default value, appropriate for most environments.

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

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

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

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

UTF-8 encoding of tokens for cross domain single signon

The use of UTF-8 encoding for strings within tokens that are used for cross domain single signon is specified in the WebSEAL configuration file.

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

The default value is `true`.

When **use-utf8** is set to `false`, strings are encoded by using the local code page. Use this value when you implement cross domain single signon with versions of WebSEAL before version 5.1. WebSEAL versions before 5.1 do not use UTF-8 encoding for tokens. When you deploy an environment that includes these older servers, configure WebSEAL to not use UTF-8 encoding. This setting is necessary for compatibility with an earlier version.

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

UTF-8 encoding of tokens for e-community single signon

The use of UTF-8 encoding for strings within tokens that are used for e-community single signon is specified in the WebSEAL configuration file.

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

The default value is `yes`.

When **use-utf8** is set to `no`, strings are encoded by using the local code page. Use this value when you implement an e-community single signon with versions of WebSEAL before version 5.1. WebSEAL versions before 5.1 do not use UTF-8 encoding for tokens. When you deploy an environment that includes these older servers, configure the WebSEAL server to not use UTF-8 encoding. This setting is necessary for compatibility with an earlier version.

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

UTF-8 encoding of cookies for failover authentication

You can specify the use of UTF-8 encoding for strings within failover authentication cookies in the WebSEAL configuration file.

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

The default value is `yes`.

When **use-utf8** is set to `no`, failover authentication cookies are encoded by using the local code page. Use this value when you implement failover authentication with versions of WebSEAL before version 5.1. WebSEAL versions before 5.1 do not use UTF-8 encoding for failover authentication cookies. When you deploy an environment that includes these older servers, configure the WebSEAL server to not use UTF-8 encoding. This setting is necessary for compatibility with an earlier version.

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

UTF-8 encoding of cookies for LTPA authentication

WebSEAL supports LTPA version 2 cookies only for LTPA authentication. The specification for this version of LTPA cookies requires the use of UTF-8 encoding.

Due to this requirement, there is no option to enable or disable UTF-8 encoding for LTPA cookies. LTPA cookies are always UTF-8 encoded.

UTF-8 encoding in junction requests

By default, WebSEAL adds information to HTTP headers by using a UTF-8 code page. This action prevents any potential data loss that can occur when it converts to a non-UTF-8 code page. This data is sent URI encoded. For compatibility with an earlier version, the format of the header data can be configured to the local code page. In addition, two other formats are supported, raw UTF-8 and URI encoded local code page.

WebSEAL inserts information into HTTP headers for requests to the back-end server. This information can include extended attributes or user data. In WebSEAL versions before version 5.1, the headers were added to the request by using the local code page. In WebSEAL version 5.1 and greater, the header data is transmitted in a configurable format.

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

Argument	Description
utf8_uri	URI encoded UTF-8 data. All white space and non-ASCII bytes are encoded %XY, where X and Y are hex values (0–F). Encoding applies also to the following entries: • ASCII characters below 0x1F and above 0x7F • Escape characters for tab, carriage return, and line feed • Percent symbol
utf8_bin	Unencoded UTF-8 data. This setting allows data to be transmitted without data loss, and the user does not need to URI-decode the data. This setting must be used with caution because it is not part of the HTTP specification.

Argument	Description
lcp_uri	URI encoded local code page data. Any UTF-8 characters that cannot be converted to a local code page are converted to question marks (?). Use this option with caution and only in environments where the local code page produces the wanted strings.
lcp_bin	Unencoded local code page data. This mode was used by versions of WebSEAL before version 5.1. Use of this mode enables migration from previous versions, and is used in upgrade environments. Data loss can potentially occur with this mode. Use with caution.

Validation of character encoding in request data

WebSEAL parses requests to ensure that character encoding is compatible with the back-end server requirements. For example, it is possible for the query string of a request to contain character encoding, such as raw binary data, that is unacceptable to WebSEAL, and therefore rejected by WebSEAL.

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

One solution to the problem of incorrect validation of character encoding is to configure WebSEAL to not validate the query string and POST body data of requests. The request data can then be passed unchanged to the back-end application.

To instruct WebSEAL to not validate query string and POST body data, set the value of the **decode-query** stanza entry in the **[server]** stanza of the WebSEAL configuration file to "no":

```
[server]
decode-query = no
```

The following setting is the default.

```
decode-query = yes
```

If `decode-query` is set to `yes`, WebSEAL validates the query string in requests according to the **utf8-qstring-support-enabled** stanza entry. See "UTF-8 support in query strings" on page 53. This setting applies to POST body data in requests when dynamic URL is enabled. Dynamic URL converts the POST body data in a request to query string format. See "Conversion of POST body dynamic data to query string format" on page 602.

If `decode-query` is set to `yes`, WebSEAL validates the POST body in requests according to the **utf8-form-support-enabled** stanza entry. See "UTF-8 support in POST body information (forms)" on page 52.

If you set decode-query=no, you must understand the possible consequences to securing protected objects. In particular, if WebSEAL is configured to not validate query strings in requests (decode-query=no), then dynamic URL mapping for authorization checking, if enabled, must be disabled.

To disable the dynamic URL feature, comment out the **dynurl-map** stanza entry in the **[server]** stanza of the WebSEAL configuration file:

```
[server]
#dynurl-map = bin/dynurl.conf
```

Supported wildcard pattern matching characters

WebSEAL supports wildcard pattern matching characters.

Table 3. Supported wildcard matching characters

Character	Description
\	The character that follows the backslash is part of a special sequence. For example, \t is the TAB character. Can be used to escape the other pattern matching characters: (? * [ ] ^). To match the backslash character, use \\.
?	Wildcard that matches a single character. For example, the string abcde is matched by the expression ab?de
*	Wildcard that matches zero or more characters.
[]	Defines a set of characters, from which any can match. For example, the string abcde is matched with the regular expression ab[cty]de.
^	Indicates a negation. For example, the expression [^ab] matches anything but the a or b characters.

For more examples of pattern matching by using wildcards, see the following topics.

- "Extended attributes to add to token" on page 557
- "Extended attributes to add to token" on page 578
- "Mapping ACL and POP objects to dynamic URLs" on page 602

Setting system environment variables

Use the **system-environment-variables** stanza to list the system environment variables that the WebSEAL daemon exports during initialization. Include a separate entry for each system environment variable that you want to export.

Note: The environment variable names are case-sensitive.

The following line is the format of each configuration entry.

```
<env-name> = <env-value>
```

<env-name>
The name of the system environment variable.

<env-value>
The value of the system environment variable.

For example:

```
[system-environment-variables]
LANG = de
```

Web server response configuration

This chapter discusses the resources available to the WebSEAL server for responding to client requests.

Topic Index:

Static server response pages

WebSEAL provides a number of static server response pages that can be used to provide responses to client requests.

These pages include:

- Error messages
- Informational messages
- Login forms
- Password management forms

You can modify the contents of these pages to include site-specific messages or perform site-specific actions. Most pages are appropriate for forms and basic authentication over HTTP or HTTPS.

The names, content, and descriptions of the message pages are listed in the following table. The following codes are used to indicate the message type:

- **ER** - error message
- **IN** - informational message
- **LG** - login form
- **PW** - password management form
- **NA** - not applicable

Filename	Status and HTTP Code	Description	Type
132120c8.html	Authentication Failed (HTTP 403)	Credentials cannot be retrieved for the client certificate used. Possible reasons include: • The user supplied an incorrect certificate. • The user's credentials are missing from the authentication database.	ER
38ad52fa.html	Non-empty Directory (HTTP 500)	The requested operation requires the removal of a non-empty directory. The requested operation is an illegal operation.	ER
38b9a4b0.html	Application Server is Offline (HTTP 503)	The application server you are accessing has been taken offline by the system administrator. Returned when a request is blocked due to a junction that has been placed in a throttled or offline operational state.	ER

Filename	Status and HTTP Code	Description	Type
38b9a4b1.html	Service Unavailable (HTTP 503)	The WebSEAL server is unable to service a request because a needed resource is unavailable.	ER
38b9a41f.html	Additional Login Denied (HTTP 200)	You have already logged in to this Web server from another client. No more new logins are permitted until your initial session has ended.	ER
38cf013d.html	Request Caching Failed (HTTP 500)	The **request-max-cache** or **request-body-max-read** values have been exceeded.	ER
38cf0259.html	Could Not Sign User On (HTTP 500)	The resource requested requires the WebSEAL server to sign the user on to another Web server. However, a problem occurred while WebSEAL was attempting to retrieve the information.	ER
38cf025a.html	User Has No Single Signon Information (HTTP 500)	WebSEAL could not locate the GSO user for the requested resource.	ER
38cf025b.html	No Single Signon Target for User (HTTP 500)	WebSEAL could not locate the GSO target for the requested resource.	ER
38cf025c.html	Multiple Signon Targets for User (HTTP 500)	Multiple GSO targets are defined for the requested resource. This is an incorrect configuration.	ER
38cf025d.html	Login Required (HTTP 500)	The resource requested is protected by a junctioned back-end Web server, requiring WebSEAL to sign the user on to that Web server. In order to do this, user must first log in to WebSEAL.	ER
38cf025e.html	Could Not Sign User On (HTTP 500)	The resource requested requires WebSEAL to sign the user on to another Web server. However, the signon information for the user account is incorrect.	ER
38cf025f.html	Unexpected Authentication Challenge (HTTP 500)	WebSEAL received an unexpected authentication challenge from a junctioned back-end Web server.	ER
38cf0421.html	Moved Temporarily (HTTP 302)	The requested resource has been temporarily moved. This event usually occurs if there has been a mishandled redirect.	ER
38cf0424.html	Bad Request (HTTP 400)	WebSEAL received an HTTP request that is not valid.	ER

Filename	Status and HTTP Code	Description	Type
38cf0425.html	Login Required (HTTP 401)	The resource you have requested is secured by WebSEAL, and in order to access it, you must first log in.	ER
38cf0427.html	Forbidden (HTTP 403)	The user does not have permissions to access the requested resource.	ER
38cf0428.html	Not Found (HTTP 404)	The requested resource cannot be located.	ER
38cf0432.html	Service Unavailable (HTTP 503)	A service required by WebSEAL to complete the request is currently not available.	ER
38cf0434.html	Privacy required (HTTP 403)	Quality of protection at the privacy level is required.	ER
38cf0437.html	Server Suspended (HTTP 500)	The WebSEAL server has been temporarily suspended by the System Administrator. No requests will be handled until the server is returned to service by the administrator.	ER
38cf0439.html	Session Information Lost (HTTP 500)	The browser and server interaction was a stateful session with a junctioned back-end server that is no longer responding. WebSEAL requires a service located on this server to complete your request.	ER
38cf0442.html	Service Unavailable (HTTP 503)	The service required by WebSEAL is located on a junctioned back-end server where SSL mutual authentication has failed.	ER
38cf04c6.html	Third-party server not responding (HTTP 500)	The requested resource is located on a third-party server. WebSEAL has tried to contact that server, but the server is not responding.	ER
38cf04d7.html	Third-party server not responding (HTTP 500)	The requested resource is located on a third-party server. WebSEAL has tried to contact that server, but the server is not responding.	ER
38cf07aa.html	CGI Program Failed (HTTP 500)	A CGI program failed to execute properly.	ER
38cf08cc.html	Access Denied (HTTP 403)	The resource you have requested is protected by a policy that restricts access to specific time periods. The current time is outside of those permitted time periods.	ER

Filename	Status and HTTP Code	Description	Type
`acct_locked.html`	Account locked (HTTP 200)	Page displayed in these circumstances: • Authentication failed because the user's account was temporarily locked due after too many unsuccessful login attempts. • `nsAccountLock` is `true` for a user (in Sun Directory Server) when they attempt to login. This is only displayed if the user provides the correct password during login. **Note:** When using basic authentication (BA-auth) with Security Access Manager, the acct_locked.html file cannot be customized to contain additional images. Although you can embed images in the file, subsequent requests to access the embedded images will fail.	ER
`certfailure.html`	Certification authentication failed (HTTP 200)	An attempt to authenticate with a client certificate failed. Page displayed if client fails to authenticate with a certificate when **accept-client-certs = required**. A valid client certificate is required to make this connection.	ER
`certlogin.html`	Access Manager Login (HTTP 200)	Certificate login form used when **accept-client-certs = prompt_as_needed**.	LG
`certstepuphttp.html`	Attempt to Step-up to Certification authentication failed (HTTP 200)	An attempt to step-up to certificates over HTTP failed. Use of HTTPS is required. Try re-accessing the page over HTTPS.	ER
`default.html`	Server Error (HTTP 500)	WebSEAL could not complete your request due to an unexpected error.	ER
`deletesuccess.html`	Success (HTTP 200)	The client-initiated DELETE request completed successfully.	IN
`help.html`	PKMS Administration (HTTP 200)	Help information for **pkmslogout** and **pkmspasswd**.	IN
`login.html`	Access Manager Login (HTTP 200)	Standard request form for user name and password	LG
`login_success.html`	Success (HTTP 200)	Normally, WebSEAL caches the URL of the requested resource, and returns the resource to the user upon successful login. The `login_success.html` page is displayed after successful login if for some reason WebSEAL cannot determine the URL of the originally requested resource.	IN
`logout.html`	PKMS Administration: User Log Out (HTTP 200)	Page displayed after successful logout. User USERNAME has logged out.	IN

Filename	Status and HTTP Code	Description	Type
passwd.html	PKMS Administration: Change Password (HTTP 200)	Change password form. Also displayed if password change request failed.	PW
passwd_exp.html	PKMS Administration: Expired Password (HTTP 200)	Page displayed if user authentication failed due to an expired password. Change expired password.	PW
passwd_rep.html	PKMS Administration: Change Password (HTTP 200)	Page displayed if password change request was successful.	PW
passwd_warn.html	Password Administration The password will expire at approximately... (HTTP 200)	Page displayed if user LDAP password is due to expire soon.	PW
putsuccess.html	Success (HTTP 200)	The client-initiated PUT operation completed successfully.	IN
query_contents.html	Junctioning Win32 Web Servers (HTTP 200)	Installation and configuration information for making a junction from an Access Manager WebSEAL server to third-party Web servers running on the Win32 platform.	IN
relocated.html	Temporarily Moved (HTTP 302)	The requested resource has temporarily moved.	IN
stepuplogin.html	Access Manager Step Up Login (HTTP 200)	Login form for step-up authentication.	LG
switchuser.html	Access Manager Switch User (HTTP 200)	Login form for switch user.	LG
template.html	Template	Template form for custom error messages.	NA
too_many_sessions.html	PKMS Administration: Session Displacement (HTTP xxx)	Error message when exceeding the limit of concurrent logins by a single user.	ER
websealerror.html	WebSEAL Server Error (HTTP 400)	WebSEAL server internal error.	ER

Server response page locations

For storage location purposes on the WebSEAL server, static server response pages are grouped into three general categories:

- Account management pages

- Error message pages
- Junction-specific static server response pages

The location of these three storage categories is configurable, as described in the following sections:

Management Root

On the appliance, you can use the local management interface to access the Management Root file structure.

The file structure is available on the Manage Reverse Proxy Management Root page for each instance. To open this Management Root page and access the file structure from the LMI web interface, you must:

1. Select **Secure Web Settings** > **Manage** > **Reverse Proxy** from the top menu. The Reverse Proxy management page displays.
2. Click the instance that you want to manage from the available list of Instance Names.
3. Select **Manage** > **Management root**. The Manage Reverse Proxy Management Root page for the selected instance displays with the available file structure.

The available directories include:

- `management`
- `errors`
- `oauth`
- `junction-root`

Account management page location

Account management pages include login forms, password management forms, and some informational messages.

You can use the LMI to access the account management pages. Open the `management` directory on the Manage Reverse Proxy Management Root page. Pages are located in language-specific subdirectories at this location. The subdirectory specific to your locale is automatically appended to the end of the directory hierarchy during WebSEAL installation and configuration.

The default United States English directory is:

```
management/C
```

The Japanese locale directory is:

```
management/JP
```

See “Multi-locale support for server responses” on page 75 for further information on multi-locale support.

Error message page location

Pages are in language-specific subdirectories of the `errors` directory. You can access these directories from the Manage Reverse Proxy Management Root page. The subdirectory specific to your locale is automatically appended to the end of the directory hierarchy during WebSEAL installation and configuration.

The default United States English directory is:

```
errors/C
```

The Japanese locale directory is:

```
errors/JP
```

See “Multi-locale support for server responses” on page 75 for further information on multi-locale support.

For information on creating junction-specific static server response pages, see “Junction-specific static server response pages.”

Junction-specific static server response pages

You can customize static server response pages on a per-junction basis.

Add the customized static server response page files into a junction-specific directory:

```
errors/language/junction_id
```

where *junction_id* refers to the junction point for a standard junction (excluding the leading / character) or the virtual host label for a virtual host junction. For example:

```
errors/C/test_junction
```

WebSEAL searches for static server response page files in the following sequence, returning the first file found to the client:

1. `errors/`*`language`*`/`*`junction_id`*`/`*`page`*`.<content-type>`
2. `errors/`*`language`*`/`*`junction_id`*`/default.<content-type>`
3. `errors/`*`language`*`/`*`page`*`.<content-type>`
4. `errors/`*`language`*`/default.<content-type>`

You can use the / character in the junction name. For example, if you created a junction directory named `test` under the `jct` directory, the junction is specified as `/jct/test`. In this instance WebSEAL searches for files in the `errors/`*`language`*`/jct/test` directory.

Content Aware Server Responses

WebSEAL supports the ability to use different response template pages for different content types. It also provides the ability to specify custom HTTP response codes for different generated responses.

The content type and response code for a template file is determined by the name of the file itself. The MIME sub-type is defined by the file extension and the custom response code (if required) is embedded within the file name. The format of the file name is as follows:

```
<file identifier>{.<response_code>}.<mime sub-type>
```

The following table includes example file names:

Table 4. File examples

Filename	Description
`login.html`	This file is returned for requests with the 'HTML' MIME sub-type. For example, `text/html`. The default login response code is used.

Table 4. File examples (continued)

Filename	Description
`login.json`	This file is returned for requests with the 'json' MIME sub-type. For example, `application/json`. The default login response code is used.
`login.401.json`	This file is returned for requests with the 'json' MIME sub-type. For example, `application/json`. The 401 HTTP response code is returned.

WebSEAL determines the correct file to use when generating a response by one of the following scenarios:

- Locating the MIME sub-type in the 'accept' HTTP header of the request. It uses the first MIME entry that is found in the 'accept' header disregarding any specified quality factor and any subsequent MIME types;
- If the 'accept' HTTP header is not present it locates the MIME sub-type from the 'content-type' HTTP header in the request;
- If both headers are missing from the request it uses the content-type as defined by the 'default-response-type' configuration entry within the **`[acnt-mgt]`** stanza of the configuration file. See [acnt-mgt] stanza.

Once the correct content type is determined, WebSEAL searches for a matching file in the appropriate directory, as determined by the locale and junction. See "Junction-specific static server response pages" on page 64. If no matching file for the requested response type is located the appropriate directories are then searched for a file that matches the default content type, as defined by the 'default-response-type' configuration entry.

By default WebSEAL sends a `302` response when a redirect is required, along with the `38cf0421.html` error page. This error page can be customized with different response codes for different response types if a 302 response is not desired.

WebSEAL provides two out-of-the-box (OOTB) JSON response files: `management/C/default.json` and `errors/C/default.500.json`. If you want to revert to the legacy WebSEAL behaviour of returning HTML files for all responses, delete these two response files from the management root of your WebSEAL instance.

HTML server response page modification

This section contains the following topics:

Guidelines for customizing response pages

You can customize the static server response pages to better reflect your current WebSEAL implementation. Observe the following notes:

- Do not modify the name of the file. The hexadecimal number is used by WebSEAL to display the proper error file.
- Use an HTML or text editor to modify page contents. Ensure that you use valid HTML tagging for HTML response files.
- Specify server-relative URIs (rather than relative URIs) for any URIs for resources such as images or CSS. If virtual host junctions are being used for WebSEAL, you must use absolute URIs for such resources.

- WebSEAL supplies a set of macros that you can use to capture dynamic information. See “Macro resources for customizing response pages.”

Macro resources for customizing response pages

Macros are predefined, specially formatted strings that are used to dynamically add information to static server response pages.

When WebSEAL responds with a static page, it parses the page and searches for occurrences of macros. When a macro is found, the appropriate content is dynamically substituted. Macros are populated only when the value is relevant to that page.

For example, WebSEAL returns a static server response page in response to a request that results in an error. If WebSEAL encounters the ERROR macro in the static server response page, WebSEAL substitutes a string representation of the error code that was generated when handling the request.

The following macros occur in some of the static server response pages provided by WebSEAL, and are available for use in customizing these pages:

Macro	Description
`AUTHNLEVEL`	Substitutes the authentication level used in authentication strength policy (step-up).
`BACK_NAME`	Substitutes the value "BACK" if a referer header is present in the request, or "NONE" if no referer header is present in the request.
`BACK_URL`	Substitutes the value of the referer header from the request, or "/" if none.
`BASICAUTHN`	Used to control the display of information in the `certlogin` and `stepuplogin.html` login forms. When the authentication method (indicated by the macro name) is valid, the section in the form governed by the macro is displayed. When the authentication method is not valid, the macro is replaced by a start comment delimiter (<!--). All subsequent information in the form is commented out until a comment closing delimiter (-->) is reached. **Note:** This macro is only appropriate for HTML responses.
`CERTAUTHN`	Used to control the display of information in the `certlogin` and `stepuplogin` login forms. When the authentication method (indicated by the macro name) is valid, the section in the form governed by the macro is displayed. When the authentication method is not valid, the macro is replaced by a start comment delimiter (<!--). All subsequent information in the form is commented out until a comment closing delimiter (-->) is reached.
`CREDATTR{`*name*`}`	The value of the user credential attribute that has the specified *name*. For example, **`CREDATTR{tagvalue_session_index}`** returns the session token.
`EAIAUTHN`	Used to control the display of information in the `certlogin` and `stepuplogin` login forms. When the authentication method (indicated by the macro name) is valid, the section in the form governed by the macro is displayed. When the authentication method is not valid, the macro is replaced by a start comment delimiter (<!--). All subsequent information in the form is commented out until a comment closing delimiter (-->) is reached. **Note:** This macro is only appropriate for HTML responses.

Macro	Description
ERROR	The hard-coded error message returned from Security Access Manager. Same as ERROR_TEXT. Both macros exist for compatibility with prior versions of WebSEAL.
ERROR_CODE	The numeric value of the error code.
ERROR_TEXT	The text associated with an error code in the message catalog. Same as ERROR. Both macros exist for compatibility with prior versions of WebSEAL.
ERROR_URL	The URI of a Web page which provides additional information on the error. This macro is only used for OIDC error responses.
EXPIRE_SECS	Contains the numbers of seconds before the password expires. This can be included into the password warning form (passwd_warn) to display the time left the user has to change their password.
FAILREASON	Error message.
HOSTNAME	Fully qualified host name.
HTTP_BASE	Base HTTP URL of the server "http://*host*:*tcpport*/".
HTTPS_BASE	Base HTTPS URL of the server, "https://*host*:*sslport*/".
HTTPHDR{name}	Used to include the contents of a specified HTTP header. If the specified HTTP header does not exist within the request, the macro will contain the text: 'Unknown'. For example, the macro name to include the 'Host' HTTP header would be HTTPHDR{Host}.
LOCATION	Contains the URL to which the client is being redirected. Sent only in redirects.
METHOD	The HTTP method requested by the client.
OIDCAUTHN	Used to control the display of information in the login, certlogin and stepuplogin login forms. When the authentication method (indicated by the macro name) is valid, the section in the form governed by the macro is displayed. When the authentication method is not valid, the macro is replaced by a start comment delimiter (<!--). All subsequent information in the form is commented out until a comment closing delimiter (-->) is reached. **Note:** This macro is only appropriate for HTML responses.

Macro	Description
OLDSESSION	When WebSEAL receives a user request that contains an old ("stale") session cookie that no longer matches any existing entry in the WebSEAL session cache, the macro (normally set to "0") is set to the value of "1". The macro is set whenever WebSEAL sees a session cookie that is not recognized. Unrecognized session cookies can occur, for example, during session timeouts, session displacement, and when a user switches WebSEAL servers. Used in the standard WebSEAL login form to provide a trigger mechanism for a customized response to the user. This custom response could more accurately explain to the user why the session is not valid anymore. See “Customized responses for old session cookies” on page 315.
PROTOCOL	The client connection protocol used. Can be HTTP or HTTPS.
REFERER	The value of the HTTP referer header from the request, or "Unknown", if none.
REFERER_ENCODED	A URI encoded version of the HTTP referer header and macro.
STEPUP	A message specifying the step-up level required. Only sent when returning a step-up login form
TAM_OP	The operation code for the response. The values for this macro are identical to the values for local response redirects. See Operation for local response redirection.
URL	The URL requested by the client.
URL_ENCODED	A URI encoded version of the URI and macro.
USERNAME	The name of the user responsible for the request. (See also “Customization of login forms for reauthentication” on page 186.)

Macro data string format:

WebSEAL provides a configuration stanza entry that specifies the format of macro data strings that are inserted into server response pages. The default setting specifies UTF-8 format.

```
[content]
utf8-template-macros-enabled = yes
```

Static server response pages use a UTF-8 character set by default. If you modify the character set to specify the local code page, set this entry to “no”.

Note that this setting affects the pages that are generated by WebSEAL, as defined in the **[acnt-mgt]** stanza of the WebSEAL configuration file.

Macros embedded in a template

Security Access Manager verifies macro values before embedding the macros in templates, but additional changes might be necessary in some customer environments, depending on how the templates have been customized. Macros used in Security Access Manager templates are either URL macros or non-URL macros. URL macros are used to represent URLs, such as the HTTP request URL and the referer URL. Non-URL macros are used to represent other values, such as user names and error messages.

The following macros are URL macros:

- %LOCATION%
- %URL%
- %REFERER%
- %BACK_URL%
- %HOSTNAME%
- %HTTP_BASE%
- %HTTPS_BASE%
- %REFERER_ENCODED%
- %URL_ENCODED%

All other macros used in Security Access Manager are non-URL macros.

How Security Access Manager encodes macros:

Security Access Manager encodes macros before embedding the macros in a template.

Encoding ensures that macro values are interpreted as text and not JavaScript or HTML meta-characters. URL macros are encoded as Uniform Resource Identifier (URI) characters. For example, the left bracket (<) character in a URL is converted to **%3c** during encoding. Non-URL macros are encoded using HTML entities. For example, left (<) and right (>) bracket characters are encoded as **&lt;** and **&gt;**, respectively. Other HTML meta-characters in non-URL macros are encoded using numeric character references. For example, a double quotation mark (") is rendered as **&#34;**.

Security Access Manager encodes the following characters in both URL and non-URL macros:

Table 5. Characters encoded in URL and non-URL macros

Common Name	Character or Description
less-than symbol	<
greater-than symbol	>
colon	:
apostrophe	'
quotation mark	"
backslash	\
All values less than ASCII 0x20	Examples include escape, tab, carriage return, newline, formfeed, backspace, null byte.

In addition, Security Access Manager encodes the ampersand (&) character in non-URL macros.

Access Manger verifies that all URL macros except HOSTNAME are either absolute or server-relative URLs and use either HTTP or HTTPS protocol. The HOSTNAME macro must contain only alpha-numeric ASCII characters, dots, and hyphens.

Use of macros in an HTML template:

Caution must be exercised when embedding macros in HTML templates to avoid introducing cross-site scripting vulnerabilities to the Security Access Manager environment. Use the following guidelines when embedding macros:

- URL macros may be safely used as HTML text. To use a macro as HTML text, embed the macro between HTML tags. For example:
  ```
  <b>%URL%</b>
  ```
- URL macros may be safely used as HTML attribute values for HTML attributes, but only for attribute values that are intended for use with URLs. When using macros as HTML attribute values, the macro must be surrounded by double or single-quotes. For example:
  ```
  <a href="%URL%">clickable link</a>
  ```
- URL macros may be safely used as JavaScript string values, but must be surrounded by double or single-quotes. For example:
  ```
  var url = '%URL%';
  ```
- Non-URL macros may be safely used as HTML text. To use a macro as HTML text, embed the macro between HTML tags. For example:
  ```
  <b>%USERNAME%</b>
  ```
- Non-URL macros may be safely used as HTML attribute values, but only for attribute values that are NOT intended for use with URLs. When using macros as HTML attribute values, the macro must be surrounded by double or single-quotes. For example:
  ```
  <input type="text" name="user" value="%USERNAME%">
  ```
- Non-URL macros may be safely used as JavaScript string values, but must be surrounded by double or single-quotes. For example:
  ```
  var user = '%USERNAME%';
  ```

HTML tags and attributes:

Both URL and non-URL macros can be safely used with most basic HTML tags. However, there are certain HTML tags and attributes that use special syntax that is different from typical HTML tags. For example, the content of `<style>` tags is interpreted as cascading style sheet rather than typical HTML. Security Access Manager macros are encoded for use with typical HTML tags and attributes and should not be used within tags whose content is not interpreted as HTML.

In general, there is no reason to include Security Access Manager macros in HTML tags whose contents do not follow typical HTML syntax. If you are unsure of the syntax used by particular HTML tags, refer to W3C HTML specifications and your Web browser documentation for additional information.

Note also that non-URL macros such as %USERNAME% should not be used as HTML attribute values for attributes that are intended for URLs. Using non-URL macros as URLs can introduce cross-site scripting vulnerabilities in your deployed Security Access Manager environment.

Use of JavaScript to work with macros:

There are two methods of using JavaScript to work with Security Access Manager macros. You can use macros as JavaScript strings or you can use JavaScript to work with the HTML Document Object Model (DOM). To use a macro as a JavaScript string, simply insert the macro name between double or single quotes. For example:

```
var username = "%USERNAME%";
```

When using a macro as a JavaScript string, be aware that the macro value may contain URI encoding or HTML entity encoding. You can use the JavaScript `unescape()` function to remove URI encoding from macro values.

The recommended method for removing HTML entity encoding is to use JavaScript to work with the HTML DOM. For example, the following HTML code can be used to remove entity encoding from the %USERNAME% macro:

```
<span id='user' style='visibility: hidden'>%USERNAME%</span>
<script>
 var user = document.getElementById('user');
 if (user && user.firstChild)
  {
  var name = user.firstChild.nodeValue;
  }
</script>
```

The `name` variable contains the contents of the %USERNAME% macro; you can then use the variable as needed. However, use caution to avoid introducing DOM-based cross-site scripting vulnerabilities to the HTML template pages when using macro values.

Adding an image to a custom login form

About this task

When you customize a server response page, such as a login form, you can add images (graphics) to the page or form. Perform the following steps:

Procedure

1. Place the image file in an appropriate subdirectory under `junction-root`. You can use the LMI to manage this directory. Go to the Manage Reverse Proxy Management Root page. A suggested location for the image might be:

   ```
   junction-root/icons
   ```

   You can use HTML code similar to the following example to describe the image in the custom login form:

   ```
   <image src="/icons/logo.jpg" alt="Company Logo">
   ```

2. Ensure the definition of the image's file format is listed in the **`[content-mime-types]`** of the WebSEAL configuration file. For example:

   ```
   [content-mime-types]
   jpg = image/jpeg
   ```

3. Create an ACL that allows unauthenticated access to `logo.jpg`. Since this is the login page, there is no user ID established at the point of access. Therefore you must allow unauthenticated access to the image file object or directory object containing the image (such as the `icons` directory). The minimum permission required are "Tr" for both Unauthenticated and Any-other. For example:

   ```
   pdadmin> acl show icons-acl
   ACL name: icons-acl
   Description:
   Entries:
      Any-other Tr
      Unauthenticated Tr
      User sec_master TcmdbsvaBRrl
   ```

   Note: When setting the permissions on the Unauthenticated ACL entry, you must have as a minimum the same permissions as the Any-other ACL entry.

4. For this example, attach this ACL explicitly to the `icons` directory (or ensure that the unauthenticated permission is inherited to this point). For example:

```
pdadmin> acl attach /WebSEAL/abc.ibm.com-default/icons icons-acl
```

Account management page configuration

This section contains the following topics:

Configuration file stanza entries and values

The following response page stanza entries and values are located in the **[acnt-mgt]** stanza of the WebSEAL configuration file. Some pages are used only by the Forms login method of providing identity information.

Stanza Entry	HTML Response Page	Usage
login =	`login.html`	Forms login
login-success =	`login_success.html`	Forms login
logout =	`logout.html`	Forms login
account-inactivated =	`acct_locked.html`	Any method
account-locked =	`acct_locked.html`	Any method
passwd-expired =	`passwd_exp.html`	Any method
passwd-change =	`passwd.html`	Any method
passwd-change-success =	`passwd_rep.html`	Any method
passwd-change-failure =	`passwd.html`	Any method
passwd-warn =	`passwd_warn.html`	Any method
passwd-warn-failure =	`passwd_warn.html`	Any method
help =	`help.html`	Any method
certificate-login =	`certlogin.html`	Certificate login
cert-stepup-http =	`certstepuphttp.html`	Certificate login
stepup-login =	`stepuplogin.html`	Step-up authentication
switch-user =	`switchuser.html`	Any method
cert-failure =	`certfailure.html`	Certificate login
eai-auth-error =	`eaiautherror.html`	External authentication interface login error page
too-many-sessions =	`too_many_sessions.html`	Too many concurrent sessions error page
html-redirect =	`redirect.html`	HTML redirection

Configuration of the account expiration error message

WebSEAL returns an error message to a user when a login attempt fails. The message is conveyed through the ERROR macro contained in the appropriate account management page returned to the user. The generic error message ("Login failed") applies to a variety of situations where the user has supplied authentication information that is not valid, such as an incorrect user name or password.

You can use the **account-expiry-notification** stanza entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file to control whether additional information is revealed in the error message when the login failure is due to an expired account.

The default "no" setting allows only the generic error message ("Login failed") to be returned when the user login fails due to an expired user account:

```
[acnt-mgt]
account-expiry-notification = no
```

A "yes" setting for the **account-expiry-notification** stanza entry allows a more detailed error message to be returned when the user login fails due to an expired user account. This more detailed error message ("Account expired") indicates the exact reason for the failure (an expired account):

```
[acnt-mgt]
account-expiry-notification = yes
```

Note that the "Account expired" message implies that the correct user name is being used. This level of information might be considered a security exposure in some environments.

Configuration of the password policy options

The following WebSEAL options are available in the **[acnt-mgt]** stanza to use the password policy and account state for LDAP users.

```
[acnt-mgt]
enable-passwd-warn = yes
passwd-warn = passwd_warn.html
passwd-warn-failure = passwd_warn.html
account-inactivated = acct_locked.html
```

These options have no effect unless the corresponding Security Access Manager LDAP option is also enabled (`[ldap] enhanced-pwd-policy=yes`) and is supported for the particular LDAP registry type.

The **enable-passwd-warn** stanza entry enables WebSEAL to detect the attribute `REGISTRY_PASSWORD_EXPIRE_TIME` added to a user's credential when the LDAP password policy indicates that the user's password is soon to expire. The value of this new attribute is the number of seconds until the user's password expires. If this attribute is detected, a password warning form will be displayed when the user logs in to WebSEAL.

The page macro `EXPIRE_SECS` is available containing the number of seconds before the password expires. You can use this macro in the password warning form to display the time left for the user to change his password.

The **account-inactivated** stanza entry specifies a page to display if the value of `nsAccountLock` is `true` for a user in the Sun Directory when he attempts to log in. This page is only displayed if the user provides the correct password during login.

The **passwd-warn** stanza entry specifies the page to display after login if WebSEAL detects the LDAP password is soon to expire.

The **passwd-warn-failure** stanza entry specifies the page to display if the user fails to change his password that is due to expire. This page is often the same as the one specified by the **passwd-warn** stanza entry to give the users another chance to change their password.

Pages specified by the **passwd-warn** and **passwd-warn-failure** entries must provide a (hidden) field called `warn` when posting to the `/pkmspasswd.form`. Keep the value of the `warn` field short, as the value is ignored. The `/pkmspasswd.form` management URL detects this hidden field and proceeds to use the warning versions of the password change page. If the `warn` field is not detected then the non-warning forms are used instead.

```
<input type="HIDDEN" name="warn" value="*">
```

You can use the `/pkmsskip` WebSEAL Management URL to allow the **passwd-warn** page to skip changing the password and continue on with the login. This URL effectively redirects the users to the page that they were originally trying to access before being interrupted by the login process.

You can use the local response redirect options: **passwd_warn**, **passwd_warn_failure**, and **acct_inactivated**. See "Operation for local response redirection" on page 82 for more information.

Error message page configuration

When WebSEAL is unable to process a request from a client, WebSEAL returns an error message page to the client. The error message page explains why the request failed.

The error message pages are installed when the WebSEAL instance is configured. Each error message page is a separate static file. The names of the files are the hexadecimal values of the returned error codes. Do not modify these file names.

Note: You must specify server-relative URIs (rather than relative URIs) for any URIs for resources such as images or CSS. If virtual host junctions are being used for WebSEAL, you must use absolute URIs for such resources.

This section contains the following topics:

Enabling the time of day error page

About this task

The `38cf08cc` error message page is used when access is denied because a protected object policy (POP) time of day policy was not satisfied. WebSEAL controls the use of this error message page through a configuration file setting.

To enable WebSEAL to display `38cf08cc`, you must set the following entry in the WebSEAL configuration file:

```
[acnt-mgt]
client-notify-tod = yes
```

When `client-notify-tod = yes`, WebSEAL sends the client an error message stating that the authorization failure was due to a failed time-of-day POP access check.

This entry is set to "no" by default.

Note: A 403 error is always logged, regardless of the value assigned to **client-notify-tod**.

Creating new error message pages

About this task

You can create new error message pages for hexadecimal errors returned by WebSEAL. The hexadecimal errors returned by WebSEAL are documented in the IBM Security Access Manager Error messages section in the Knowledge Center.

For example, when WebSEAL encounters an invalid HTTP header, it returns the following error:

```
wand_s_jct_invalid_http_header           0x38cf04d5
```

To create a new error message page for this error, complete the following steps:

Procedure

1. Create a new file. To name the file, delete the 0x (hex) prefix characters from the error number and supply the response type suffix. For example, 0x38cf04d5 becomes:

   ```
   38cf04d5.html
   ```

   Optionally, you can use one of the existing HTTP error files as a template. Copy it and rename it.
2. Consult the Error messages section in the Knowledge Center for information about the exact error encountered. Use this information to edit the body of the page.
3. Optionally, you can use the macros described in "Macro resources for customizing response pages" on page 66.
4. Save the new file in the same directory as the rest of the HTTP error messages.

Results

If you want to create customized error message pages for a specific junction, you must place the customized files in a junction-specific location. For more information, see "Junction-specific static server response pages" on page 64.

Compatibility with previous versions of WebSEAL

WebSEAL version 5.1 introduced the following new error pages:

- `38cf04d7`
- `38cf04c6`

These messages provide information indicating that the encountered failure originated with a back-end server, not with WebSEAL.

In past releases, WebSEAL returned the default error page only. If you want to retain the previous behavior, remove the new error message pages from the error message page directory.

Multi-locale support for server responses

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

This section contains the following topics:

The accept-language HTTP header

WebSEAL supports multi-locale capabilities by using the values that are contained in the **Accept-Language** HTTP header to determine the correct language for server-generated messages and pages.

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

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

The order in which the values appear in the header determines the hierarchy of importance. WebSEAL checks the first listed value to see whether it is a supported language. If it is not a supported language, WebSEAL checks the next language in the list. If no supported languages are found, WebSEAL defaults to English.

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

Process flow for multi-locale support

The following example process flow illustrates how WebSEAL evaluates the **Accept-Language** header:

1. The **Accept-Language** header contains **pt-br** as the first value in the list.
2. The **pt-br** language is converted to `pt_BR`, representing the WebSEAL language subdirectory for this language.
3. If this subdirectory does not exist for the required message (for example, no language pack is installed for this language), WebSEAL checks for a `pt` directory.
4. If no `pt` directory exists, WebSEAL attempts to find message subdirectories for the next language listed in the header.
5. If there are no installed language packs for all languages listed in the header, WebSEAL defaults to the language environment that WebSEAL is running in, as determined by the LC_ALL or LANG environment variables set in the operating system's environment when WebSEAL is started.

Conditions affecting multi-locale support on WebSEAL

- Multi-locale support is enabled at all times on the WebSEAL server.
- Installation of specific language packs determines what languages are supported.
- If WebSEAL receives a message with no **Accept-Language** HTTP header, WebSEAL defaults to `C`.
- WebSEAL always returns the UTF-8 character set to the user, regardless of what the **Accept-Charset** HTTP header value requests.
- If WebSEAL accesses a locale directory for a translated message, and the directory is empty (for example, the contents were removed by the administrator), a server error page is returned.

Handling the favicon.ico file with Mozilla Firefox

About this task

Problem background:

The favicon.ico file is a small graphic icon that is used by some browsers (including Microsoft Internet Explorer and Mozilla Firefox) to enhance the display of address bar information and "favorites" bookmark lists. When requesting a resource, these browsers also try to locate the site's custom favicon.ico file.

There is a difference, however, in the way Internet Explorer and Mozilla Firefox decide when to request the favicon.ico file:

- Internet Explorer requests favicon.ico only when the returned page is bookmarked.
- Mozilla Firefox requests favicon.ico at the same time as the request for the page.

The request and response exchange between a Mozilla Firefox browser and a WebSEAL server can result in an HTTP 404 "Not found" message for the user when the favicon.ico does not exist.

In a protected WebSEAL environment, Mozilla Firefox's attempt to access the favicon.ico file triggers a login prompt. WebSEAL caches /favicon.ico as its "last requested URL." Once the user successfully logs in, WebSEAL redirects the request to this "last requested URL" location. The file (being non-existent in this example) is not found and a 404 "Not found" error is returned to the user. The originally requested page is never accessed because of the redirection process.

Solution:

The following steps solve this problem:

Procedure

1. Place a favicon.ico file in the **junction-root** directory. You can access this directory from the Manage Reverse Proxy Management Root page of the LMI.
2. Add a definition for the ico file format in the **[content-mime-types]** of the WebSEAL configuration file:
   ```
   [content-mime-types]
   ico = image/x-icon
   ```
3. Create an ACL that allows unauthenticated access to /favicon.ico. For example:
   ```
   pdadmin> acl show favicon
   ACL name: favicon
   Description:
   Entries:
      Any-other Tr
      Unauthenticated Tr
      User sec_master TcmdbsvaBRrl
   ```
4. Attach this ACL explicitly to /favicon.ico or ensure that the unauthenticated permission is inherited to this point. For example:
   ```
   pdadmin> acl attach /WebSEAL/abc.ibm.com-default/favicon.ico favicon
   ```

Results

If you do not want to create and install a favicon.ico file, the problem can be solved by following steps 3 and 4 only. An ACL can be attached to an object space representation of a resource, even if the resource does not physically exist. Although the browser still does not find the file, the unauthenticated ACL prevents a login prompt. The browser handles the 404 error internally and proceeds to access the requested page.

Adding custom headers to server response pages

You can add headers, which contain information about a custom response, to generated server responses.

Use the macros in the following table to define the information in the custom headers:

Table 6. Macros for defining custom headers

Macro	Description
`TAM_OP`	The operation code for the response. The values for this macro are identical to the values for local response redirects. See "Operation for local response redirection" on page 82.
`AUTHNLEVEL`	Authentication level required by the authentication strength policy (step-up).
`ERROR_CODE`	The hexadecimal value of the error code.
`ERROR_TEXT`	The error message text that is associated with the error code in the message catalog. This text is supplied by WebSEAL.
`USERNAME`	The name of the logged in user. WebSEAL uses the value "unauthenticated" for users who are not logged in.
`CREDATTR{`*name*`}`	The value of the user credential attribute that has the specified *name*. For example, **`CREDATTR{tagvalue_session_index}`** returns the session token.

The **`http-rsp-header`** configuration entry defines the headers that are included with the server response pages. This configuration entry is in the **`[acnt-mgt]`** stanza of the WebSEAL configuration file.

The format of the configuration entry is:

```
http-rsp-header = <header-name>:<macro>
```

where

<header-name>
The name of the header to hold the value.

<macro>
The type of value that is to be inserted as described in Table 6.

For example, the following configuration entry includes the error message text from WebSEAL in a header named **error_msg**:

```
[acnt-mgt]
http-rsp-header = error_msg:ERROR_TEXT
```

Note: You can specify this configuration entry multiple times to include more than one custom header in the response.

For example:

```
[acnt-mgt]
http-rsp-header = error_msg:ERROR_TEXT
http-rsp-header = tam-error-code:ERROR_CODE
```

For further information, see the **http-rsp-header** configuration entry in the **[acnt-mgt]** stanza in the IBM Knowledge Center.

Configuring the location URL format in redirect responses

About this task

When WebSEAL responds to a request with an HTTP 302 redirect response, the format of the URL in the **Location** header is, by default, expressed as an absolute path. For example:

```
Location: http://www.example.com/images/logo.jpg
```

If your environment requires the use of a server-relative format for the URL in the **Location** header, you can configure this specification in the WebSEAL configuration file.

Manually add the **redirect-using-relative** stanza entry to the **[server]** stanza and set the value to "true":

```
[server]
redirect-using-relative = true
```

With this configuration, the above **Location** header example will now appear as follows:

```
Location: /images/logo.jpg
```

This hidden configuration option normally defaults to "false".

This configuration change affects all redirect responses generated by WebSEAL. These redirect situations include:

- Redirect after authentication
- Redirect after logout
- Redirect after changing password
- Redirects during the e-community single signon authentication process
- Redirects during the cross-domain single signon authentication process
- Switch user processing
- Certificate authentication (**prompt-as-needed** only)
- Session displacement

This configuration change does not affect redirect responses that are returned from back-end application servers.

Local response redirection

This section contains the following topics:

Local response redirection overview

WebSEAL provides a number of static server response pages that can be used to provide server responses to client requests. These pages include:

- Error messages
- Informational messages
- Login forms

- Password management forms

In some environments, it might be beneficial to extend or modify server responses beyond what WebSEAL provides with its default set of static pages. Local response redirection provides a means to externalize the handling of such responses to an external server.

When using local response redirection, WebSEAL no longer has the responsibility of generating responses to client requests. WebSEAL's default "local response" is now redirected to a separate server that runs a custom application designed to generate appropriate responses.

When local response redirection is enabled, the redirection is used for all local WebSEAL response types: login, error, informational, and password management.

You can combine an external authentication interface application and a local response redirection application to create a complete remote solution for handling authentication and server response. Alternatively, you can implement local response redirection without using the external authentication interface for authentication handling. In this case, the server response handling is performed remotely and authentication is handled locally by WebSEAL. See “Remote response handling with local authentication” on page 88.

Local response redirection process flow

The local response redirection solution takes advantage of the fact that most devices handle HTTP 302 redirection. The following process flow illustrates how local response redirection works at a high level:

- WebSEAL receives the client request.
- WebSEAL determines that it must return a response that would normally be handled by returning a static page, such as a login or error page.
- WebSEAL builds a Location header in the response that contains the URI of a custom response handling application located on a separate server.
- WebSEAL includes, as an attribute in the query string of the Location header, the type of operation required (such as “login required”, “expired password”, or "error") and an optional set of configurable WebSEAL macros and their values.
- WebSEAL redirects the client to this custom response handling application.
- The custom response handling application is responsible for presenting an appropriate response or set of responses to the client.

Enabling and disabling local response redirection

About this task

The **enable-local-response-redirect** stanza entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file allows you to explicitly enable or disable local response redirection. Valid values are “yes” (enable) and “no” (disable).

Local response redirection is disabled by default. For example:

```
[acnt-mgt]
enable-local-response-redirect = no
```

You can customize this configuration item for a particular junction by adding the adjusted configuration item to a **[acnt-mgt:*{junction_name}*]** stanza.

where *{junction_name}* is the junction point for a standard junction (including the leading / character) or the virtual host label for a virtual host junction.

Contents of a redirected response

Redirected responses resulting from local response redirection are standard HTTP 302 redirect responses containing a specially constructed Location header.

The Location header contains the following components:

- A configurable destination URI.
- A query string indicating the required server response operation, and an optional set of configurable WebSEAL macros and their values.

The Location header has the following format:

```
Location header = location-URI?TAM_OP=operation-value[&optional-macros]
```

For details on each of the components of the Location header, refer to the following sections:

- "URI for local response redirection"
- "Operation for local response redirection" on page 82
- "Macro support for local response redirection" on page 83

URI for local response redirection

The **local-response-redirect-uri** stanza entry in the **[local-response-redirect]** stanza of the WebSEAL configuration file specifies the location (URI) of the custom application that provides the response handling service. WebSEAL uses this URI to construct the value of the Location header required by the HTTP 302 response.

The server used for local response redirection can be a junctioned server or an entirely separate server.

The format of the URI value for the **local-response-redirect-uri** stanza entry can be absolute or server-relative. Server-relative URIs must contain an appropriate junction name.

In the following example:

- **jct** is the name of the WebSEAL junction
- **redirect-app** is the name of the custom redirection application
- **response-handler** is the name of the custom response handling service (such as a servlet, JSP, or CGI)

```
[local-response-redirect]
local-response-redirect-uri = /jct/redirect-app/response-handler
```

The **local-response-redirect-uri** stanza entry must be specified when `enable-local-response-redirect = yes`.

You can customize this configuration item for a particular junction by adding the adjusted configuration item to a **[local-response-redirect:*{junction_name}*]** stanza.

where *{junction_name}* refers to the junction point for a standard junction (including the leading / character) or the virtual host label for a virtual host junction.

Operation for local response redirection

When WebSEAL receives a client request, it determines the appropriate operation required in response to the request.

To respond appropriately, the response handler application must be informed of the required response operation, as determined by WebSEAL. Example operations include serving a standard login form, a change password form, or an access denied error message.

The required operation is provided as an argument in the query string of the HTTP 302 Location URI header. The label for the operation argument is **TAM_OP**.

The following table lists the valid values for the **TAM_OP** query string argument:

Values for TAM_OP Operation Argument	Description
`acct_inactivated`	User has provided correct authentication details, but nsAccountLock is set to true for the user in Sun Java System Directory Server.
`acct_locked`	User authentication failed due to a locked (invalid) account.
`cert_login`	User must login with a certificate when **accept-client-certs = prompt_as_needed**.
`cert_stepup_http`	User tried to step-up to certificate authentication over HTTP, which is not allowed (HTTPS is required).
`eai_auth_error`	External authentication interface information returned to WebSEAL is invalid.
`error`	An error occurred. Check the ERROR_CODE macro for the hexadecimal error code. See the Error messages section of the IBM Knowledge Center.
`failed_cert`	An attempt to authenticate with a client certificate failed. Client failed to authenticate with a certificate when **accept-client-certs = required**. A valid client certificate is required to make this connection. User's certificate is invalid.
`help`	User performed an action that makes no sense, such as requesting **/pkmslogout** while logged in using basic authentication.
`login`	User needs to authenticate.
`login_success`	User successfully authenticated, but there is no last cached URL to redirect to.
`logout`	User has logged out.
`passwd`	User requests password change.
`passwd_exp`	User's password has expired.

Values for TAM_OP Operation Argument	Description
passwd_rep_failure	Password change request failed.
passwd_rep_success	Password change request succeeded.
passwd_warn	Password is soon to expire.
passwd_warn_failure	Password change not performed after notification that the password is soon to expire.
stepup	User must step-up to another authentication level. Check the AUTHNLEVEL macro for the required authentication level.
switch_user	User requested the switch user login page.
too_many_sessions	User has reached or exceeded the maximum number of allowed sessions.

The following example header shows a Location URI with a password change operation indicated in the query string:

```
Location: https://webseal/jct/handler-svr/handler?TAM_OP=passwd
```

Macro support for local response redirection

Local response redirection provides support for a subset of the macros provided by WebSEAL to customize static server response pages. Macros allow dynamic substitution of information from WebSEAL.

As with the operation information (provided by the TAM_OP argument), macros are specified as an argument in the query string of the location header. Specific characters in the macro values are URI-encoded (see "Encoding of macro contents" on page 86).

Valid WebSEAL macros for use in local response redirection include:

Macro	Description
AUTHNLEVEL	Authentication level required by authentication strength policy (step-up).
CREDATTR{name}	Used to include the contents of a specified attribute in the user credential. If the specified credential attribute does not exist in the request, the macro contains the text: 'Unknown'. For example, use the following macro name to include the **tagvalue_session_index** attribute, which contains the secret token for the session: CREDATTR{tagvalue_session_index}.
ERROR_CODE	The hexadecimal value of the error code.
ERROR_TEXT	The WebSEAL-supplied error message text associated with an error code in the message catalog.

Macro	Description
FAILREASON	Error message text associated with a Boolean rules operation.
HOSTNAME	Fully qualified host name.
HTTPHDR{name}	Used to include the contents of a specified HTTP header. If the specified HTTP header does not exist in the request, the macro contains the text: 'Unknown'. For example, the macro name to include the "Host" HTTP header is HTTPHDR{Host}.
METHOD	The HTTP method requested by the client.
PROTOCOL	The client connection protocol used. Can be HTTP or HTTPS.
REFERER	The value of the HTTP referer header from the request, or "Unknown", if none.
URL	The URL requested by the client.
USERNAME	The name of the logged in user. For users that are not logged in, the name that they used to attempt to log in is the one that is used. (See also “Customization of login forms for reauthentication” on page 186.) When using local response redirection, WebSEAL also uses the value "unauthenticated" after an inactivity timeout. This behavior differs from the processing that occurs when WebSEAL serves static pages. You can configure WebSEAL to set the USERNAME macro value to the authenticated username as it does when serving static pages. To achieve this behavior, set the **use-existing-username-macro-in-custom-redirects** configuration entry in the **[server]** stanza to yes. You must restart WebSEAL for this change to take effect.

For example, the originally requested URL might be required by an external authentication interface server that provides authentication services. The standard WebSEAL URL macro can be included in the query string of the Location header of the HTTP 302 response. The value of the macro is dynamically provided by WebSEAL.

To specify macro-supplied information (to be returned in the Location header query string), uncomment the appropriate **macro** stanza entries in the **[local-response-macros]** stanza of the WebSEAL configuration file. For example:

```
[local-response-macros]
macro = TAM_OP
#macro = USERNAME
#macro = METHOD
#macro = REFERER
#macro = HOSTNAME
#macro = AUTHNLEVEL
#macro = FAILREASON
#macro = PROTOCOL
#macro = ERROR_CODE
#macro = ERROR_TEXT
#macro = HTTPHDR{header_name}
macro = URL
```

Note: WebSEAL inserts the **TAM_OP** macro in *all* local redirect responses regardless of whether it is included in the configuration file.

The following example header shows a Location URI with the TAM_OP and URL macros (and values) indicated in the query string (entered as one line):

```
Location: https://webseal/jct/handler-svr/handler?TAM_OP=login&
 URL=%2FjctB%2Fresource.html
```

In this example, the forward slash character (/) in the value of the URL macro is encoded as *%2F*. See "Encoding of macro contents" on page 86.

If a configured macro contains no information, the macro name and the "=" delimiter still appear in the query string. For example (entered as one line):

```
Location: https://webseal/jct/handler-svr/handler?TAM_OP=stepup&USERNAME=eric&
 FAILREASON=&AUTHNLEVEL=2
```

The `HTTPHDR` macro is used to include a specified HTTP header in the query string. The desired header name should be configured in the `[local-response-macros]` stanza. For example, to insert the Host header into the query string, the following configuration entry should be added to the `[local-response-macros]` stanza:

```
macro = HTTPHDR{Host}
```

This will result in a query string similar to the following:

```
Location: https://webseal/jct/handler-svr/handler?TAM_OP=login&HTTPHDR_Host=webseal
```

Customizing macro field names:

You can customize the names that WebSEAL uses for the Location URL macros in the generated query strings. To configure these values, place a colon after the macro name followed by the customized name.

For example:

```
[local-response-macros]
macro = TAM_OP:myOperation
#macro = USERNAME
#macro = METHOD
#macro = REFERER
#macro = HOSTNAME
#macro = AUTHNLEVEL
#macro = FAILREASON
#macro = PROTOCOL
#macro = ERROR_CODE
#macro = ERROR_TEXT
#macro = HTTPHDR{header_name}
#macro = CREDATTR{name}
macro = URL:destination
```

This example configuration causes WebSEAL to generate a Location URI that contains the URL and TAM_OP macros. WebSEAL inserts the specified custom values, rather than the WebSEAL macro name, into the query string. For this example, the generated Location URI is similar to the following query string:

```
Location: https://webseal/jct/handler-svr/handler?myOperation=login&destination
=%2FjctB%2Fresource.html
```

If you configure a custom name for the HTTPHDR macro then the name of the header (**header_name**) is not included in the query string. For example, consider the following example entry in the **[local-response-macros]** stanza:

```
macro = HTTPHDR{Host}:myHost
```

The resulting query string looks similar to the following example:

```
Location: https://webseal/jct/handler-svr/handler?myOperation=login&destination=
%2FjctB%2Fresource.html&myHost=webseal
```

You can configure multiple HTTPHDR entries to be included in the query string. You can customize the macro field names for each of these entries.

Note: WebSEAL inserts the **TAM_OP** macro in *all* local redirect responses regardless of whether it is included in the configuration file. You can configure the name that WebSEAL uses for this macro by including a customized name in the configuration file.

Encoding of macro contents:

Some macro content contains user-provided data such as the requested URI or the Referer header of that request. It is important for security reasons to ensure that reserved, or special characters in client-supplied data are encoded.

WebSEAL URI encodes macro contents to ensure that the content does not return reserved, or special characters back to the client. URI encoding is an international standard that allows you to map the wide range of characters used worldwide into the limited character-set used by a URI.

Notes on encoding macro contents:

- WebSEAL always applies URI encoding to macro contents, even if the original data has already been encoded.
- Encoded macro contents must be decoded using standard URI decoding rules.
- URI encoding increases the string length of macro content, and therefore the Location header (where the content is embedded in the query string). For a discussion of Location header length issues, see "Macro content length considerations."

Macro content length considerations:

Information supplied by macros increases the string length of the Location URI header. URI encoding of macro content further increases this string length.

Some client applications (such as WAP browsers on cellular phones) have URI length limitations due to the small memory capacity of the device. If a URI exceeds the length limitation on such a client device, errors can occur and the link will likely fail.

WebSEAL does not impose any length restrictions on the Location URI header. Therefore, when configuring macros for local response redirection, you must carefully consider the possible limitations of client devices that access your site. You can estimate the length of the Location header by determining the fixed lengths of the URI and factoring in the expected sizes of any macros used in the query string.

The following table provides information about the possible lengths of the content provided by the macros used for local response redirection:

Macro	Size of Content
AUTHNLEVEL	No more than 10 characters.
ERROR_CODE	No more than 20 characters.

Macro	Size of Content
ERROR_TEXT	The error message length.
FAILREASON	The error message length.
HOSTNAME	The length of the HOST header of the corresponding request, or the fully qualified host name of the WebSEAL system if the HOST header is not present.
METHOD	Length of request method (such as GET or POST). No more than 20 characters.
PROTOCOL	No more than 10 characters.
REFERER	The length of the REFERER header of the corresponding request.
URL	Length of the request URI.
USERNAME	Maximum length defined by user name length policy for this implementation of WebSEAL.
HTTPHDR{name}	Length of the specified HTTP header.
CREDATTR{name}	Length of the contents for the specified attribute in the user credential.

Local response redirection configuration example

The following example steps summarize the configuration required to implement local response redirection. This example illustrates the combined implementation of local response redirection with an external authentication interface service.

The following variables are used in this example:

- **jct** is the name of the WebSEAL junction
- **eai-redirect-app** is the name of the custom application that provides combined external authentication interface and local response redirection services
- **authn-handler** is the name of the custom authentication service (such as a servlet, JSP, or CGI)
- **response-handler** is the name of the custom response handling service (such as a servlet, JSP, or CGI)

Example:

- A custom external authentication interface service is implemented on the junctioned server to handle the WebSEAL authentication process:
  ```
  webseal/jct/eai-redirect-app/authn-handler
  ```
- Enable local response redirection to handle responses to requests:
  ```
  [acnt-mgt]
  enable-local-response-redirect = yes
  ```
- Specify the location of the custom response handling application (Location URI):
  ```
  [local-response-redirect]
  local-response-redirect-uri = /jct/eai-redirect-app/response-handler
  ```
- Specify that the URL requested by the client (supplied by the URL macro) be returned in the Location URI query string of the local response redirection:
  ```
  [local-response-macros]
  macro = URL
  ```
- Client requests a resource requiring authentication:
  ```
  https://webseal/jctB/resource.html
  ```

- WebSEAL returns an HTTP 302 response containing the following Location URI header (entered as one line). The login operation required is specified in the query string as: `TAM_OP=login`:

  ```
  Location: https://webseal/jct/eai-redirect-app/response-handler?TAM_OP=login&
  URL=http%3A//webseal/jctB/resource.html
  ```
- The custom response handling application provides a response to the client (consistent with a login operation) and makes use of the resource URL information provided.
- The client completes one or more interactions with the custom response handler and is eventually routed to the external authentication interface where the actual authentication is performed.

Technical notes for local response redirection

- If a client makes a bad or erroneous request to WebSEAL as a result of a redirected error page, WebSEAL returns a static error message page rather than initiating another redirection operation.
- To avoid a possible redirection loop condition, the custom message handling application must be a resource that is available to unauthenticated users. Security Access Manager ACLs must be configured to ensure this availability to unauthenticated users.
- When implementing external authentication interface and local response redirection together, ensure that the configured location URI is not the same as any external authentication interface trigger URLs.
- All requests that are redirected by local response redirection are cached in the same manner as for default response handling.

Remote response handling with local authentication

You can implement local response redirection without using the external authentication interface. In this case, the server response handling is performed remotely and authentication is handled locally by WebSEAL. For example, the remote response handler can serve a login page that requires a local WebSEAL authentication handler such as `pkmslogin.form` to implement the authentication process.

In this example, the login page served by the remote response handler contains a FORM tag with an ACTION attribute. The value of the ACTION attribute points to the location of the local WebSEAL authentication handler (`pkmslogin.form`). When the client submits the completed login form, the data is directed to this handler.

When WebSEAL receives a request for `pkmslogin.form`, it responds by invoking the appropriate authentication mechanism and passing the appropriate authentication data to this mechanism.

Note: The `pkmslogin.form` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

You can use the appropriate static HTML response pages provided by WebSEAL as templates for your custom pages. If necessary, edit the pages to customize the content for your environment. Ensure that all URLs are expressed to correctly satisfy the filtering rules of WebSEAL.

Junction filtering issues for the ACTION URL:

The login page served by a response handler located on a junctioned server is filtered by WebSEAL when it is returned to the client over the junction. The page might contain a FORM tag similar to WebSEAL's standard static HTML login response page (`login.html`):

```
<FORM METHOD=POST ACTION="/pkmslogin.form">
```

If this FORM tag appears in a page served by a remote response handler located on a junctioned server, WebSEAL filters the server-relative ACTION URL by prepending the junction name to the path:

```
/jct/pkmslogin.form
```

Because the `pkmslogin.form` authentication handler is local to WebSEAL, WebSEAL cannot find the filtered version of the URL and the request for an authentication operation from WebSEAL fails.

You must ensure that the response handler serves a login page with an ACTION attribute that correctly points to `pkmslogin.form`. Use a relative path expression to navigate upwards from the location where the custom login page is served. For example, if the login page is served from the following JSP (JavaServer Pages) application:

```
/jct/redirect-app/custom-login.jsp
```

the ACTION URL in this page must be expressed as follows:

```
<FORM METHOD=POST ACTION="../../pkmslogin.form">
```

WebSEAL does not filter relative path names. Therefore, this path is correctly resolved by the client browser as being located in the document root space of the WebSEAL server.

For complete information about WebSEAL filtering rules, see "Filter rules for tag-based static URLs" on page 423.

HTML redirection

When a user successfully authenticates, WebSEAL typically uses an HTTP 302 response to redirect the user back to the resource that was originally requested. This process causes problems for applications that rely on HTML fragments as the fragment information is stored locally in the browser and is lost during redirection.

Alternatively, you can configure WebSEAL to enable HTML redirection. HTML redirection causes WebSEAL to send a static page back to the browser instead of a 302 redirect. The browser can then use JavaScript or any other code that is embedded in this static page to perform the redirect. WebSEAL provides the macro LOCATION, which contains the URL for the redirection.

Enabling HTML redirection

About this task

Use the **enable-html-redirect** stanza entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file to enable or disable HTML redirection. You can use HTML redirection, in conjunction with some JavaScript code, to preserve the HTML fragment in the response.

Valid values are “yes” (enable) and “no” (disable). HTML redirection is disabled by default. For example:

```
[acnt-mgt]
enable-html-redirect = no
```

If you enable this configuration entry, WebSEAL returns the redirect page that is specified by **html-redirect** in the **[acnt-mgt]** stanza instead of an HTTP 302 response. For example:

```
[acnt-mgt]
html-redirect = redirect.html
```

Note: This configuration for HTML redirection does not affect redirect responses that are returned by junctioned web servers.

Preserving HTML fragments on redirection

About this task

WebSEAL provides an example configuration of HTML redirection. This example uses cookies to store the original URL in the browser and JavaScript to later read that URL and perform the redirection. The example shows how to save the HTML fragment during an authentication operation.

In addition to enabling HTML redirection, you must modify the JavaScript on the user interface form (such as `login.html`) to store the HTML fragment for the subsequent redirect. The page before redirection must store the originally requested resource, complete with HTML fragment, in a cookie in the client web browser cookie jar. When the redirection page returns, the example JavaScript reads the cookie value and redirects the client to the originally requested resource while preserving the HTML fragment.

You can use this configuration with any of the login mechanisms for which WebSEAL provides a login page. You must uncomment the line of JavaScript that sets the **TAMOriginalURL** cookie in the corresponding WebSEAL templates. These WebSEAL templates include: `login.html`, `stepuplogin.html` and `certlogin.html`.

For example, to preserve the HTML fragment with forms-based authentication you must complete the following steps:

Procedure

1. Update the **login.html** page. Uncomment the bold line of JavaScript to set a cookie named **TAMOriginalURL** on the client browser with a URI encoded copy of the originally requested URL. This JavaScript is included in the default `login.html` file that is supplied with WebSEAL.

   ```
   <SCRIPT LANGUAGE=JavaScript>
   var warningString = "<B>WARNING:</B> To maintain your login session,
       make sure that your browser is configured to accept Cookies.";
   document.cookie = 'acceptsCookies=yes';
   if(document.cookie == ''){
    document.write(warningString);
   }
   else{
   document.cookie = 'acceptsCookies=yes; expires=Fri, 13-Apr-1970 00:00:00 GMT';
   document.cookie = 'TAMOriginalURL=' + encodeURIComponent(window.location) +
      "; Path=/;";
   }
   </SCRIPT>
   ```

2. Ensure that the **html-redirect** configuration entry in the **[acnt-mgt]** stanza specifies the **redirect.html** file that is supplied with WebSEAL. This file contains the following script, which parses the cookies in the browser and looks for the **TAMOriginalURL** cookie set by the preceding page. When this cookie is found, it is URI decoded and set to expire immediately. The line containing `window.location.href` then performs the redirection.

   ```
   <SCRIPT LANGUAGE=JavaScript>
   var redirect = "TAMOriginalURL=";
   var cookies = document.cookie.split(';');
   var redirectURL = "%LOCATION%";
   for(var i=0; i<cookies.length; i++) {
    var cookie = cookies[i];
    while (cookie.charAt(0)==' ') {
     cookie = cookie.substring(1, cookie.length);
     if (cookie.indexOf(redirect) == 0) {
      redirectURL = cookie.substring(redirect.length, cookie.length);
      document.cookie = 'TAMOriginalURL=; expires=Thu, 01-Jan-70 00:00:01 GMT;';
      i = cookies.length;
      break;
     }
    }
   }
   window.location.href = decodeURIComponent(redirectURL);
   </SCRIPT>
   ```

   Note: In some situations it is not possible to set a cookie before the redirection is to take place, such as when a local response redirect is performed. For these situations, WebSEAL includes the macro %LOCATION%, which is inserted into the static redirect page. This macro contains the complete URL of the redirection and can be used when cookies cannot be set. However, in this situation any HTTP fragment information is lost.

Web server security configuration

This chapter contains information about configuring added security for the WebSEAL server.

Topic Index:

Cryptographic hardware for encryption and key storage

Cryptographic hardware concepts

WebSEAL uses GSKit for SSL communication and key management to provide interface support for cryptographic hardware.

Cryptographic hardware can provide one or both of the following features:

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

Hardware cryptographic acceleration and key storage apply to the following WebSEAL connections:

- Browser to WebSEAL
- WebSEAL to back-end junctioned server

The following product functions or features do not currently support cryptographic hardware integration:

- Symmetric key operations (including key storage), such as eCSSO, CDSSO, LTPA, and any other SSL connections.
- Any cryptographic operations (including certificate and key storage) completed with SSL configured between the Security Access Manager directory client and directory server.
- Any cryptographic operations (including certificate and key storage) completed when Security Access Manager components communicate as part of the authorization database management (pdadmin or database replication).
- Any cryptographic operations (including certificate and key storage) completed with SSL configured between WebSEAL and the Security Access Manager session management server.

Configuration of the Cipher engine and FIPS mode processing

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

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

Valid values for this entry are:

- **Default**

  This value tells GSKit to select the optimal cryptographic base to use. For WebSEAL Version 7, the default cryptographic base is ICC.
- **ICC**

You can specify whether to enable FIPS mode processing. FIPS mode processing is disabled by default. To enable FIPS mode processing, set the following entry:

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

Set the value to "yes" when you are using ICC and you want to use the FIPS 140-1 approved protocols and ciphers.

Configuring WebSEAL for cryptographic hardware

About this task

Perform the following steps to configure WebSEAL for cryptographic hardware:

1. Install the cryptographic card and device driver:
About this task

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

2. Create a token device label and password to store WebSEAL keys:
About this task

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

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

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

For example:

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

3. Configure iKeyman to use the PKCS#11 module:

About this task

The **iKeyman** utility is packaged with the Java Runtime Environment version 6.0 or later. You must configure **iKeyman** for the PKCS#11 device module (shared library) of the installed cryptographic hardware device. The **iKeyman** utility uses this module to understand the following components:

- The WebSEAL token label for the hardware device.
- The password or PIN for the token.
- The key label of any WebSEAL key stored on the device.

Procedure

1. Locate the `java.security` file in the directory location that is applicable to your environment:

   Solaris:
   User-defined location during Java installation. For example, `/usr/java`.

   Linux: `/opt/ibm/java-s390x-60/jre/lib/security`

   AIX®: `/usr/java6_64/jre/lib/security`

   Windows:
   `C:\Program Files\IBM\Java60\jre\lib\security`
2. Edit this file by adding a line to include the `IBMPKCS11Impl` provider in the provider list. That is, `com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl`. For example:

   ```
   #
   # List of providers and their preference orders:
   #
   security.provider.1=com.ibm.crypto.provider.IBMJCE
   security.provider.2=com.ibm.jsse.IBMJSSEProvider
   security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
   security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
   security.provider.5=com.ibm.security.cert.IBMCertPath
   security.provider.6=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl
   security.provider.7=com.ibm.security.cmskeystore.CMSProvider
   security.provider.8=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
   ```
3. Save these updates.

Results

When the shared library is configured, the **iKeyman** utility includes a new menu option: **PKCS11Direct**. Now you can use **iKeyman** to create, store, and manipulate keys for WebSEAL on the cryptographic hardware.

4. Use iKeyman to open the WebSEAL token device:
Procedure

1. Start the **iKeyman** utility that is packaged with the Java Runtime Environment version 6.0 or later.
2. Select **Key Database File**, then **Open**.
   A separate Open dialog box displays.
3. In the Open dialog window, select **Cryptographic Tokens** from the **Key database type** menu.
4. If you have the cryptographic token specified in the `java.security` file, the dialog box contains both the path and the library. If you do not see the library, you can use the **Browse** menu option. Click **OK** when this step is complete.
5. Additionally, if you want to open an existing secondary key database (for key data not stored on the cryptographic hardware-such as CA root certificates), check **Open Existing Key Database**.
6. Browse for and select the default WebSEAL key database:
   UNIX or Linux
   /var/pdweb/www-*instance*/certs/pdsrv.kdb
   Windows
   C:\Program Files\Tivoli\pdweb\www-*instance*\certs\pdsrv.kdb
7. Click **OK**.
   The Token Password dialogue box displays.
8. Enter the default password `pdsrv`. Click **OK**.

Results

The main **iKeyman** window returns.

5. Request and store the WebSEAL server certificate:
Procedure

1. Follow instructions in the *IBM Global Security Kit: Secure Sockets Layer Introduction and iKeyman User's Guide* to request a secure, signed digital certificate for WebSEAL from a Certificate Authority (CA).
2. Follow instructions in the *IBM Global Security Kit: Secure Sockets Layer Introduction and iKeyman User's Guide* to receive the WebSEAL certificate from the CA and store it in a key database. When performing this procedure, select the token device representing the cryptographic hardware as the storage location for the certificate.

Results

When it is stored on the token device, the key (certificate) appears (for example) as:`websealtoken:webseal`
The WebSEAL key is stored on the cryptographic hardware and assigned to the token device labeled "websealtoken".

6. Configure WebSEAL and GSKit to use the PKCS#11 shared library:

Procedure

1. In the WebSEAL configuration file, specify the names of the token label and password under the **[ssl]** stanza:
   For this example:
   ```
   [ssl] pkcs11-token-label = websealtoken pkcs11-token-pwd             =
   secret
   ```
2. (For IBM 4960 only) In the WebSEAL configuration file, modify the WebSEAL instance UNIX group account configuration to specify group **pkcs11**. This allows WebSEAL to access the PKCS token:
   ```
   [server]
   unix-group = pkcs11
   ```

7. Modify the WebSEAL server certificate label:
About this task

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

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

For this example:

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

8. Configure WebSEAL for PKCS#11 symmetric algorithms:
About this task

You can configure WebSEAL to support the GSKit option for using PKCS#11 for symmetric algorithms.

To enable PKCS#11 for symmetric algorithms, uncomment the **pkcs11-symmetric-cipher-support** stanza entry in the **[ssl]** stanza of the WebSEAL configuration file and set the value to "yes". For example:

```
[ssl]
pkcs11-symmetric-cipher-support = yes
```

To disable support for symmetric algorithms, uncomment the **pkcs11-symmetric-cipher-support** stanza entry in the **[ssl]** stanza of the WebSEAL configuration file and set the value to "no". For example:

```
[ssl]
pkcs11-symmetric-cipher-support = no
```

The PKCS#11 symmetric cipher support does not include removable devices. If a removable device is encountered, it is ignored even if the support has been requested. Additionally, not all devices support symmetric ciphers. Refer to the appropriate vendor documentation for the device you are using.

9. Restart WebSEAL:
Procedure

1. Restart WebSEAL for all cryptographic hardware configuration to take effect.
2. Verify that WebSEAL is using the cryptographic hardware by examining entries contained in the `msg_webseald.log` file.

Configuring network Hardware Security Module (HSM) support

You can register a network HSM device with the local management interface. WebSEAL can then be configured to use this HSM for the secure storage of SSL keys.

About this task

The appliance supports the use of the following HSM devices:

- nCipher nShield Connect HSM

  The appliance supports the nCipher nShield Connect HSM device. To enable the integration with this device the 'IBM Security Access Manager nCipher nShield Connect HSM Extension' should be installed on the appliance. This extension is available for download from the IBM Security App Exchange (https://exchange.xforce.ibmcloud.com/hub/IdentityandAccess).

  Due to a limitation in key protection type support, the appliance does not support “HSM Pool mode”. The appliance continues to support high availability by using the load sharing capabilities provided by nShield HSMs.
- SafeNet Luna Network HSM

  The appliance supports the SafeNet Luna Network HSM device. To enable the integration with this device the 'IBM Security Access Manager SafeNet Luna Network HSM Extension' must be installed on the appliance. This extension is available for download from the IBM Security App Exchange. See https://exchange.xforce.ibmcloud.com/hub/IdentityandAccess.

Note: The appliance can connect to a maximum of one nCipher nShield Connect device and multiple SafeNet Luna SA devices.

Perform the following steps to configure WebSEAL for the network HSM device.

Procedure

1. Create a network key file with the local management interface.
   a. From the top menu, select **Manage System Settings** > **Secure Settings** > **SSL Certificates**.
   b. From the menu bar, click **New**.
   c. On the Create SSL Certificate Database page, enter the name of the certificate database that you want to create.
   d. Select **Network** as the type of the certificate database.
   e. Complete the **Token Label** and **Passcode** fields.
   f. Select the HSM type.
      - If you select **nCipher nShield Connect** as the HSM type, complete the **HSM IP Address** and **RFS IP Address** fields on the **nCipher nShield Connect** tab. The rest of the fields are optional.
      - If you select **SafeNet Luna SA** as the HSM type, complete the **IP Address** and **Admin Password** fields on the SafeNet tab.

        Note: You can use the appliance to manage the certificates that are contained on the HSM device. However, some operations, such as certificate extract, are not supported.
   g. Click **Save**.
2. Edit the WebSEAL configuration file directly or through the Edit panel in the local management interface to make the following changes.

   a. Set the value of the **pkcs11-keyfile** configuration entry in the **[ssl]** stanza to be the name of the pkcs11 key file that contains the configuration information for the network HSM device.
   b. Set the **webseal-cert-keyfile-label** configuration entry in the **[ssl]** stanza, which defines the WebSEAL key file label, to use a key from the HSM device.
3. Restart WebSEAL for the changes to take effect.

Registering the appliance as a client in the SafeNet Luna SA HSM:

Configure the Luna SA HSM to allow the appliance to access the required partitions.

Before you begin

After you create an SSL certificate database on the appliance for the HSM, the appliance generates a certificate and transfers it to the HSM. Before you start these steps, check if the certificate exists on the HSM.

The IP address that the appliance uses to communicate with the HSM is required to complete these steps. This IP address can be found on the Details tab of the certificate database on the appliance.

Note: Configure only one client on a single HSM for the appliance, regardless of how many certificate databases are created for that HSM.

Procedure

1. Use SSH to access the Luna SA HSM as **admin**.
2. Run the following command:

   ```
   client register -client <client_name> -hostname <client_ip>
   ```

   where *client_name* is a name that represents the appliance and *client_ip* is the IP address that is listed on the Details tab.
3. Run the following command:

   ```
   client assignPartition -client <client_name> -partition <partition_name>
   ```

   where *client_name* is the name that is used in the previous step and *partition_name* is the partition against which the SSL Certificate database was configured.

Deleting a client from the SafeNet Luna SA HSM:

If all SSL certificates that refer to a Luna SA HSM are deleted from the appliance, remember to delete the client from that HSM.

Procedure

1. Use SSH to access the Luna SA HSM as **admin**.
2. Run the following command:

   ```
   client delete -client <client_name>
   ```

   where *client_name* is the name that is used to represent the appliance.

Configuring RSA one-time password support

Configure the Web Reverse Proxy to use RSA tokens as an authentication mechanism.

Before you begin

The appliance embeds an RSA agent. It is compatible with the following RSA server versions:

- RSA Authentication Manager 6.1 or later
- RSA Authentication Manager 7.1 or later

On your RSA server, generate the `sdconf.rec` file, which is the configuration file for connecting to the RSA Authentication server. See your RSA Authentication server documentation for details on how to create this file.

Procedure

1. Upload the generated `sdconf.rec` file to the appliance.
   1. Log in to the local management interface of the appliance.
   2. Go to **Secure Web Settings** > **Global Settings** > **RSA SecurID Configuration**.
   3. Click **Upload** to upload the `sdconf.rec` file. The status area indicates one of two statuses:

      Unavailable
      Upload is not completed.

      Available
      Upload is complete.
   4. Optional: To remove the contents of a previously loaded file, click **Clear**.
2. Register the appliance with the RSA Server.

   Note: The appliance must be registered with the RSA server before the **Test** button or RSA authentication can be used.
   1. Take note of the values that are displayed in the **RSA Server** and **Agent IP Address** fields on the RSA Configuration page of the appliance local management interface.
   2. On the RSA server, create an authentication agent with the **Agent IP Address** in the previous step. For Agent type, choose **Standard**.
3. Generate the node secret file. The node secret file is automatically generated on the first authentication attempt.
   1. On the RSA Configuration page of the appliance local management interface, click **Test** to perform a test authentication.
   2. Enter the user name and passcode for a valid RSA user.
   3. Click **OK** to generate the node secret file. The status area of the node secret file indicates one of two statuses:

      Unavailable
      The node secret file was not generated.

      Auto-generated
      The node secret file was automatically generated.
   4. Optional: To remove the contents of a previously generated node secret file, click **Clear**.

      Note: This step must be performed if the node secret file is changed on the RSA server. If the node secret file is cleared on the appliance, then it must also be cleared for the configured authentication agent on the RSA Manager manually. Errors occur until the node secret file is cleared on the RSA Manager.

Configuring WebSEAL to support only Suite B ciphers

You can configure WebSEAL to use only Suite B ciphers when negotiating an SSL connection.

About this task

Suite B is a set of cryptographic standards, protocols, and algorithms that the National Security Agency (NSA) developed in 2005. This suite defines security standards for protecting classified information. NSA Suite B includes the Advanced Encryption Standard (AES) and a set of cryptographic algorithms for key exchange, digital signatures, and hashing.

Suite B meets the NSA security standards for classified government communications up to the SECRET level. For more information, go to the NSA website and search for *Suite B Cryptography*.

Use the **gsk-attr-name** and **jct-gsk-attr-name** entries to configure WebSEAL support for Suite B ciphers. Set GSKit attribute 454 to the value 1.

The **gsk-attr-name** configuration entry is available in the **[ssl]**, **[dsess-cluster]**, and **[tfim-cluster:<cluster>]** stanzas. The **[ssl]** stanza also includes the **jct-gsk-attr-name** configuration entry. These stanza entries specify the additional GSKit attributes to use when initializing SSL connections as follows:

[ssl] stanza

The **gsk-attr-name** applies to SSL connections with clients.

The **jct-gsk-attr-name** applies to SSL connections with junctioned servers.

[dsess-cluster] stanza

The **gsk-attr-name** applies to SSL connections with the distributed session cache.

[tfim-cluster:<cluster>]

The **gsk-attr-name** applies to SSL connections with the Federation Runtime.

Example

The following entry configures WebSEAL to use only Suite B ciphers for client connections:

```
[ssl]
gsk-attr-name = enum:454:1
```

Configuring NIST SP800-131A compliance

Special Publication 800-131a (SP 800-131a) is an information security standard of the National Institute of Standards and Technology (NIST). SP 800-131a requires longer key lengths and stronger cryptography than other standards. You can configure WebSEAL to comply with NIST SP800-131A when it is negotiating SSL connections.

About this task

Use the **nist-compliance**, **ssl-nist-compliance**, and **jct-nist-compliance** entries to configure NIST SP800-131A compliance.

Enabling NIST SP800-131A compliance automatically configures the following settings:

- Enables FIPS mode processing.
- Enables TLS v1.2.

  Note: TLS v1 and TLS v1.1 are not disabled.
- Enables the appropriate signature algorithms.
- Sets the minimum RSA key size to 2048 bytes.

You can individually enable NIST SP800-131A for specific communication channels by using the following configuration entries:

[ssl] stanza
WebSEAL uses the **nist-compliance** configuration entry as the global NIST setting for SSL connections with clients. If the **ssl-nist-compliance** configuration entries are not present in the WebSEAL configuration file, WebSEAL uses the **nist-compliance** value as the default NIST setting for all client connections.

[dsess-cluster] stanza
The **ssl-nist-compliance** configuration entry controls NIST SP800-131A compliance for SSL connections with the distributed session cache.

[rtss-cluster:<*cluster*>] stanza
The **ssl-nist-compliance** configuration entry controls NIST SP800-131A compliance for runtime security services SOAP communication.

[tfim-cluster:<*cluster*>] stanza
The **ssl-nist-compliance** controls NIST SP800-131A compliance for SSL connections with the Federation Runtime.

[junction] stanza
The **jct-nist-compliance** configuration entry controls NIST SP800-131A compliance for SSL connections to junctioned servers.

For more information about these configuration entries, see the Reference topics in the IBM Knowledge Center.

Example

For example, the following configuration disables NIST SP800-131A compliance for SSL connections with the distributed session cache, but enables NIST SP800-131A compliance for other client connections.

```
[dsess-cluster]
ssl-nist-compliance = no

[ssl]
nist-compliance = yes
```

Prevention of vulnerability caused by cross-site scripting

Cross-site scripting is a known technique for deploying malicious scripts on browsers. Web servers that incorrectly reflect user-supplied data to the browser without properly escaping the data are vulnerable to this type of attack.

WebSEAL provides limited protection against cross-site scripting for junctioned applications through URL string filtering. Other solutions, such as the Web Content Protection feature of the appliance, can also help protect against these types of attacks.

Configuration of URL string filtering

You can configure WebSEAL to reject an incoming request if the request URL contains a defined string pattern. WebSEAL rejects incoming URL requests if they contain any of the string patterns that are defined in the **[illegal-url-substrings]** stanza.

Note: The **[illegal-url-substrings]** feature is deprecated. IBM might remove this feature in a subsequent release of the product.

In the WebSEAL configuration file, add a separate entry in the **[illegal-url-substrings]** stanza to represent each string pattern that you want WebSEAL to reject. For example:

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

If WebSEAL detects any configured string fragment in the requested URL, WebSEAL rejects the request and returns a 400 "Bad Request" error page.

WebSEAL, by default, filters strings that contain `<script`. If you require additional filtering, you must create the **[illegal-url-substrings]** stanza and list all substrings individually.

You can completely disable the URL string filtering feature, including the default behavior, by placing an empty **[illegal-url-substrings]** stanza in the WebSEAL configuration file.

Functional notes:

- Substring entries in the configuration file must be ASCII. WebSEAL decodes URLs before checking for the presence of these strings. Therefore, if these strings are present in the URL in another encoding, WebSEAL still filters them.
- WebSEAL locates these substrings by using a search that is not case sensitive.
- Substring filtering accommodates multi-byte characters.

Prevention of Cross-site Request Forgery (CSRF) attacks

Cross-site request forgery (CSRF) is a type of malicious website attack. A CSRF attack is sometimes called a *one-click attack* or *session riding*. This type of attack sends unauthorized requests from a user that the website trusts.

CSRF uses the trust that a site has in the browser of an authenticated user for malicious attacks. CSRF uses links or scripts to send involuntary HTTP requests to a target site where the user is authenticated. Unless precautions are taken, the WebSEAL management pages, such as **/pkmslogout**, are susceptible to a CSRF attack. For example, an attacker might get an authenticated WebSEAL user to involuntarily log out by getting their browser to follow a link to **/pkmslogout**.

You can configure WebSEAL to help mitigate this type of vulnerability.

Secret token validation

You can configure WebSEAL to require that certain management operation requests include a secret token. WebSEAL uses the secret token in the received request to validate its authenticity.

Secret token validation affects the following WebSEAL management pages:

- /pkmslogin.form
- /pkmslogout
- /pkmslogout-nomas
- /pkmssu.form
- /pkmsskip
- /pkmsdisplace
- /pkmspaswd.form
- /pkmsoidc

Use the **enable-secret-token-validation** configuration entry in the **[acnt-mgt]** stanza to enable secret token validation. By default, **enable-secret-token-validation** is set to `false`, which disables secret token validation.

If you want WebSEAL to use secret token validation, set this entry to `true`:

```
[acnt-mgt]
enable-secret-token-validation = true
```

When secret token validation is enabled, WebSEAL adds a token to each session and validates the "token" query argument for these account management requests. For example, the request to `/pkmslogout` changes to `pkmslogout?token=<`*`value`*`>`, where `<`*`value`*`>` is the unique session token.

Note: This setting modifies the URLs for these WebSEAL management pages. Each of the affected management requests must contain a "token" argument with the current session token. For example, `/pkmslogout?token=a861582a-c445-4462-94c9-b1074e135b9f`.

If secret token validation is enabled and the token argument is missing from the request or does not match the real session token, WebSEAL returns a "400 Bad Request" error page.

If you are using secret token validation then WebSEAL includes the session token as the **tagvalue_session_index** attribute in the user credential. WebSEAL provides a **CREDATTR** macro that you can use to access a credential attribute and insert it into the following locations:

- Generated HTML pages (for example, **/pkmshelp**).
- Local response redirect URLs. See “Macro support for local response redirection” on page 83.
- HTTP response headers (**http-rsp-header** configuration item). See “Adding custom headers to server response pages” on page 78.

To reference the secret token, use the **CREDATTR{tagvalue_session_index}** macro.

Note: Secret token validation does not affect the CDSSO or eCSSO functionality in WebSEAL.

Referrer validation

To help mitigate CSRF attacks, you can configure WebSEAL to validate the **referer** header in incoming HTTP requests. WebSEAL compares this **referer** header with a list of configured **allowed-referers** to determine whether the request is valid.

Referrer validation affects the following WebSEAL management pages:

- /pkmslogout
- /pkmslogout-nomas
- /pkmspasswd.form
- /pkmscdsso
- /pkmsvouchfor
- /pkmsskip
- /pkmsdisplace

Use the **allowed-referers** configuration entry in the **[acnt-mgt]** stanza to define valid **referer** headers. The value for this entry can contain alphanumeric characters, spaces, periods, and wildcard characters.

Note: You can specify this entry multiple times to define multiple valid **referer** headers. WebSEAL uses all of these entries to validate the referrer.

You can set the **allowed-referers** to %HOST%, which is a special filter. This filter indicates to WebSEAL that a referrer is valid if the host name portion of the **referer** HTTP Request header matches the **host** HTTP Request header.

If you want WebSEAL to use referrer validation, you must include at least one **allowed-referers** entry. For example:

```
[acnt-mgt]
allowed-referers = %HOST%
```

When attempting to validate an incoming request, if WebSEAL does not find an **allowed-referers** entry that matches the **referer** header in the request then the request fails. WebSEAL returns an error page.

Note: If there are no **allowed-referers** entries, referrer validation is disabled and WebSEAL does not validate the **referer** headers in incoming requests.

Reject unsolicited authentication requests

For extra mitigation against cross-site request forgery (CSRF), you can configure WebSEAL to reject any unsolicited login requests. This configuration ensures that WebSEAL does not process login requests without first issuing a login form.

The following steps outline the general process for a client to authenticate to WebSEAL and access a protected resource:

1. The client requests the protected resource.
2. WebSEAL detects that the client is not authenticated so WebSEAL returns a login form to the client.
3. The client enters login information and submits the form to WebSEAL.
4. WebSEAL processes the login information as follows:
   a. Authenticates the user.

   b. Creates a session.
   c. Sends a redirect to the requested resource.
5. The client requests the protected resource.
6. WebSEAL detects that the user is authenticated and returns the resource to the client.

By default, it is possible for a client to skip directly to step 3 and initiate authentication with WebSEAL by sending through an unsolicited login request. However, you can configure WebSEAL to reject these unsolicited requests. You can set **allow-unsolicited-logins** in the **[server]** stanza to no to ensure that the first two steps are required for a client to gain access to a resource. If you set this option to no, WebSEAL must always issue a login form to unauthenticated clients.

By default, **allow-unsolicited-logins** is set to yes, which means that WebSEAL does accept unsolicited authentication requests.

Set this entry to no if you are concerned that CSRF might cause a user to inadvertently authenticate with authentication data provided by an attacker.

```
[server]
allow-unsolicited-logins = no
```

Suppression of WebSEAL and back-end server identity

This section contains the following topics:

Suppressing WebSEAL server identity

HTTP responses normally include a **Server** header containing the identity and version of the server that is sending the response.

About this task

The following example illustrates the header output for a response sent from WebSEAL:

```
Content-Type: text/html
Date: Tue, 09 Nov 2004 02:34:18 GMT
Content-length: 515
Server: WebSEAL/6.0.0
Last-Modified: Thu, 04 Nov 2004 08:03:46 GMT
Connection: close
```

For security reasons, you might want WebSEAL to suppress the **Server** header in its responses to clients.

To suppress WebSEAL server identity in HTTP server responses, set the **suppress-server-identity** stanza entry in the **[server]** stanza of the WebSEAL configuration file to "yes":

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

The default setting is "no".

Suppressing back-end application server identity

About this task

HTTP responses normally include a **Server** header containing the identity and version of the server that is sending the response. The following example illustrates the header output for a response sent from a back-end junctioned application server:

```
Content-Type: text/html
Date: Tue, 09 Nov 2004 03:34:18 GMT
Content-Length: 515
Server: IBM_HTTP_SERVER/1.3.19Apache/1.3.20 (Win32)
Last-Modified: Thu, 04 Nov 2004 09:03:46 GMT
Connection: close
```

To suppress back-end application server identity in HTTP server responses, set the **suppress-backend-server-identity** stanza entry in the **[server]** stanza of the WebSEAL configuration file to "yes":

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

The default setting is "no".

Disabling HTTP methods

You can block the use of HTTP methods to request local or remote resources to reduce security vulnerability.

About this task

Modify the WebSEAL configuration file to disable specific HTTP methods.

Procedure

- Use the **`http-method-disabled-local`** stanza entry in the **`[server]`** stanza to disable the use of specific methods to request resources over a local junction.
- Use the **`http-method-disabled-remote`** stanza entry in the **`[server]`** stanza to disable the use of specific methods to request remote resources.

  You can use a comma (**,**) to separate multiple methods. For example, the following configuration entry blocks access to the TRACE and PUT methods over local junctions:

  ```
  [server]
  http-method-disabled-local = TRACE,PUT
  ```

  By default, WebSEAL disables the TRACE, PUT, DELETE, CONNECT methods. The default values for these configuration entries are as follows:

  ```
  [server]
  http-method-disabled-local  = TRACE,PUT,DELETE,CONNECT
  http-method-disabled-remote = TRACE,PUT,DELETE,CONNECT
  ```

  Note:

  You can enable a blocked method by removing the method name from these two entries in the WebSEAL configuration file.

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

  ```
  [server]
  http-method-disabled-local =
  ```

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

```
[server]
http-method-disabled-remote =
```

Platform for Privacy Preferences (P3P)

This section contains the following topics:

Compact policy overview

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

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

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

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

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

- Session cookie
- Failover cookie
- e-community cookie
- LTPA cookie

There is no need to configure policy for the e-community cookie. The cookie contents are limited to specifying the location of the Web server to which the user authenticated. This cookie contains no information that identifies the user.

The session cookie links to session data, and the failover cookie contains enough session information to enable reconstruction of the session. The session cookie is intended only for the origin server, is not retained past the end of the session, and assists in the process of session maintenance. The failover cookie is intended for the failover (replicated) server, is not retained past the end of the session, and also assists in the process of session maintenance. Thus, session and failover cookies have the same P3P policy. This means that the combined worst case policy for the cookies is the session cookie policy.

Compact policy declaration

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

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

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

- **access**

  Indicates whether the site provides access to various kinds of information.
- **categories**

  Type of information stored in the cookie.
- **disputes**

  Specifies whether the full P3P policy contains some information regarding disputes over the information contained within the cookie.
- **non-identifiable**

  Signifies that either no data is collected (including Web logs), or that the organization collecting the data will make the data anonymous.
- **purpose**

  Purposes for data processing relevant to the Web.
- **recipients**

  Legal entity, or domain, beyond the service provider and its agents where data can be distributed.
- **remedies**

  Remedies in case a policy breach occurs.
- **retention**

  Type of retention policy in effect.
- **p3p-element**

  Specifies any elements to add to the P3P header in addition to the compact policy. This element can be used to supply a reference to a full XML policy.

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

Junction header preservation

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

The configuration file entry is:

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

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

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

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

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

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

Default compact policy in the P3P header

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

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

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

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

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

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

Table 7. P3P default header values

Term	Definition
NON	User has no access to information either in the cookie or linked to by the cookie.
CUR	Cookie helps provide the current service. The current service is the access to the protected Web site.

Table 7. P3P default header values (continued)

Term	Definition
OTPi	Cookie provides another service, to which the user has opted-in.
OUR	The Web site itself is the only recipient of the cookie and the information linked to by the cookie.
NOR	Neither the cookie data nor the data to which it links is retained after the user logs out or after the user session expires.
UNI	The cookie uses a unique identifier that represents the user, by using the session ID and the user name.

Configuring the P3P header

About this task

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

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

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

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

Procedure

1. Open the WebSEAL configuration file for editing. Go to the **[server]** stanza.
2. Decide if P3P headers from junctioned servers will be replaced or preserved. Set the following value:`[p3p-header] preserve-p3p-policy = {yes|no}` The default value is "no". Set this to "yes" if you want to preserve P3P headers. For more information, see “Junction header preservation” on page 107
3. Go to the **[p3p-header]** stanza. Specify the access that the user will have to the information in the cookie. Set the value for the following entry:

   ```
   [p3p-header]
   access = {none|all|nonident|contact-and-other|ident-contact|other-ident}
   ```

   . The default setting is:

   ```
   [p3p-header]
   access = none
   ```

Table 8. Supported values for the access entry

Value	Description
none	No access to identified data is given.
all	Access is given to all identified data.
nonident	Web site does not collect identified data.
contact-and-other	Access is given to identified online and physical contact information as well as to certain other identified data.
ident-contact	Access is given to identified online and physical contact information. For example, users can access things such as a postal address.
other-ident	Access is given to certain other identified data. For example, users can access things such as their online account charges.

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

```
[p3p-header] categories = {physical|online
|uniqueid|purchase|financial|computer
|navigation| interactive|demographic
|content|state|political|health
|preference|location|
government|other-category}
```

The default setting is:

```
[p3p-header]
categories = uniqueid
```

Table 9. Supported values for the categories entry

Value	Description
physical	Information that allows an individual to be contacted or located in the physical world. For example, telephone number or address.
online	Information that allows an individual to be contacted or located on the Internet.
uniqueid	Non-financial identifiers, excluding government-issued identifiers, issued for purposes of consistently identifying or recognizing the individual.
purchase	Information actively generated by the purchase of a product or service, including information about the method of payment.
financial	Information about an individual's finances including account status and activity information such as account balance, payment or overdraft history, and information about an individual's purchase or use of financial instruments including credit or debit card information.
computer	Information about the computer system that the individual is using to access the network. For example, IP number, domain name, browser type, or operating system.
navigation	Data *passively* generated by *browsing* the Web site. For example, which pages are visited, and how long users stay on each page.
interactive	Data *actively* generated from or reflecting *explicit interactions* with a service provider through its site. For example, queries to a search engine, or logs of account activity.
demographic	Data about an individual's characteristics. For example, gender, age, and income.

Table 9. Supported values for the categories entry (continued)

Value	Description
content	The words and expressions contained in the body of a communication. For example, the text of email, bulletin board postings, or chat room communications.
state	Mechanisms for maintaining a stateful session with a user or automatically recognizing users who have visited a particular site or accessed particular content previously. For example, HTTP cookies.
political	Membership in or affiliation with groups such as religious organizations, trade unions, professional associations and political parties.
health	Information about an individual's physical or mental health, sexual orientation, use or inquiry into health care services or products, and purchase of health care services or products.
preference	Data about an individual's likes and dislikes. For example, favorite color or musical tastes.
location	Information that can be used to identify an individual's current physical location and track them as their location changes. For example, Global Positioning System position data.
government	Identifiers issued by a government for purposes of consistently identifying the individual.
other-category	Other types of data not captured by the above definitions.

5. Specify whether the full P3P policy contains some information regarding disputes over the information contained within the cookie. Set the value for the following entry:

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

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

Table 10. Supported values for the disputes entry

Value	Description
yes	The full P3P policy contains information regarding disputes over the information contained within the cookie.
no	The full P3P policy *does not* contain information regarding disputes over the information contained within the cookie.

6. Specify the types of remedies in case a policy breach occurs. Set the value for the following entry:

   ```
   [p3p-header]
   remedies = {correct|money|law}
   ```

   The default setting is:

   ```
   [p3p-header]
   remedies = correct
   ```

Table 11. Supported values for the remedies entry

Value	Description
correct	Errors or wrongful actions arising in connection with the privacy policy will be remedied by the service.

Table 11. Supported values for the remedies entry (continued)

Value	Description
money	If the service provider violates its privacy policy, it will pay the individual an amount specified in the human readable privacy policy or the amount of damages.
law	Remedies for breaches of the policy statement will be determined based on the law referenced in the human readable description.

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

   ```
   [p3p-header]
   non-identifiable = {yes|no}
   ```

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

Table 12. Supported values for the non-identifiable entry

Value	Description
yes	Data that is collected identifies the user.
no	No data is collected (including Web logs), or the information collected does not identify the user.

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

   ```
   [p3p-header]
   purpose = {current|admin|develop|tailoring|pseudo-analysis|pseudo-decision|
     individual-analysis|individual-decision|contact|historical|
     telemarketing|other-purpose} [:[opt-in|opt-out|always]]
   ```

   The default setting is:

   ```
   [p3p-header]
   purpose = current
   ```

Table 13. Supported values for the purpose entry

Value	Description
current	Information can be used by the service provider to complete the activity for which it was provided.
admin	Information can be used for the technical support of the Web site and its computer system.
develop	Information can be used to enhance, evaluate, or otherwise review the site, service, product, or market.
tailoring	Information can be used to tailor or modify content or design of the site where the information is used only for a single visit to the site
pseudo-analysis	Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals *for purpose of research, analysis and reporting*.

Table 13. Supported values for the purpose entry (continued)

Value	Description
pseudo-decision	Information can be used to create or build a record of a particular individual or computer that is tied to a pseudonymous identifier, without tying identified data to the record. This profile will be used to determine the habits, interests, or other characteristics of individuals *to make a decision that directly affects that individual*.
individual-analysis	Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data *for the purpose of research, analysis and reporting*.
individual-decision	Information can be used to determine the habits, interests, or other characteristics of individuals and combine it with identified data *to make a decision that directly affects that individual*.
contact	Information can be used to contact the individual, through a communications channel other than voice telephone, for the promotion of a product or service.
historical	Information can be archived or stored for the purpose of preserving social history as governed by an existing law or policy.
telemarketing	Information can be used to contact the individual with a voice telephone call for promotion of a product or service.
other-purpose	Information can be used in other ways not captured by the above definitions.

For each value specified for **purpose**, except the value `current`, you can optionally specify the opt-in policy. The syntax consists of a colon (:) immediately following the purpose value, followed by one of the supported values for the opt-in policy. For example: `[p3p-header] purpose = telemarketing:opt-in`.

The following table lists the supported values:

Table 14. Supported values for the opt-in or opt-out policy

Value	Description
opt-in	Data can be used for this purpose only when the user affirmatively requests this use.
opt-out	Data can be used for this purpose unless the user requests that it not be used in this way.
always	Users cannot opt-in or opt-out of this use of their data. This is the default value. When the opt-in policy is not specified, the `always` policy applies.

9. Specify the recipients of the information in the cookie. Set the value for the following entry (enter recipient value on one line):

```
[p3p-header]
recipient = {ours|delivery|same|unrelated|public|other-recipient}
  [:[opt-in|opt-out|always]]
```

The default setting is:

```
[p3p-header]
recipient = ours
```

Table 15. Supported values for the recipient entry

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

For each value specified for **recipient**, excepting `ours`, you can optionally specify the opt-in policy. The syntax consists of a colon (:) immediately following the **recipient**, followed by one of the supported values for the opt-in policy. For example:`[p3p-header] recipient = delivery:opt-in`The following table lists the supported values:

Table 16. Opt-in policy values

Value	Description
opt-in	Data can be used for this purpose only when the user affirmatively requests this use.
opt-out	Data can be used for this purpose unless the user requests that it not be used in this way.
always	Users cannot opt-in or opt-out of this use of their data. This is the default value. When the opt-in policy is not specified, the `always` policy applies.

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

```
[p3p-header]
retention = {no-retention|stated-purpose|legal-requirement|business-practices|
  indefinitely}
```

The default setting is:

```
[p3p-header]
retention = no-retention
```

Table 17. Supported values for the retention entry

Value	Description
no-retention	Information is not retained for more than the brief period of time necessary to make use of it during the course of a single online interaction.
stated-purpose	Information is retained to meet the stated purpose, and is to be discarded at the earliest time possible.
legal-requirement	Information is retained to meet a stated purpose, but the retention period is longer because of a legal requirement or liability.

Table 17. Supported values for the retention entry (continued)

Value	Description
business-practices	Information is retained under a service provider's stated business practices.
indefinitely	Information is retained for an indeterminate period of time.

11. Optionally, specify a reference to a full XML compact policy file. Specify a value for the following entry:`[p3p-header] p3p-element = policyref=`*`url_to_default_location_of_full_policy`*
    This entry is present but commented out, and therefore not active, in the default WebSEAL configuration file. The default entry is the default location for the full policy on any Web site.
    `[p3p-header] # p3p-element = policyref=="/w3c/p3p.xml"`
    When **p3p-element** is not set, browsers look by default for the full policy in /w3c/p3p.xml. Note that some browsers might not refer to **p3p-element** but proceed directly to `/w3c/p3p.xml`.
    Note: Ensure that unauthenticated access is granted to `/w3c/p3p.xml`. See "P3P configuration troubleshooting."

Specifying a custom P3P compact policy

About this task

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

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

Procedure

1. Comment out or remove the predefined policy elements from the WebSEAL configuration file. For example, change the default WebSEAL entries to the following:

   ```
   [p3p-header]
   #access =
   #purpose =
   #purpose =
   #recipients =
   #retention =
   #categories =
   ```

2. Add your custom compact policy string to the **p3p-element** entry:

   ```
   [p3p-header]
   p3p-element = CP="your_series_of_compact_policy_abbreviations"
   ```

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

P3P configuration troubleshooting

Problem:

Browser cannot access the full P3P policy file.

Solution:

When the **p3p-element** stanza entry is used to specify the location of a file containing the full policy, the browser attempts to access the file. The P3P specification does not require browsers to submit cookies with the request for the full policy. Internet Explorer 6 does not submit a session cookie when accessing the full policy.

Therefore access to the full policy must be granted to unauthenticated users. When the browser receives either a login form or a 401 error, modify the permissions on the full policy to allow access by unauthenticated users.

Default port numbers

The installation uses several port numbers by default.

Table 18. Default port numbers used during Security Access Manager installation

Installation components	Fields to be completed	Default port
Security Access Manager Policy Server	Policy server port	7135
Security Access Manager Policy Server Security Access Manager Runtime Security Access Manager Runtime for Java	Policy server SSL port	7135
Security Access Manager Authorization Server	Authorization request port	7136
Security Access Manager Authorization Server	Administration request port	7137
Security Access Manager WebSEAL	WebSEAL listening port	7234
LDAP servers	Non-SSL port	389
LDAP servers	SSL port	636
Security Access Manager WebSEAL	HTTP port	80
Security Access Manager WebSEAL	HTTPS port	443

Chapter 3. Authentication

Authentication overview

This chapter discusses basic concepts of WebSEAL authentication.

Topic Index:

Definition and purpose of authentication

Authentication is the process of identifying an individual process or entity that is attempting to log in to a secure domain. Requests for protected resources by unauthenticated users always result in an authentication challenge.

- WebSEAL provides several built-in authentication methods by default.

  WebSEAL also provides the flexibility to customize the authentication mechanism.
- The result of successful authentication to WebSEAL is a Security Access Manager client identity.
- WebSEAL uses this client identity to build a credential for that user.
- The authorization service uses this credential to permit or deny access to protected resources after evaluating the authorization policies governing each object.

Information in a user request

During authentication, WebSEAL examines a user request for the following information:

- **Session key**

  A session key is a piece of data that is stored with a client and sent with every request to WebSEAL made by that client. The session key is used by WebSEAL to identify a series of requests as coming from the same client. It allows WebSEAL to avoid the overhead of performing authentication for each request. The session key is a locator index to the associated session data stored in the WebSEAL server session cache. The session key is also known as the WebSEAL session ID.
- **Authentication data**

  Authentication data is information found in the user request that identifies the user to the WebSEAL server. Examples of authentication data types include client-side certificates, passwords, and token codes.

When WebSEAL receives a user request, WebSEAL always looks for the session key first, followed by authentication data.

Client identities and credentials

The result of authentication is a client identity. WebSEAL requires the client identity to build a credential for the user. The authorization service uses this credential to permit or deny access to protected resources requested by the user.

The following process flow explains the relationship between authentication, a client identity, and a credential:

1. WebSEAL always builds an unauthenticated credential for unauthenticated users.

   An unauthenticated user can still participate in the secure domain because ACLs can contain rules that specifically govern unauthenticated users.
2. When a user requests a protected object and is required to authenticate, WebSEAL first examines the user request for authentication data.

   Authentication data includes method-specific authentication information, such as passwords and certificates, that represent physical identity properties of the user.
3. The result of successful authentication is a **client identity**.

   The client identity is a data structure that includes the user name and any extended attribute information that is to be added to the resulting credential.
4. Security Access Manager uses the client identity information to build a **credential** for that user.

   Security Access Manager matches the client identity with a registered Security Access Manager user and builds a credential appropriate to this user. This action is known as **credentials acquisition**.

   The credential is a complex structure that includes the user name, any group memberships, and any special extended security attributes associated with the user's session. The credential describes the user in a specific context and is valid only for the lifetime of that session.

   The authorization service uses this credential to permit or deny access to protected resources after evaluating the authorization policies governing each object.

   Credential acquisition can succeed only if the user has an account defined in the Security Access Manager user registry.

   If credential acquisition fails (the user is not a member of the Security Access Manager user registry), WebSEAL returns an error.

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

Authentication process flow

The following diagram illustrates the general process flow for WebSEAL authentication when an external authentication interface (EAI) is not being used:

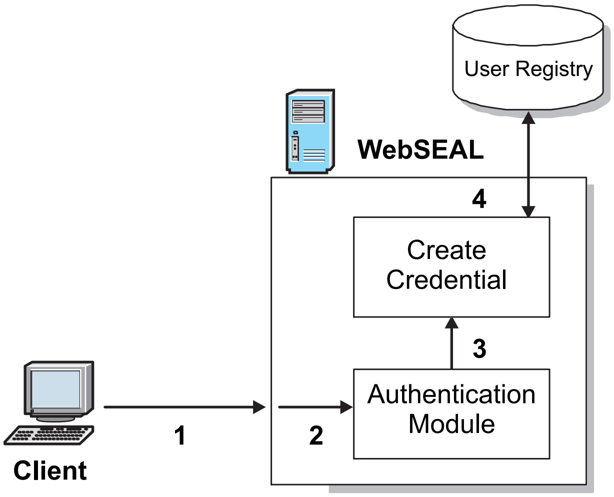


Figure 13. Authentication process flow

1. The user presents authentication information to WebSEAL (for example, password, certificate, HTTP header) during a request for a resource in the protected object space.
2. WebSEAL invokes the configured authentication module for that type of authentication information.
3. The authentication module validates the authentication information and returns an identity to WebSEAL.
4. WebSEAL uses this identity to create a credential for that user, based on data stored for that user in the user registry. This credential is used during authorization decisions for requests made by this user.

Note: The external authentication interface (EAI) supports external authentication.

Authenticated and unauthenticated access to resources

In a Security Access Manager environment, the identity of a user is proven to WebSEAL through the process of authentication. But WebSEAL can accept requests from both authenticated and unauthenticated users over HTTP and HTTPS. WebSEAL then relies on the authorization service to enforce security policy by permitting or denying access to protected resources. In general, a user can participate in the secure domain as authenticated or unauthenticated.

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

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

The credential for an authenticated user is a unique passport that describes a specific user who belongs to the Security Access Manager user registry. The authenticated user credential contains the user identity, any group memberships, and any special extended security attributes.

Request process for authenticated users

The following conditions describe the request process for authenticated users:

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

Request process for unauthenticated users

The following conditions describe the request process for unauthenticated users:

- A user makes a request for a resource protected by WebSEAL. The protection on the resource does not require that the user be authenticated. WebSEAL does not prompt the user to log in.
- WebSEAL builds an unauthenticated credential for the user.
- No entry is created in the WebSEAL session cache.
- The request proceeds, with this credential, to the protected Web object.
- The authorization service checks the permissions on the unauthenticated entry of the ACL for this object, and permits or denies the requested operation. The user can access resources that contain the correct permissions for the unauthenticated type category of user.
- Successful access to this object depends on the unauthenticated ACL entry containing at least the read (r) and traverse (T) permissions.
- If the user requires access to a resource not available to unauthenticated users, WebSEAL prompts the user to log in.
- A successful login changes the user's status to authenticated.
- If login is unsuccessful, a 403 "Forbidden" message is returned. However, the user can still continue to access other resources that are available to unauthenticated users.

Access conditions over SSL

The following conditions apply to unauthenticated users who access WebSEAL over SSL:

- The exchange of information between the unauthenticated user and WebSEAL is encrypted—just as it is with an authenticated user.
- An SSL connection between an unauthenticated user and WebSEAL requires only server-side authentication.

Forcing user login

About this task

You can force an unauthenticated user to log in by correctly setting the appropriate permissions on the unauthenticated entry in the ACL policy that protects the requested object.

The read (r) and traverse (T) permissions allow unauthenticated access to an object.

To force an unauthenticated user to log in, remove the read (r) permission from the unauthenticated entry in the ACL policy that protects the object.

The user receives a login prompt (basic authentication or forms).

Use of unauthenticated HTTPS

There are many practical business reasons for supporting unauthenticated access to WebSEAL over HTTPS:

- Some applications do not require a personal login, but require sensitive information, such as addresses and credit card numbers. Examples include online purchases of airline tickets and other merchandise.
- Some applications require that you register for an account with the business before you can proceed with further transactions. Again, sensitive information must be passed over the network.

Supported authentication methods

Although WebSEAL functions independently of the authentication process, WebSEAL uses credentials to monitor all users participating in the secure domain.

To obtain the necessary identity information for credentials acquisition, WebSEAL relies on the information gained from the authentication process.

The following table lists the authentication methods supported by WebSEAL for credentials acquisition. When WebSEAL examines a client request, it searches for authentication data in the order specified in this table.

Authentication Method	Supported Connection Type
Failover cookie	HTTP and HTTPS
LTPA cookie	HTTP and HTTPS
CDSSO ID token	HTTP and HTTPS
Client-side certificate	HTTPS
Forms authentication (username and password)	HTTP and HTTPS
Basic authentication (username and password)	HTTP and HTTPS
External Authentication Interface	HTTP and HTTPS
Open Authentication (OAuth)	HTTP and HTTPS

Authentication methods can be independently enabled and disabled for both HTTP and HTTPS transports. If no authentication methods are enabled for a particular transport, the authentication process is inactive for clients using that transport.

Authentication challenge based on user agent

WebSEAL provides a mechanism that allows the authentication challenge type to be configured based on the user agent of a client requesting a protected resource. This mechanism allows for tight integration and fine grained control over how different clients can authenticate to WebSEAL.

Each authentication type, as specified by the `auth-challenge-type` configuration entry, can be qualified with a set of rules. These rules define the user agent strings that are included or excluded for different authentication types.

For example: `auth-challenge-type = [-msie*+ms*]ba, [+mozilla*; +msie]forms; eai`

Based on the configuration example, WebSEAL:

- Does not return a basic authentication challenge to user agent strings beginning with `msie`, but does return a basic authentication challenge for agents beginning with `ms`.
- Returns a forms based authentication challenge client to user agents beginning with `mozilla` or `msie`.
- Returns an EAI authentication challenge to any user agent.

User Agent String	Authentication Challenges
msie	forms, eai
ms_office_word	ba, eai
mozilla	forms, eai
chrome	eai

Rule Syntax

Each authentication challenge type can be defined only once in the `auth-challenge-type` string. The rules must precede the authentication type enclosed in square brackets with different patterns separated by semicolons. A plus (+) or minus (-) character indicates whether that challenge type is included or excluded for that user agent string respectively.

The pattern can contain alphanumeric characters, spaces, periods, and wildcard characters, such as, question mark (?) and asterisk (*).

When WebSEAL evaluates these rules based on the user agent, the first rule with a pattern that matches the current string is applied. Any other rules that match the given authentication mechanism are ignored. WebSEAL performs these evaluations in the order in which the rules are defined.

An authentication type with no defined rule set will match any user agent string.

If you do not want the authentication type to match any user agent string, indicate the given authentication challenge by using a negative wildcard string, such as `[-*]ba`.

Note: The Authentication challenge based on the user agent functionality must not be used as a security or enforcement measure.

Authentication methods

This chapter presents information about how to configure the core set of authentication methods supported by WebSEAL.

Successful authentication results in a Security Access Manager identity that represents the user. WebSEAL uses this identity to acquire credentials for that user. Credentials allow the user to participate in the secure domain and are used by the authorization service to permit or deny access to protected resources.

Topic Index:

Authentication terminology

The following terminology is used when discussing authentication in this document:

- **method**

  An authentication method describes the overall process and strategy of an authentication type. Examples of authentication methods include, but are not limited to:

  – Username/password

  – Certificate

  Typically, but not always, authentication methods have a one to one relationship with a particular type of data used to prove a user's identity.
- **operations**

  An authentication operation describes any action that supports the authentication method. For example:

  – Performing an LDAP lookup during username and password authentication.

  – Changing a user password.

  – Verifying that a new password meets certain criteria.

  – Adding attributes to an authenticated identity.

Logout and password change operations

Security Access Manager provides authenticated users with the following resources for managing logout and password change operations:

Logging out: pkmslogout

About this task

For some authentication methods, users can use the **pkmslogout** command to log out from the current session.

The **pkmslogout** command is not appropriate for authentication methods that supply authentication data with each request, such as basic authentication, certificates, or IP address authentication. In these cases, you must close the browser to log out.

Procedure

1. Run the **pkmslogout** command to log out from the current session. For example: `https://www.example.com/pkmslogout` When this request is made,

WebSEAL returns the appropriate logout form defined in the WebSEAL configuration file: `[acnt-mgt]logout = logout.html`

2. Modify the contents of the appropriate response page (such as `logout.html`) to meet your specific requirements.

Controlling custom response pages for pkmslogout

The **pkmslogout** command 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 is appended to the **pkmslogout** URL. For example:

```
https://www.example.com/pkmslogout?filename=custom_logout_file
```

where *custom_logout_file* is the file name of the custom logout response page. This file must be located in the same `management/`*`lang`* directory that contains the default HTML response forms (such as `logout.html`) in the LMI.

The custom response page feature allows, for example, multiple logout response pages when the network architecture requires different exit screens for users logging out of distinctly different back-end systems.

You can control whether or not the appended query string is allowed to override the default response page through the **`use-filename-for-pkmslogout`** stanza entry in the **`[acnt-mgt]`** stanza of the WebSEAL configuration file.

A `no` value (default) disables the use of the query string. Any query string in a **pkmslogout** URL that specifies a custom response page is ignored. Only the default response page is used upon logout.

A `yes` value enables the use of the query string. If a query string in a **pkmslogout** URL specifies a custom response page, that custom page is used instead of the default page. For example:

```
[acnt-mgt]
use-filename-for-pkmslogout = yes
```

See also “Customized responses for old session cookies” on page 315.

Changing passwords: pkmspasswd

About this task

Users can use this command to change their login password when using basic authentication (BA) or forms authentication. For example:

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

Results

To assure maximum security when BA is used with WebSEAL, this command has the following behavior for a BA client:

1. The password is changed.
2. The client user is logged out from the current session.
3. When the client makes an additional request, the browser presents the client with a BA prompt.
4. The client must log back in to continue making requests.

This scenario applies only to a client using basic authentication.

Note: The `pkmspasswd` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

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 about what occurs and for instructions on disabling the behavior if necessary.

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

Basic authentication

Basic authentication (BA) is a standard method for providing a username and password to the authentication mechanism. BA is defined by the HTTP protocol and can be implemented over HTTP and over HTTPS.

Enabling and disabling basic authentication

About this task

The **ba-auth** stanza entry, located in the **[ba]** stanza of the WebSEAL configuration file, enables and disables the basic authentication method.

Basic authentication is enabled by default. To configure basic authentication:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[ba]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 19. Configuring basic authentication

Protocol to Support	Configuration File Entry
HTTP	`ba-auth = http`
HTTPS	`ba-auth = https`
Both HTTP and HTTPS	`ba-auth = both`
Disable basic authentication	`ba-auth = none`

For example, to support both protocols:

```
[ba]
ba-auth = both
```

3. Restart the WebSEAL server.

Setting the realm name

About this task

The realm name is the text that is displayed in the dialog box that appears when the browser prompts the user for login data. The realm name is also the name of the realm to which the user will be authenticated when the user login succeeds.

The **basic-auth-realm** stanza entry located in the **[ba]** stanza of the WebSEAL configuration file sets the realm name.

For example:

```
[ba]
basic-auth-realm = Access Manager
```

The dialog box would display (for example):

```
Enter username for Access Manager at www.ibm.com:
```

Forms authentication

Security Access Manager provides forms authentication as an alternative to the standard basic authentication mechanism. This method produces a custom HTML login form from Security Access Manager instead of the standard login prompt resulting from a basic authentication challenge.

When you use forms-based login, the browser does not cache the username and password information as it does in basic authentication.

By default, the Web Reverse Proxy is configured for forms-based authentication.

Enabling and disabling forms authentication

About this task

The **forms-auth** stanza entry, located in the **[forms]** stanza of the WebSEAL configuration file, enables and disables the forms authentication method.

Forms authentication is disabled by default. To configure forms authentication:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[forms]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 20. Configuring forms authentication

Protocol to Support	Configuration File Entry
HTTP	`forms-auth = http`
HTTPS	`forms-auth = https`
Both HTTP and HTTPS	`forms-auth = both`
Disable forms authentication (default)	`forms-auth = none`

For example, to support both protocols:

```
[forms]
forms-auth = both
```

3. Restart the WebSEAL server.

Customizing HTML response forms

Forms authentication requires you to use a custom login form called `login.html`.

About this task

You can use the LMI access the default `login.html` form.

You can customize the content and design of this form.

For detailed information on the available HTML forms that you can customize, see "Static server response pages" on page 58.

Submitting login form data directly to WebSEAL

About this task

It is possible to perform forms authentication to WebSEAL without being prompted by WebSEAL.

The following sequence describes the events that occur during a typical WebSEAL login where the user is prompted by WebSEAL with a login form:

Procedure

1. The user requests a protected resource.
2. WebSEAL caches the user's request.
3. WebSEAL returns a login form to the user.
4. The user fills in the login form fields (providing the user name and password) and clicks a submit button.
5. The submit button triggers a POST request to `/pkmslogin.form`. The request body contains the form field data.

   Note: The `pkmslogin.form` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.
6. WebSEAL authenticates the user and, upon successful authentication, follows an order of precedence for redirecting the user to one of the following three locations:
   a. The location specified by the **login-redirect-page** entry in the **[acnt-mgt]** stanza, if configured. See "Automatic redirection after authentication" on page 231.
   b. The user's originally requested resource (if known).
   c. The generic `login_success.html` page. See "Static server response pages" on page 58.

Results

Some application integration implementations might require logging in directly without making an initial request for a protected resource or being prompted by WebSEAL to login. Such a direct login can be accomplished using a POST request directly to `/pkmslogin.form`.

The following sequence describes the events that occur during a direct login:

1. The client sends a POST request to `/pkmslogin.form` with the proper form field data in the body of the request.
2. WebSEAL authenticates the user and, upon successful authentication, follows an order of precedence for redirecting the user to one of the following two locations:
   a. The location specified by the **login-redirect-page** entry in the **[acnt-mgt]** stanza, if configured.

      See “Automatic redirection after authentication” on page 231.
   b. The generic `login_success.html` page.

      See “Static server response pages” on page 58.

The format of the POST data must follow these conventions:

- The POST must be made to `/pkmslogin.form`.
- The POST request body must contain the field data for three fields:
  - **username**
  - **password**
  - **login-form-type**
- The value of **login-form-type** must be "pwd" for forms logins.
- The **content-length** header must indicate the length of the resulting request body.

Example (using **telnet**):

```
prompt> telnet webseal.example.com 80
Connected to webseal.example.com.
Escape character is '^]'.
POST /pkmslogin.form HTTP/1.1
host: webseal.webseal.com
content-length: 56

username=testuser&password=my0passwd&login-form-type=pwd
```

Client-side certificate authentication

This section contains the following topics:

Client-side certificate authentication modes

Client-side certificate authentication enables a user to use a client-side digital certificate to request an authenticated identity for use within a Security Access Manager secure domain. When authentication is successful, WebSEAL obtains a Security Access Manager identity that is used to build a credential for the user. The credential specifies the permissions and authorities to be granted to the user.

Client-side certificate authentication is disabled by default.

WebSEAL supports client-side certificate authentication in three different modes. The administrator must specify the appropriate mode at configuration time. The following sections describe each mode:

Required certificate authentication mode:

In the required certificate authentication mode, WebSEAL always requires a client-side certificate with the first HTTPS request.

When the user requests access to a resource over SSL, WebSEAL provides its server-side certificate, which allows the user to establish an SSL session. WebSEAL then asks the user for a client-side certificate.

If the user does not present a valid certificate, the SSL connection with the user is closed and client-side certificate authentication is not attempted.

Note: To be valid, the data in the certificate must not be corrupted and the certificate itself must not have been revoked by a certificate revocation list (CRL).

If a valid certificate is presented, but the authentication or authorization of the Distinguished Name (DN) in the certificate fails, the connection is established and an unauthenticated session is created. Access to protected resources is not allowed.

Optional certificate authentication mode:

In this mode, WebSEAL requests a client-side certificate with the first HTTPS request, but does not require it.

When the user requests access to a resource over SSL, WebSEAL provides its server-side certificate, which allows the user to establish an SSL session. WebSEAL then asks the user for a client-side certificate. If the user presents a client-side certificate, WebSEAL uses it to initiate a certificate-based authentication session. If the user does not present a client-side certificate, WebSEAL allows the SSL session to continue but the user remains unauthenticated to Security Access Manager.

Delayed certificate authentication mode:

In this mode, WebSEAL does not request a client-side certificate for the purpose of client-side certificate authentication until the user attempts to access a protected resource that requires certificate-based authentication.

When the user requests access to a resource over SSL, WebSEAL provides its server-side certificate, which allows the user to establish an SSL session. WebSEAL checks the security policy on the requested resource to determine if certificate authentication is required. The security policy is described in the contents of an access control list (ACL) or protected object policy (POP) that has been attached to the protected resource.

If the security policy does not require certificate authentication, WebSEAL does not request a client-side digital certificate.

If the security policy *does* require certificate authentication, WebSEAL returns a login form. The user clicks a button contained in this form to initiate the certificate exchange.

In this mode, the SSL session ID cannot be used to track user session activity, because the SSL session will be renegotiated (resulting in a new SSL session ID). All connections for the existing SSL session will be closed.

Delayed certificate authentication is used in two scenarios, based on the user's authentication status at the time that the user requests a resource that requires certificate authentication. In both scenarios, a user can have an unlimited number of exchanges with the WebSEAL server prior to establishing a need to authenticate using certificates.

The two scenarios include the following:

- **User is unauthenticated**

  In this scenario, the user remains unauthenticated because the user does not attempt to access any resources that require any authentication. When the user eventually attempts to access a resource that requires authentication because of an ACL, WebSEAL presents a certificate login form, and the user can initiate certificate transfer (by clicking the button on this form).

  WebSEAL retains the entry in the session cache for the unauthenticated user, but obtains a new SSL ID from GSKit. The old SSL session ID is discarded. When the user successfully authenticates, WebSEAL replaces the old unauthenticated user credentials from the session cache data with the new user credentials. The user is now authenticated, and is able to request access to resources that require authentication (because of an ACL).
- **User has previously authenticated using another authentication method**

  In this scenario (know as authentication strength policy or step-up authentication), the user was required to authenticate to Security Access Manager during the previous exchanges with WebSEAL. The previous authentication took place through a different authentication method, such as forms authentication.

  The user eventually attempts to access a resource that is protected by a protected object policy (POP) that requires client-side certificate authentication in order to access the resource. WebSEAL examines the current WebSEAL authentication strength policy configuration to determine the ranking of the enabled authentication methods. (The authentication strength policy ranks authentication methods in a hierarchy from weakest to strongest.)

  When certificate authentication is ranked stronger than the user's current authentication method, WebSEAL serves the user a step-up login form that contains the certificate login button. The user can click the button to initiate the certificate exchange. When the user successfully authenticates using a certificate, the user's authentication strength level is increased for the duration of the current session.

  WebSEAL retains the user's entry in the session cache, but obtains a new SSL session ID from GSKit. The old SSL session ID is discarded. WebSEAL replaces the old user credentials (which were based the user's previous authentication method) with the new user credentials.

  The authentication strength policy enables a user to move between different authentication levels during a session. Certificate authentication is one of the authentication levels that can be entered when a user needs to increase (step-up) authentication level in order to access protected object resources.

  To enable a user to move up to a certificate authentication level, administrators must modify the WebSEAL configuration file to include certificate authentication in the list of supported levels for authentication strength.

For authentication strength policy configuration instructions, see "Authentication strength policy (step-up)" on page 186.

Certificate authentication configuration task summary

All of the certificate authentication modes share a common set of configuration tasks. The delayed certificate authentication mode requires additional tasks.

To enable client-side certificate authentication in any of the supported modes, complete the following tasks:

1. "Enabling certificate authentication"
2. "Configuration of the certificate authentication mechanism" on page 132
3. "Certificate login error page" on page 134

When enabling delayed certificate authentication mode, complete the following additional tasks:

1. "Certificate login form" on page 135
2. "Disabling SSL session IDs for session tracking" on page 135
3. "Enabling and configuring the Certificate SSL ID cache" on page 135
4. "Setting the timeout for Certificate SSL ID cache" on page 136
5. "Error page for incorrect protocol" on page 136

Note: The WebSEAL server must be stopped and restarted to activate the new configuration settings.

To disable (unconfigure) client-side certificate authentication, complete the following tasks:

- "Disabling certificate authentication" on page 137
- "Disabling the Certificate SSL ID cache" on page 137

Technical notes for certificate authentication:

- "Technical notes for certificate authentication" on page 137

The WebSEAL configuration file settings for certificate authentication are summarized in the Web Reverse Proxy Stanza Reference section.

Enabling certificate authentication

About this task

Certificate authentication is disabled by default. To enable certificate authentication:

Procedure

Edit the WebSEAL configuration file. In the **[certificate]** stanza, specify a value to the **accept-client-certs** stanza entry that instructs WebSEAL how to handle client-side certificate authentication requests. The following table provides the valid values:

Table 21. Configuring certificate authentication

Configuration	Description
`accept-client-certs = optional`	Client can optionally use certificate-based authentication. WebSEAL asks clients for an X.509 certificate. If the user supplies a certificate, certificate-based authentication is used.
`accept-client-certs = required`	Client must use certificate-based authentication. WebSEAL asks clients for an X.509 certificate. If the user does not present a certificate, WebSEAL does not allow a connection.
`accept-client-certs = prompt_as_needed`	The user is not required to authenticate with a certificate at session start-up. The user can later initiate certificate authentication. This setting enables *delayed certificate authentication* mode.

For example, to prompt users for a client-side certificate only when the user encounters a resource that requires certificate authentication, enter:

```
[certificate]
accept-client-certs = prompt_as_needed
```

This setting is used when implementing an authentication strength policy (step-up) for certificate authentication.

Configuration of the certificate authentication mechanism

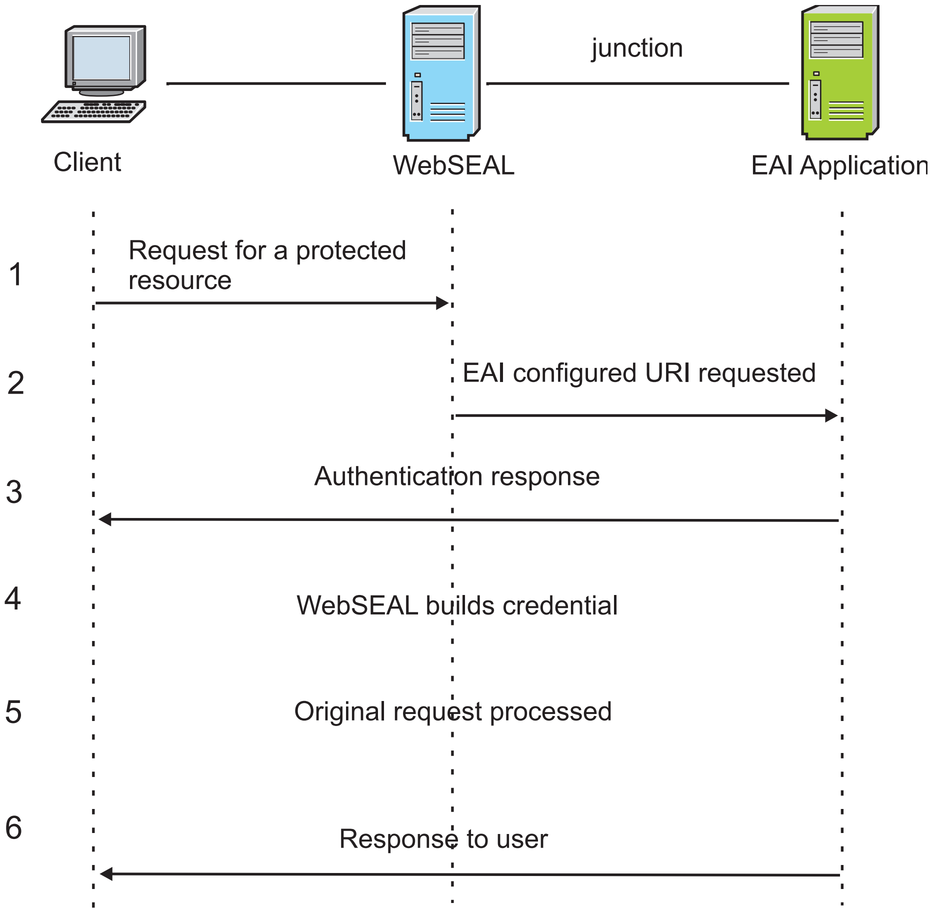
You can use the External Authentication Interface (EAI) protocol to configure a junctioned web application to handle certificate authentication on behalf of WebSEAL.

Note: For more information about EAI, see "External authentication interface" on page 260.

EAI certificate authentication:

An external application can also be used to authenticate the client certificate. This external application uses the EAI protocol to provide the authentication data that WebSEAL uses when generating the user credential.

The following diagram highlights the process flow for the authentication operation:


junction
Client
WebSEAL
EAI Application
1
Request for a protected resource
2
EAI configured URI requested
3
Authentication response
4
WebSEAL builds credential
5
Original request processed
6
Response to user

1. A request is made for a resource which is protected by WebSEAL. WebSEAL negotiates the client certificate based on the setting of the **accept-client-certs** configuration entry.
2. WebSEAL creates a sub-request, which is then sent to the configured EAI application. The URI for the EAI application is configured through the **eai-uri** configuration entry.
3. The EAI application authenticates the user (based on the client certificate data) and provides the necessary EAI headers so that WebSEAL is able to correctly construct the credential for the user. If the authentication fails, the EAI should return no authentication data, which indicates to WebSEAL that an authentication error has occurred. At this point WebSEAL will generate an authentication error page and return this to the client.
4. WebSEAL uses the authentication data to build a credential for the user.
5. Now that the user has been correctly authenticated, WebSEAL continues to process the original request.
6. The response to the original request is passed back to the client.

Configuring EAI certificate authentication:

To configure the external authentication mechanism complete the following steps.

Procedure

1. Verify that certificate authentication is enabled. See “Enabling certificate authentication” on page 131.

2. In the **[certificate]** stanza, specify the URI which is invoked to perform the authentication as the value for the **eai-uri** stanza entry. This URI must be relative to the root web space of the WebSEAL server. See the Web Reverse Proxy Stanza Reference in the IBM Knowledge Center.
3. In the **[certificate]** stanza, specify the client certificate data elements that is passed to the EAI application, as the value for the **eai-data** stanza entry. This must be of the form `eai-data = data: header_name`. Multiple pieces of client certificate data can be passed to the EAI application by including multiple eai-data configuration entries. For details, see the Web Reverse Proxy Stanza Reference in the IBM Knowledge Center.

What to do next

For more information on the EAI protocol, see the following sections:

1. "HTTP header names for authentication data" on page 265
2. "Extracting authentication data from special HTTP headers" on page 267
3. "How to generate the credential" on page 267
4. "How to write an external authentication application" on page 269

   Note: When using an external application to authenticate the client certificate, multi-step authentications are not allowed, and the external authentication application does not need to be available to unauthenticated users.
5. "External authentication interface HTTP header reference" on page 270
6. "Post-authentication redirection with external authentication interface" on page 272
7. "Session handling with external authentication interface" on page 273
8. "Authentication strength level with external authentication interface" on page 273
9. "Reauthentication with external authentication interface" on page 273
10. "Setting a client-specific session cache entry lifetime value" on page 274
11. "Setting a client-specific session cache entry inactivity timeout value" on page 276

Certificate login error page

Administrators can choose to use the default error page, customize the error message, or specify an entirely different customized error page. Typically, administrators use the default page but might customize the contents of the error message.

WebSEAL returns a default HTML response page containing an error message that is displayed when a user fails to successfully authenticate using client-side certificate authentication. Specifically, the error page is returned when the certificate is valid, but does not correspond to a Security Access Manager user.

This page is not returned when a revoked certificate is presented. Certificate revocation is handled by SSL. When a revoked certificate is presented, the SSL connection is immediately closed, resulting in a browser error page (and not the WebSEAL error page).

Administrators who choose to create a new HTML error page must edit the WebSEAL configuration file to indicate the location of the new page.

The default WebSEAL configuration file entry is:

```
[acnt-mgt]
cert-failure = certfailure.html
```

Certificate login form

WebSEAL provides an HTML page containing a login form, to be presented to users when the need for delayed certificate authentication has been identified.

Administrators can choose to use the default login form, customize the login form, or specify an entirely different customized login page. Typically, administrators use the default file but customize the contents of the form.

Administrators who choose to create a new HTML file must edit the WebSEAL configuration file to indicate the location of the new file.

The default WebSEAL configuration file entry is:

```
[acnt-mgt]
certificate-login = certlogin.html
```

Disabling SSL session IDs for session tracking

About this task

This configuration step applies only when delayed certificate authentication has been enabled.

Procedure

Disable the use of SSL session IDs to track session state. Verify the default "no" value for the **ssl-id-sessions** stanza entry in the WebSEAL configuration file:

```
[session]
ssl-id-sessions = no
```

Note: In this case, SSL IDs cannot be used to maintain user sessions because when the user is prompted for a certificate, the user's SSL ID will change. If **ssl-id-sessions** is set to "yes", WebSEAL generates an error message upon startup and shuts down.

Enabling and configuring the Certificate SSL ID cache

About this task

This configuration step applies only when delayed certificate authentication has been enabled.

To configure the cache, complete the following steps:

Procedure

1. Verify that certificate authentication is enabled.

   See “Enabling certificate authentication” on page 131.
2. Specify the maximum number of entries allowed in the cache. Edit the WebSEAL configuration file. In the **[certificate]** stanza, assign a value to **cert-cache-max-entries**. For example:

   ```
   [certificate]
   cert-cache-max-entries = 1024
   ```

The value corresponds to the maximum number of concurrent certificate authentications. The default value is one quarter of the default number of entries in the SSL ID cache. (Most SSL sessions do not require certificate logins or require certificate authentication only once for the session). The number of entries in the SSL ID cache is set in the **[ssl]** stanza. For example:

```
[ssl]
ssl-max-entries = 4096
```

Therefore, the default value for **cert-cache-max-entries** is 1024, which is one quarter of the default value for **ssl-max-entries**, which is 4096.

Note: Most user requests to WebSEAL occur over SSL connections, and all requests over SSL connections without certificates must check the cache. Keeping the cache size smaller can significantly improve performance.

Setting the timeout for Certificate SSL ID cache

About this task

This configuration step applies only when delayed certificate authentication has been enabled.

Complete the following steps:

Procedure

1. Verify that certificate authentication is enabled.
   See "Enabling certificate authentication" on page 131.
2. Edit the WebSEAL configuration file. In the **[certificate]** stanza, adjust the value of **cert-cache-timeout** as necessary. For example:

   ```
   [certificate]
   cert-cache-timeout = 120
   ```

   The value is the maximum lifetime for an entry in the cache, expressed as a number of seconds. Use the default value unless your conditions warrant modifying it. Possible reasons to modify the value include:
   - Systems with memory restrictions might need a reduced expiration time.
   - The expiration time might need to be increased if there is a significant lag between the time when the user initiates a certificate transfer and when the user actually submits the certificate.
   - Lower values clean out the cache sooner when no certificate authentications are required. Cleaning the cache frees system memory.

Error page for incorrect protocol

This configuration step applies only when delayed certificate authentication has been enabled.

WebSEAL provides a default HTML page containing an error message to be displayed when an authenticated user attempts to increase the authentication strength level to certificate authentication from an HTTP session. Users attempting to increase the authentication level to certificate authentication must use the HTTPS protocol.

Administrators can choose to use the default error page, customize the error message, or specify an entirely different customized error page. Typically, administrators use the default page but might customize the contents of the error message.

Administrators who choose to create a new HTML error page must edit the WebSEAL configuration file to indicate the location of the new page.

The default WebSEAL configuration file entry is:

```
[acnt-mgt]
cert-stepup-http = certstepuphttp.html
```

Disabling certificate authentication

About this task

To disable certificate authentication:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[certificate]** stanza, specify the following *key* = *value* pair:
   ```
   [certificate]
   accept-client-certs = never
   ```
3. Restart the WebSEAL server.

Disabling the Certificate SSL ID cache

About this task

The Certificate SSL ID cache is used only with delayed certificate authentication or authentication strength step-up to certificate authentication.

The cache is disabled automatically, based on the configuration settings for certificate authentication.

Procedure

To verify that the cache is disabled, examine the value for **accept-client-certs** in the **[certificate]** stanza. Verify that the value is one of the following:

- `required`
- `optional`
- `never`

Verify that the value is *not* `prompt_as_needed`.

Technical notes for certificate authentication

For all certificate configurations, a client-side certificate can be presented only once per browser session.

If the client-side certificate exchange fails, WebSEAL requires that the browser session be restarted before a client-side certificate can be presented again.

Token authentication

Security Access Manager supports authentication using a token passcode supplied by the client.

Token authentication concepts

This section contains the following topics:

Token authentication module:

Two-factor authentication requires users to provide two forms of identification.For example, a single factor of identification, such as a password, plus a second factor in the form of an authentication token.A simple two-factor method --- based on something the user knows plus something the user possesses -- provides a more reliable level of user authentication than reusable passwords.

Security Access Manager reverse proxies have a built-in client that is compatible with the RSA SecurID authentication server(RSA ACE/Server) and is written against the RSA Authentication API. WebSEAL provides RSA authentication client functions (RSA ACE/Agent), and is certified as RSA SecurID Ready.

By default, this built-in module for token authentication is hard-coded to map RSA SecurID token passcode data. This token authentication mechanism expects the user name used by the client to map to an existing user account in the Security Access Manager LDAP registry.

SecurID Token authentication:

The reverse proxy token authentication process uses the RSA ACE/Agent client version 8.1.2.

RSA ACE/Servers authenticate several different tokens, including software tokens and hand-held microprocessor-controlled devices. RSA SecurID authenticators (tokens) are binary programs running on a workstation, installed on a smartcard, or running as a plug-in to a Web browser. RSA SecurID authenticators can run as an application. The application displays a window into which a user enters a Personal Identification Number (PIN), and the Software Token computes the passcode. The user can then authenticate to WebSEAL by entering the passcode into a login form.

The most typical form of RSA SecurID authenticator (token) is the hand-held device. The device is usually a key fob or slim card. The token can have a PIN pad, onto which a user enters a PIN, in order to generate a passcode. When the token has no PIN pad, the passcode is created by concatenating the user's PIN and tokencode. A tokencode is changing number displayed on the key fob. The tokencode is a number generated by the RSA SecurID authenticator at one minute intervals. A user then enters the PIN and tokencode to authenticate to the RSA ACE/Server.

WebSEAL supports both RSA token modes:

- **Next tokencode mode**

  This mode is used when the user enters the correct PIN but an incorrect tokencode. Typically, the tokencode must be entered incorrectly three times in a row to send the tokencard into next tokencode mode. When the user enters the correct passcode, the tokencode is automatically changed. The user waits for the new tokencode, and then enters the passcode again.

- **New PIN mode**

  The token can be in New PIN mode when the old PIN is still assigned. The token is placed in this mode when the administrator wants to enforce a maximum password age policy. The token is also in New PIN mode when the PIN is cleared or has not been assigned. Newly assigned tokens might not yet have a PIN. A PIN can be cleared by an administrator when the user has forgotten it or suspects that it has been compromised.

RSA SecurID PINs can be created in different ways:

- User-defined
- System-generated
- User-selectable

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

WebSEAL supports the following types of user-defined PINs:

- 4-8 alphanumeric characters,non-PINPAD token
- 4-8 alphanumeric characters,password
- 5–7 numeric characters, non-PINPAD token
- 5-7 numeric characters, PINPAD token
- 5-7 numeric characters, Deny 4-digit PIN
- 5-7 numeric characters, Deny alphanumeric

WebSEAL does not support the following types of new PINs:

- System-generated, non-PINPAD token
- System-generated, PINPAD token
- User-selectable, non-PINPAD token
- User-selectable, PINPAD token

Token users cannot reset their PIN without an ACE administrator first clearing the token or putting it in new PIN mode. This means users with valid PINs cannot post to `pkmspassword.form`. Attempts to access this form return an error message.

Authentication process flow for tokens in new PIN mode:

1. A user requests a protected Web object that requires token authentication.
2. WebSEAL returns an authentication page, requesting username and passcode.
3. The user types the username and tokencode and submits the form to WebSEAL's authentication module.

   When the user has no PIN, either because the tokencard is new or the administrator reset the PIN, the tokencode is the same as the passcode. When the user has a PIN, but the tokencard is in New PIN mode, the user enters the PIN plus the tokencode.
4. The WebSEAL token authentication module sends the authentication request to the RSA ACE/Server.
5. The RSA ACE/Server processes the request as follows:
   a. If the authentication is unsuccessful, the result is returned to WebSEAL by the WebSEAL token authentication module. WebSEAL displays an error page to the client (return to step 2).

   b. If the token was not in new PIN mode, the user is authenticated. The WebSEAL token authentication module returns success to the WebSEAL server, which serves the requested protected Web object. (End of authentication workflow).
   c. If the token is in new PIN mode, the RSA ACE/Server returns the NEW_PIN error code to the WebSEAL token authentication module.
6. WebSEAL presents the password expired form to the user.
7. The user enters tokencode or passcode and the new PIN and posts it to WebSEAL.
8. WebSEAL checks to see if a password strength module is configured.
   a. If no password strength module is configured, WebSEAL continues to step 9.
   b. If a password strength module is configured, WebSEAL checks the new PIN. If the PIN is valid, WebSEAL continues to step 9. If the PIN is not valid, WebSEAL returns to step 6.
9. The WebSEAL authentication module sends the tokencode and new PIN to the RSA ACE/Server.
10. The RSA ACE/Server returns a response code.
11. If the PIN set call to the RSA ACE/Server is successful, WebSEAL returns the originally requested protected Web object to the client. If the PIN set call fails, authentication workflow returns to step 6.

Token authentication configuration task summary

To configure token authentication, the instructions in the following sections must be completed:

- “Enabling token authentication”

When a password strength module is used with token authentication, the instructions in the following section must be completed:

- “Password strength” on page 226

To unconfigure token authentication, complete the instructions in the following section:

- “Disabling token authentication” on page 141

To submit login form data directly to WebSEAL:

- “Submitting login form data directly to WebSEAL” on page 141

Enabling token authentication

About this task

The **token-auth** stanza entry, located in the **[token]** stanza of the WebSEAL configuration file, enables and disables the token authentication method.

Token authentication is disabled by default. To configure token authentication:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[token]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 22. Configuring token authentication

Protocol to Support	Configuration File Entry
HTTP	`token-auth = http`
HTTPS	`token-auth = https`
Both HTTP and HTTPS	`token-auth = both`
Disable token authentication (default)	`token-auth = none`

For example, to support both protocols:

```
[token]
token-auth = both
```

3. Restart the WebSEAL server.

Disabling token authentication

About this task

To disable token authentication:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. Set **token-auth** to "none":`[token]`
   `token-auth = none`
3. Restart the WebSEAL server.

Results

Token authentication is disabled by default.

Submitting login form data directly to WebSEAL

About this task

It is possible to perform token (or forms) authentication to WebSEAL without being prompted by WebSEAL.

The following sequence describes the events that occur during a typical WebSEAL login where the user is prompted by WebSEAL with a login form.

Procedure

1. The user requests a protected resource.
2. WebSEAL caches the user's request.
3. WebSEAL returns a login form to the user.
4. The user fills in the login form fields (providing the user name and passcode) and clicks a submit button.
5. The submit button triggers a POST request to `/pkmslogin.form`. The request body contains the form field data.

   Note: The `pkmslogin.form` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

6. WebSEAL authenticates the user and, upon successful authentication, follows an order of precedence for redirecting the user to one of the following three locations:
   a. The location specified by the **login-redirect-page** entry in the **[acnt-mgt]** stanza, if configured.

      See "Automatic redirection after authentication" on page 231.
   b. The user's originally requested resource (if known).
   c. The generic `login_success.html` page.

      See "Static server response pages" on page 58.

Results

Some application integration implementations might require logging in directly without making an initial request for a protected resource or being prompted by WebSEAL to login. Such a direct login can be accomplished using a POST request directly to `/pkmslogin.form`.

The following sequence describes the events that occur during a direct login:

1. The client sends a POST request to `/pkmslogin.form` with the proper form field data in the body of the request.
2. WebSEAL authenticates the user and, upon successful authentication, follows an order of precedence for redirecting the user to one of the following two locations:
   a. The location specified by the **login-redirect-page** entry in the **[acnt-mgt]** stanza, if configured.

      See "Automatic redirection after authentication" on page 231.
   b. The generic `login_success.html` page.

      See "Static server response pages" on page 58.

The format of the POST data must follow these conventions:

- The POST must be made to `/pkmslogin.form`.
- The POST request body must contain the field data for three fields:
  – **username**
  – **password**
  – **login-form-type**
- The value of **login-form-type** must be "token" for token logins.
- The **content-length** header must indicate the length of the resulting request body.

Example (using **telnet**):

```
prompt> telnet webseal.example.com 80
Connected to webseal.example.com.
Escape character is '^]'.
POST /pkmslogin.form HTTP/1.1
host: webseal.webseal.com
content-length: 58

username=testuser&password=123456789&login-form-type=token
```

Kerberos authentication through an External Authentication Interface (EAI)

The appliance internally supports Kerberos authentication for use with Windows clients to achieve Windows desktop single sign-on. Alternatively, you can configure a junctioned web server to handle Kerberos authentication on behalf of the appliance.

The junctioned web server acts as an External Authentication Interface (EAI) application and completes the Kerberos authentication. This web server, known as the *Kerberos Authenticator*, then passes the authenticated user identity back to the appliance in an EAI header. For more information about EAI, see "External authentication interface" on page 260.

Topic index:

Configuring Kerberos authentication with an external Kerberos Authenticator

You can achieve Windows desktop single signon by configuring a Kerberos Authenticator to authenticate clients on behalf of the appliance.

About this task

You can configure a junctioned web server to complete the actual authentication and return the authenticated identity to the appliance.

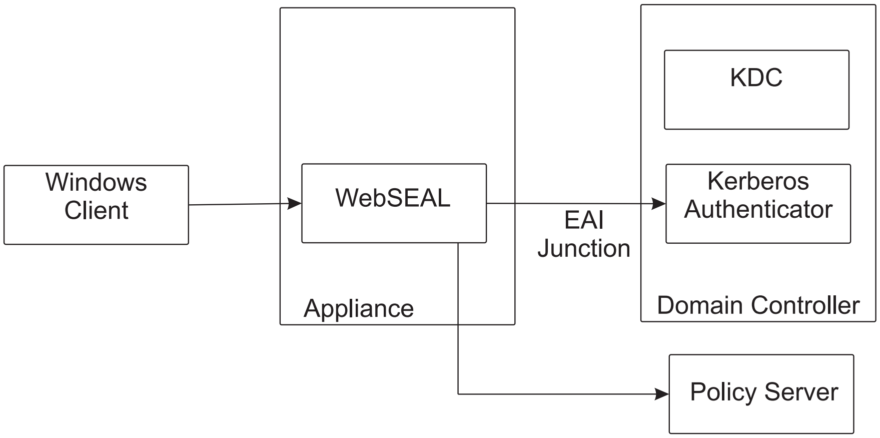


Figure 14. External Kerberos authentication

Complete the following steps to configure an external Kerberos Authenticator to do the authentication on behalf of the appliance. An example is provided for each step. Collectively, these examples describe one possible configuration that supports Windows desktop single signon.

Procedure

1. Install the Policy Server and configure its user registry. For example, Active Directory.
2. Configure a web server that supports Kerberos Authentication. This web server is the Kerberos Authenticator. For example, install WebSEAL on the domain controller and configure Kerberos authentication.

   See "Windows desktop single sign-on" on page 144.

3. Configure the External Authentication Interface (EAI) application on the Kerberos Authenticator. For example, create a simple Common Gateway Interface (CGI) to act as the EAI. This CGI creates an EAI response, setting the **am-eai-user-id** header field as the name of the authenticated user.

   You can now verify the configuration of the Kerberos Authenticator. Add a Windows client to the domain. Verify that Windows desktop single signon occurs when you access the WebSEAL server from this client.

   You can install a network protocol analyzer, such as Wireshark, on the domain controller to monitor and validate the network traffic.

4. Configure WebSEAL on the appliance to use the external Kerberos Authenticator for authentication. For example, follow these steps:

   a. Create a junction to the Kerberos authenticator.

   b. Configure the CGI script as an EAI application.

   c. Set the **strip-www-authenticate-headers** configuration entry to `no`.

      If the **strip-www-authenticate-headers** configuration entry is set to `yes`, WebSEAL removes the **Negotiate www-authenticate** and **NTLM www-authenticate** headers from junctioned server responses. Therefore, you must set the value to `no` to keep these **www-authenticate** headers in the junctioned server responses.

      For more information about this configuration entry, see the Reference information in the IBM Knowledge Center.

   You can now verify that Windows desktop single signon is available on the appliance.

   Send a request from the Windows client, through the WebSEAL server on the appliance, to the EAI application. Single signon occurs. That is, the user can access the WebSEAL server on the appliance as an authenticated user. Again, you can use a network protocol analyzer to monitor and validate the network traffic.

Limitations

When using an external Kerberos authenticator, the appliance can support Kerberos authentication only. It cannot support NTLM authentication.

The Windows NTLM implementation requires that the same connection is used during the multiple stages of the authentication process. WebSEAL cannot always provide the same connection for use throughout the authentication process.

Therefore, you cannot use a server that supports only NTLM authentication as the Kerberos Authenticator. You must use a server that supports Kerberos authentication as the Kerberos Authenticator.

Note: Microsoft Internet Information Services (IIS) uses NTLM authentication by default.

Windows desktop single sign-on

This chapter discusses the concepts and configuration steps required to implement an authentication solution that enables browser clients to access WebSEAL servers without multiple logins.

Windows desktop single sign-on concepts

This section discusses the following topics:

SPNEGO protocol and Kerberos authentication:

Microsoft provides an authentication solution so that Windows clients can use Microsoft Internet Explorer to access resources on Microsoft Internet Information Servers (IIS) without reauthenticating.

This single sign-on solution relies on proprietary Microsoft HTTP authentication mechanisms. IBM Security Access Manager WebSEAL provides an equivalent authentication solution that enables Internet Explorer clients to access WebSEAL servers without reauthenticating.

Users with an Internet Explorer browser can access resources that are protected by Security Access Manager without reentering their user name and password. The user must log in only once to the Windows domain, as is typically done when a user logs in to Windows on a desktop workstation.

WebSEAL supplies an implementation of the same HTTP authentication method that is used by Microsoft. This implementation involves two components:

- Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)
- Kerberos authentication

The SPNEGO protocol enables WebSEAL to negotiate with the browser to establish the authentication mechanism to use. The browser supplies Kerberos authentication information. WebSEAL knows how to use the user's Kerberos authentication information when it processes a user request to access resources protected by Security Access Manager.

On WebSEAL, this implementation is called **Windows desktop single sign-on**.

Deployment of this single sign-on solution requires enabling and configuring the SPNEGO protocol on the WebSEAL server. In addition, the WebSEAL server must have connectivity to an Active Directory domain controller. The Active Directory domain controller must act as a Kerberos Key Distribution Center (KDC). WebSEAL servers must use the Active Directory domain controller as their Kerberos KDC.

The WebSEAL configuration steps vary depending on the operating system platform and type of Security Access Manager user registry.

Note: Use of SPNEGO requires that a time synchronization service is deployed across the Active Directory server, the WebSEAL server, and any clients (browsers) that use SPNEGO to authenticate.

WebSEAL and IIS handle session management differently. IIS maintains session state with clients by using the SPNEGO protocol to reauthenticate each new TCP connection. SPNEGO and Kerberos are both designed for secure authentication over insecure networks. In other words, they are supposed to provide for secure authentication even when using an insecure transport such as HTTP.

The IIS method of maintaining session state can potentially have an adverse effect on performance. WebSEAL avoids this problem by using different session state methods. The WebSEAL session state methods are based on a security model that expects WebSEAL to be deployed either over a secure network or using a secure transport such as SSL. WebSEAL optimizes performance by maintaining state using SSL session IDs or HTTP cookies. Also, WebSEAL provides a scalable, secure environment by supporting junctions between WebSEAL and back-end servers. Therefore, single sign-on solutions using SPNEGO to WebSEAL must be deployed only over a secure network or over a secure transport such as SSL.

User registry and platform support for SPNEGO:

WebSEAL SPNEGO support provides single sign-on from Internet Explorer on Windows client workstations that are configured into an Active Directory domain.

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

Kerberos authentication is only possible with browsers and platforms that support the SPNEGO protocol. Client computers and browsers must be properly configured to enable Kerberos authentication. SPNEGO single sign-on to WebSEAL functions successfully with Chrome, Edge, Firefox, and Internet Explorer browsers. For additional configuration instructions, see the appropriate browser documentation or contact the support organization for the browser software.

SPNEGO compatibility with other authentication methods:

WebSEAL support for Kerberos authentication is compatible with several WebSEAL authentication methods.

The following WebSEAL authentication methods are compatible:

- Basic authentication
- Forms authentication
- HTTP header authentication
- LTPA authentication
- Failover authentication

  The failover cookie failover mechanism supports SPNEGO authenticated users.
- Cross domain single sign-on
- SSL certificate authentication
- External authentication interface

When SPNEGO is configured with another authentication method, WebSEAL simultaneously sends both an SPNEGO challenge and an HTML form login to the browser. Browsers that support SPNEGO respond with Kerberos authentication. Browsers that do not support SPNEGO display the login form.

Compatibility between Kerberos authentication and WebSEAL e-community single sign-on is limited. A WebSEAL server can be an e-community master authentication server (MAS) and support SPNEGO. However, a WebSEAL server cannot be an e-community subordinate and also support SPNEGO.

WebSEAL authentication strength policy (step-up authentication) from Kerberos authentication to other authentication methods is supported.

When Kerberos authentication is enabled, only the following methods of maintaining session state are supported:

- SSL session IDs
- HTTP cookies

- HTTP header session keys

Kerberos authentication is compatible with the automatic tag-value retrieval support provided by the Security Access Manager entitlements service. Therefore, it is possible to add extended attributes to a user's credential after the user has authenticated with SPNEGO.

Mapping of user names from multi-domain Active Directory registries:

By default, when WebSEAL is mapping user names to certain user registries, it truncates the user names that are provided by Kerberos authentication. Using truncated user names can cause name resolution conflicts if the same name is in multiple domains. However, you can control whether WebSEAL truncates the user name.

User name formats from differing user registries:

WebSEAL maps the user name that the Kerberos authentication process provides to the Security Access Manager user registry. This mapping process depends on the type of user registry.

Kerberos authentication provides Security Access Manager with a user name in the following form:

```
user@domain.com
```

When multiple-domain Active Directory is used as the Security Access Manager user registry, the user name listed in the Active Directory registry uses the same format as the user name provided by the Kerberos authentication process.

If the Security Access Manager user registry is not Active Directory, WebSEAL, by default, truncates the user name that is provided by Kerberos authentication. WebSEAL maps this truncated user name to the user registry.

For example, the following format is received from Kerberos authentication:

```
user@domain.com
```

WebSEAL truncates this name by removing the domain designation and leaving only the short-name:

```
user
```

WebSEAL creates a credential for that user based on the short-name.

This mapping from the full Active Directory user name to the short-name of the user is not always appropriate and can cause conflicts when resolving user names. For example, consider the scenario of two users with the same short-name in different Active Directory domains. When WebSEAL truncates the user names for each of these users, the users are mapped incorrectly to the same Security Access Manager user. When truncation does not occur, the users are correctly mapped to unique Security Access Manager users (for example `user@domainA.com` and `user@domainB.com`).

Setup for user name truncation handling:

You can use the **Use Domain Qualified Name** check box in the **Authentication** tab of the Reverse Proxy management page to control whether or not WebSEAL truncates the user name received from Kerberos authentication.

This configuration option is appropriate when WebSEAL receives user names from Kerberos authentication (in a multiple-domain Active directory environment) that must be mapped to a default Security Access Manager user registry that is not Active Directory. A setting of `yes` prevents WebSEAL from removing the domain from the SPNEGO user name format.

In this case, WebSEAL uses the fully-qualified user name to build the credential (in the non-Active Directory registry) for the user.

In the following example, Kerberos authentication provides the following user ID:

```
user@example.com
```

If `use-domain-qualified-name = no`, the Security Access Manager user ID becomes:

```
user
```

If `use-domain-qualified-name = yes`, the Security Access Manager user ID becomes:

```
user@example.com
```

The **`use-domain-qualified-name`** stanza entry has no effect if multiple-domain Active Directory is used as the Security Access Manager user registry. In this case, the domain name is always included as part of the Security Access Manager user name.

Note: You can use the **Authentication** tab in the LMI to configure the main settings for Kerberos Authentication on the appliance. For more information, see the Kerberos Authentication details in the "Configuration entry and file management" section of the Administering topics in the Knowledge Center.

Multiple Active Directory domain support:

Active Directory uses domains and forests to represent the logical structure of the directory hierarchy. Domains are used to manage the various populations of users, computers, and network resources in your enterprise. The forest represents the security boundary for Active Directory.

Kerberos authentication for users from multiple Active Directory domains is supported by Security Access Manager only if an appropriate trust relationship between the domains is established. This trust exists automatically for domains that are part of the same Active Directory forest. For Kerberos authentication to work across multiple forests, a forest trust relationship must be established.

For details on establishing a trust relationship between multiple Active Directory domains, refer to the appropriate Active Directory documentation from Microsoft.

Kerberos authentication limitations:

Some WebSEAL features are not supported with Kerberos authentication.

If you are using Kerberos authentication, the following limitations apply:

- POP or session-timer-based reauthentication of Kerberos authenticated clients is not supported.
- Using **`pkmspasswd`** to change a password is not supported.

- Clients who are currently authenticated with SPNEGO cannot log out of WebSEAL. Clients must log out from the workstation. Clients that access the WebSEAL **pkms** command pages, except switch user, receive the PKMS help page.
- Reauthentication when the inactive session timer expires is not supported for SPNEGO clients.

  The user cache entry is deleted. Information in the header received from the SPNEGO client is used to reauthenticate. The client is not required to log in again, but the client receives a new session cache entry.
- WebSEAL does not support reauthentication when a user accesses an object with a reauthentication policy attached.

  In this case, access is denied and the user receives a message that states reauthentication is required.
- Microsoft NT LAN Manager (NTLM) authentication is not supported.
- Using alternate user principal name (UPN) format is not supported.

  The default format for the userPrincipalName attribute in Active Directory is *user_shortname@domain*, where *domain* is the Active Directory domain in which the user was created.

  For example, a user that is created in the Active Directory domain `child.domain.com` might have a UPN of `user@child.domain.com`. However, a user can be created with an alternate UPN format, also called e-mail format, where the domain need not be the actual Active Directory domain name. For example, the user `user@domain.com` can be created in the `child.domain.com` Active Directory domain.

  Security Access Manager Kerberos authentication only supports the default format of the **`userPrincipalName`** attribute as the Active Directory user identity. The use of the alternate UPN format is not supported if you are using Kerberos authentication.

Configure Windows desktop single sign-on

There are many configuration tasks that you must complete to implement Windows desktop single sign-on using Kerberos authentication for WebSEAL on the appliance.

To configure WebSEAL on the appliance for SPNEGO authentication, complete each of the following tasks:

For troubleshooting information, see “Troubleshooting for Windows desktop single sign-on” on page 157.

Configuring the embedded Kerberos client:

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

About this task

To complete this configuration, modify the Kerberos configuration in the LMI.

Note: For more details, see "Managing Kerberos configuration" in the Administering section of the IBM Knowledge Center.

Procedure

1. Open the Kerberos Configuration management page in the LMI. From the top menu, select **Secure Web Settings** > **Global Settings** > **Kerberos Configuration**.
2. Open the **Defaults** tab to configure the **libdefaults** section of the corresponding Kerberos configuration file.
3. Set the following entries:

   default_realm
   Specifies the Active Directory domain name in uppercase.

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

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

   For example:

   ```
   [libdefaults]
   default_realm = EXAMPLE.COM
   default_tkt_enctypes = rc4-hmac
   default_tgs_enctypes = rc4-hmac
   ```
4. Open the **Realms** tab to configure the **realms** section of the corresponding Kerberos configuration file.
5. Create a realm entry for the default realm that was specified as the `default_realm` in the **Defaults** tab. 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`.

   Note: Multiple comma-separated KDCs can be added in a single line.

   default_domain
   Specify the local DNS domain of the server that runs WebSEAL. This value is the domain that client browsers use to access WebSEAL 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
   }
   ```
6. Open the **Domains** tab to configure the **domain_realm** section of the corresponding Kerberos configuration file.
7. Configure entries to map local domains and domain names to the Active Directory Kerberos realm. Use the WebSEAL host names specified in step 5.

   For example:

   ```
   [domain_realm]
   www.examplefancy.com = EXAMPLE.COM
    .examplefancy.com = EXAMPLE.COM
   ```
8. Optional: To use SPNEGO as the authentication type at a junction level, create a **[server:*jct_id*]** stanza in the WebSEAL configuration file and add an **auth-challenge-type** entry with a value of `spnego`.

where:

jct_id 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
```

Creating an identity for WebSEAL in an Active Directory domain:

To participate in a Kerberos exchange with a browser, a WebSEAL server needs an identity in the Active Directory Kerberos domain.

About this task

Creating the identity, and copying the keytab to the WebSEAL server on UNIX, provides a similar function as joining a Windows system to an Active Directory domain. The browser can then obtain a Kerberos ticket from the Active Directory domain controller and use the ticket to access the WebSEAL server.

These instructions assume that the user name is the shortened DNS name that clients use to contact the WebSEAL server. For example, if clients contact the WebSEAL server at `https://diamond.example.com`, the WebSEAL server principal in Active Directory would have a user name of **diamond**. However, any identifying name can be used.

Procedure

1. See the appropriate Microsoft documentation for instructions on how to add a WebSEAL server host identity into an Active Directory domain.

   Follow these conditions:

   - Match the user name with the host name that clients use to contact the WebSEAL server. Do not use the full domain name. For example, for the website, `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.
   - Configure DNS properly for the host name that clients use to contact the WebSEAL server.
   - To confirm the configuration is correct, run forward and reverse **nslookup** for that host name on each of the following systems: the client, the WebSEAL server, and the Active Directory domain controller.
   - The account must not be set to use DES ciphers.
   - If you intend to use AES encryption for tickets and keys that are placed in the keytab, modify the following user account properties to enable it:
     - `This account supports Kerberos AES 128 bit encryption`
     - `This account supports Kerberos AES 256 bit encryption`
   - Multiple WebSEAL instances are supported by SPNEGO when each WebSEAL server has a unique IP address and host name. Multiple instances are not supported when the instances listen on different ports but share IP addresses.
2. Optional: This step is only relevant if you use multiple WebSEAL instances or virtual host junctions. To configure SPNEGO for multiple WebSEAL servers on

the same system, you must create a separate user in Active Directory for each instance. Similarly, if you use virtual host junctions, create a separate user for each virtual host junction.

For example, if the WebSEAL server is serving requests for the host names `www.example.com`, `sales.example.com`, and `eng.example.com`, you must create three users in Active Directory, one for each DNS name.

Mapping a Kerberos principal to an Active Directory user:

You must map a Kerberos principal to the Active Directory user that represents the WebSEAL instance.

Before you begin

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

About this task

On a Windows client system, the browser makes a URL request, so the Kerberos client makes a request to the KDC in the following format:

```
HTTP/hostname_of_web_site@ACTIVE_DIRECTORY_DOMAIN_NAME
```

Note: The domain name is in uppercase.

For WebSEAL on UNIX systems, create a keytab file for verifying Kerberos tickets in addition to creating the user.

Procedure

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

```
ktpass -princ hostname_of_web_site@ACTIVE_DIRECTORY_DOMAIN_NAME
 {-pass your_password | +rndPass} -mapuser WebSEAL_server_instance
-out full_path_to_keytab_file -mapOp set -crypto cipher -ptype
KRB5_NT_PRINCIPAL
```

where:

hostname_of_web_site
Specifies the name of the website.

ACTIVE_DIRECTORY_DOMAIN_NAME
Specifies the Active Directory domain name in all uppercase.

The domain name must map to the Active Directory user that represents the WebSEAL instance, as created in “Creating an identity for WebSEAL in an Active Directory domain” on page 151.

your_password
Specifies a password to set when the `-pass` parameter is used. The specified password 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 “Verifying the authentication of the web server principal” on page 154 to test your Kerberos configuration. You need this password to test authentication from the Appliance to the Active Directory Key Distribution Center.

WebSEAL_server_instance
Specifies the WebSEAL server instance user identity. The user identity is the Active Directory user that was created in "Creating an identity for WebSEAL in an Active Directory domain" on page 151.

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 have DES ciphers disabled by default.

For Windows Server 2003, this restriction leaves RC4-HMAC-NT for the `-crypto` cipher.

For Windows Server 2008 R2, this restriction leaves the following values for `-crypto` cipher:

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

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

AES128-SHA1
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.

The following example command, entered as one line, uses the values that are listed in the included table:

```
ktpass -princ HTTP/diamond.example.com@EXAMPLE.COM +rndPass
-mapuser diamond -out C:\diamond_HTTP.keytab -mapOp set
-crypto RC4-HMAC-NT -ptype KRB5_NT_PRINCIPAL
```

Configuration setting	Value
WebSEAL host system	`diamond.example.com`
User identity for the WebSEAL instance	`diamond`
Active Directory domain	`EXAMPLE.COM`
password	A random value. The `+rndPass` argument prevents anyone from knowing the password of the **mapuser**. If the diamond user password must be known for other purposes, use the `-pass` option and provide a secure password.
Keytab file	`C:\diamond_HTTP.keytab`
`-mapOp`: Flush other Service Principal Names that are associated with the user and set the one provided by `-princ`	`set`
Cipher of the key to store in the keytab file	RC4-HMAC-NT

2. Optional: To configure SPNEGO for virtual host junctions, create a separate keytab file for each virtual host. Repeat the **ktpass** command for each principal in Active Directory.

   For example, see the following three commands, each one entered as one line:

   ```
   ktpass -princ HTTP/www.example.com@EXAMPLE.COM
   -pass mypassw0rd -mapuser www
   -out www_HTTP.keytab -mapOp set
   -crypto RC4-HMAC-NT -ptype KRB5_NT_PRINCIPAL

   ktpass -princ HTTP/sales.example.com@EXAMPLE.COM
   -pass mypassw0rd -mapuser sales
   -out sales_HTTP.keytab -mapOp set
   -crypto RC4-HMAC-NT -ptype KRB5_NT_PRINCIPAL

   ktpass -princ HTTP/eng.example.com@EXAMPLE.COM
   -pass mypassw0rd -mapuser eng
   -out eng_HTTP.keytab -mapOp set
   -crypto RC4-HMAC-NT -ptype KRB5_NT_PRINCIPAL
   ```
3. Import the keytab files on to the appliance. For more information, see "Managing keytab files" in the Administering topics in the Knowledge Center.
4. As a best security practice, delete the keytab files from the Windows client system.
5. Optional: To configure SPNEGO for multiple virtual hosts, combine all keytabs into a single file using the **Combine** function in the **Keyfiles** tab on the Kerberos Configuration management page in the LMI.

   Note:
   - You can remove redundant keys from the list of key files as they are no longer required.
   - 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.
6. Repeat these steps for each WebSEAL instance.

Verifying the authentication of the web server principal:

Verify the authentication of the web server principal to the KDC.

Before you begin

Skip this task if you used the `+rndPass` option when you generated the keytab files.

Procedure

1. To open the Kerberos Configuration management page in the LMI, select **Secure Web Settings** > **Global Settings** > **Kerberos Configuration**.
2. Click **Test** on any of the tabs other than **Keyfiles** on the Kerberos Configuration management page. This test checks whether the Kerberos principal for the WebSEAL instance can authenticate to the KDC.
3. Check the output message to determine the success of the authentication.
   - If the authentication is successful, a success message is displayed.
   - If the authentication fails, an error message is displayed.

Verifying WebSEAL authentication with the keytab file:

Ensure that WebSEAL can use the generated keytab file to authenticate to the KDC.

About this task

Verify that the WebSEAL server can authenticate with the keytab file that you created in "Mapping a Kerberos principal to an Active Directory user" on page 152.

Procedure

1. To open the Kerberos Configuration management page in the LMI, select **Secure Web Settings** > **Global Settings** > **Kerberos Configuration**.
2. Click **Test** on the **Keyfiles** tab on the Kerberos Configuration management page to verify that the WebSEAL instance can use the service name to authenticate to the KDC.
3. Check the output message to determine the success of the authentication.
   - If the authentication is successful, a success message is displayed.
   - If the authentication fails, an error message is displayed.

Adding service name and keytab file entries:

You must configure the Kerberos service name and the name of the keytab file.

About this task

Modify the Kerberos authentication settings in the LMI to add the Kerberos service name and the name of the keytab file.

Procedure

1. From the top menu of the LMI, select **Secure Web Settings** > **Manage** > **Reverse Proxy**. The Reverse Proxy management page displays.
2. Select the reverse proxy instance that you want to manage.
3. Select **Edit**.
4. Select the **Authentication** tab.
5. In the Kerberos Authentication settings, add the Kerberos service name to the list of **Kerberos Service Names**. This **Kerberos Service Names** field in the LMI adds an **spnego-krb-service-name** stanza entry to the **[spnego]** stanza in the WebSEAL configuration file.

   There is one **spnego-krb-service-name** stanza entry for the standard WebSEAL server junctions and one for each virtual host junction.

   The first entry in the list is used for authentication to standard junctions.

   For example:

   ```
   # The principal HTTP@www.example.com will be used for authentication
   # to standard junctions.
   spnego-krb-service-name = HTTP@www.example.com

   # The principal HTTP@sales.example.com will be used for the virtual
   # host junction sales.example.com
   spnego-krb-service-name = HTTP@sales.example.com

   # The principal HTTP@eng.example.com will be used for the virtual
   # host junction eng.example.com
   spnego-krb-service-name = HTTP@eng.example.com
   ```

   Note: These entries are set by updating the Kerberos settings on the Authentication tab for a Reverse Proxy instance.

6. In the Kerberos Authentication settings, add the name of the keytab file in the **Keytab File** field.

   The keytab file must contain an entry for each principal listed. To configure SPNEGO authentication for virtual host junctions, specify the combined keytab file. For example, `spnego.keytab`.

   Note: This **Keytab File** field in the LMI sets the value of the **`spnego-krb-keytab-file`** entry in the **`[spnego]`** stanza in the WebSEAL configuration file.

Enabling SPNEGO for WebSEAL:

You must configure WebSEAL to enable SPNEGO.

Procedure

1. From the top menu of the LMI, select **Secure Web Settings** > **Manage** > **Reverse Proxy**. The Reverse Proxy management page displays.
2. Select the reverse proxy instance that you want to manage.
3. Select **Edit**.
4. Select the **Authentication** tab.
5. In the Kerberos Authentication settings, select `HTTPS` for the **Transport** field to enable SPNEGO over SSL.

   Note: This **Transport** field in the LMI sets the value of the **`spnego-auth`** entry in the **`[spnego]`** stanza in the WebSEAL configuration file.
6. Optional: Edit the **`[spnego]`** stanza in the WebSEAL configuration file to enable adding the security identifier (SID) of the user as an extended attribute to the credential during authentication.

   ```
   [spnego]
   spnego-sid-attr-name = attribute_name
   ```

   where *attribute_name* defines the name of the attribute that stores the SID.
7. Click **Save**.
8. Deploy the updates as described in “Deploying WebSEAL updates in the LMI” on page 17.
9. Restart the reverse proxy instance as described in “WebSEAL instance management” on page 16.

Removing cached tokens from the Windows client:

For SPNEGO to work correctly, you must remove any cached Kerberos tokens from the Windows client. When a client contains cached Active Directory credentials, SPNEGO might not work correctly on the client until it obtains the new credentials.

Procedure

1. Run `klist` on the Windows desktop client to determine the presence of old Kerberos tokens.
2. If old tokens are present, take one of the following actions:
   - Run `klist purge` to delete each token.
   - Restart the Windows desktop.

Configuring the Internet Explorer client:

If you use Internet Explorer, you must configure the browser to use the SPNEGO protocol.

About this task

You must configure the Internet Explorer client to use the SPNEGO protocol to negotiate authentication mechanisms. Consult the Microsoft Internet Explorer documentation for configuration instructions.

Keep the following points in mind during the configuration:

- The Internet Explorer browser must recognize the WebSEAL server as an *Intranet* site. Otherwise, the Internet Explorer client does not automatically send an SPNEGO authorization token for the logged in user to the WebSEAL server. The Internet Explorer client must add the WebSEAL server to the *Intranet Sites* list.

  If required, you can add the WebSEAL server to the *Trusted Sites* instead of *Intranet Sites*.

  By default, **Automatic logon** occurs only in the *Intranet* zone. Therefore, you must create a **Custom Security Level** for *Trusted Sites* that sets the **User Authentication** > **Logon** setting to **Automatic logon**. You must provide the current user name and password for this configuration.
- You must configure Internet Explorer 6 to enable single sign-on. Use the menu item for **Internet Options...** and select the **Advanced** tab.
- The Windows client must use the correct DNS name to access the WebSEAL server. When an incorrect DNS name is used, Internet Explorer might attempt to use NT LAN Manager (NTLM) protocol to contact WebSEAL. WebSEAL does not support NTLM.

Troubleshooting for Windows desktop single sign-on:

See the section describing solutions to common Web security SPNEGO problems in the Troubleshooting section of the IBM Knowledge Center.

Configuration notes for a load balancer environment

The following notes address the configuration of Windows desktop single sign-on using SPNEGO in an environment with multiple WebSEAL servers operating behind a load balancer.

Conditions:

- Host name that is used by clients contacting the WebSEAL servers:

  `lb.example.com`
- WebSEAL servers:
  - `websealA.example.com`
  - `websealB.example.com`
  - `websealC.example.com`
  - `websealD.example.com`
- Active Directory domain: `MYDOMAIN`

General procedures:

1. Create a user ID in Active Directory for the various WebSEAL services to run as. For this example, the ID is **webseal**.

2. On the Active Directory server, run the following command:

   ```
   ktpass -princ HTTP/lb.example.com@MYDOMAIN -mapuser webseal -pass mypassw0rd
   -out lb_HTTP.keytab -mapOp set
   ```

   Note: The DNS name that is specified to the **ktpass** command must match the DNS name clients use to contact the load balanced WebSEAL servers.
3. The following message displays:

   ```
   Successfully mapped HTTP/lb.example.com to webseal.
   ```
4. You must also complete standard SPNEGO configuration for WebSEAL and Internet Explorer as described in the following section:
   - "Configure Windows desktop single sign-on" on page 149

   Ensure that you add `lb.example.com` to the Internet Explorer Trusted Sites list.
5. If you point a browser at `http://lb.example.com`, you are automatically authenticated to WebSEAL.

LTPA authentication

Security Access Manager supports authentication using an LTPA cookie received from the client. This section contains the following topics:

LTPA authentication overview

Various IBM servers provide support for the cookie-based lightweight third-party authentication mechanism (LTPA). Among these servers are WebSphere and DataPower®. To achieve a single signon solution to one or more of these servers, you can configure WebSEAL to support LTPA authentication.

The LTPA cookie, which serves as an authentication token for WebSphere/ DataPower, contains the user identity, key and token data, buffer length, and expiration information. This information is encrypted using a password-protected secret key that is shared between WebSEAL and the other LTPA enabled servers.

When an unauthenticated user makes a request for a WebSEAL protected resource, it will first determine whether an LTPA cookie is available. If an LTPA cookie is available, it will validate the contents of the cookie and, if successful, create a new session based on the user name and expiry time contained within the cookie. If no LTPA cookie is available, WebSEAL will continue to authenticate the user using the other configured authentication mechanisms. Once the authentication operation has been completed, a new LTPA cookie will inserted into the HTTP response and passed back to the client for consumption by other LTPA enabled authentication servers.

WebSEAL only supports LTPA version 2 (LtpaToken2) cookies. LtpaToken2 contains stronger encryption than prior versions of the token and enables you to add multiple attributes to the token. This token contains the authentication identity and additional information, such as the attributes that are used for contacting the original login server, and the unique cache key for looking up the Subject when considering more than just the identity in determining uniqueness. LtpaToken2 is generated for WebSphere Application Server Version 5.1.0.2 (for z/OS®) and for version 5.1.1 (for distributed) and beyond.

Enabling LTPA authentication

The `ltpa-auth` stanza entry is located in the `[ltpa]` stanza of the WebSEAL configuration file. It enables and disables the LTPA authentication method.

About this task

LTPA authentication is disabled by default. To configure LTPA authentication, complete the following steps:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the `[ltpa]` stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 23. Configuring LTPA authentication

Protocol to Support	Configuration File Entry
HTTP	`ltpa-auth = http`
HTTPS	`ltpa-auth = https`
Both HTTP and HTTPS	`ltpa-auth = both`
Disable LTPA authentication (default)	`ltpa-auth = none`

   For example, to support both protocols: `[ltpa] ltpa-auth = both`
3. Customize the entries contained within the `[ltpa]` stanza.
4. Restart the WebSEAL server.

Key file information

The LTPA token is encrypted by a password-protected secret key. The key itself is generated by WebSphere and is contained in a key file. This key file is password-protected by a clear text key.

The name of the key file that WebSEAL uses is defined by the **`keyfile`** configuration entry in the **`[ltpa]`** stanza. The permissions on the file must give read access to the user who is running the WebSEAL binary file.

The **`keyfile-password`** configuration entry in the **`[ltpa]`** stanza defines the password, which is used to protect the key file. If the password is sensitive, it can alternatively be stored in the corresponding configuration entry in the WebSEAL obfuscated database.

You can use the Local Management Interface (LMI) to manage this password.

Specifying the cookie name for clients

You can configure the name of the cookie containing the LTPA token that WebSEAL issues to clients.

By default, WebSEAL uses a cookie named **Ltpatoken2** to contain the LTPA token. Both WebSphere and DataPower expect this name by default. To customize the name of the client cookie that contains the LTPA token, change the value of the **cookie-name** configuration entry in the **[ltpa]** stanza.

For example:

```
[ltpa] cookie-name = Ltpatoken2
```

Specifying the cookie name for junctions

You can configure the name of the cookie that contains the LTPA token for junctioned web servers.

WebSphere Application Server and WebSEAL use the same default values for the LTPA cookie name:

- **LtpaToken** for LTPA tokens.
- **LtpaToken2** for LTPA version 2 tokens.

You can use the entry **jct-ltpa-cookie-name** in the **[ltpa]** stanza to configure the name of the LTPA cookies sent from WebSEAL across junctions on the backend. You can configure this item globally or on a per junction basis.

To set a cookie name for WebSEAL to use across all junctions, configure the entry in the **[ltpa]** stanza. For example:

```
[ltpa]
jct-ltpa-cookie-name = myGlobalLTPAcookie
```

To set a cookie name specific to a particular junction, configure the entry in an **[ltpa:/jct]** stanza.

where:

jct Name of the junction to the backend server.

For example:

```
[ltpa:/jct]
jct-ltpa-cookie-name = myLTPACookieForJct
```

If you use a custom LTPA cookie name in WebSEAL, you must also configure the same cookie name in WebSphere to achieve single sign-on. If you do not configure the **jct-ltpa-cookie-name** entry, WebSEAL uses the default cookie name.

Controlling the lifetime of the LTPA Token

By default, the lifetime of the LTPA cookie is set to the lifetime of the session that was used to create the token. For a more fine-grained approach, you can modify the **update-cookie** configuration entry in the **[ltpa]** stanza. This entry controls the frequency at which the token is updated with a new lifetime timeout.

Note: This configuration entry affects the LTPA cookie that WebSEAL issues to clients. It is the lifetime of the cookie that is specified by the **cookie-name** configuration entry in the **[ltpa]** stanza.

- The default value of `-1` indicates that the token is never updated and the lifetime of the token is equal to the maximum session lifetime.
- A value of zero indicates that the lifetime of the token is updated on every request. This configuration provides the functional equivalent of the inactivity timeout to the token.
- A positive number indicates the number of seconds that elapse between updates of the token. This configuration provides a less fine-grained equivalent of the inactivity timeout to the token.

Carefully consider whether to enable this configuration entry in your environment. The cost of creating the LTPA token and adding it to the HTTP response can outweigh the benefits gained by achieving an inactivity timeout for the token.

Disabling LTPA authentication

About this task

To disable LTPA authentication, complete the following steps:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file, setting **ltpa-auth** to *none*:
   ```
   [ltpa] ltpa-auth = none
   ```
3. Restart the WebSEAL server.

Results

Note: LTPA authentication is disabled by default.

OAuth Authentication

Security Access Manager supports OAuth 2.0 authentication. The implementation of OAuth in Security Access Manager strictly follows the OAuth standards.

OAuth is an HTTP-based authorization protocol. It provides third-party applications with scoped access to a protected resource on behalf of the resource owner. It provides scoped access by creating an approval interaction between the resource owner, the client, and the resource server. Users receive the ability to share their private resources between sites without providing user names and passwords.

WebSEAL provides two different mechanisms by which it can validate OAuth tokens:

ws-trust authentication
Authentication details are passed to a server for verification by using the ws-trust protocol. This is the legacy mechanism that is supported by IBM Security Access Manager and is previously known as `oauth-auth`.

OAuth introspection
Authentication details are passed to a server for verification by using an OAuth introspection endpoint.

For a complete description of the OAuth specifications, see the OAuth website: http://www.oauth.net

For more general information about OAuth support in Security Access Manager, see OAuth 2.0 and OIDC support.

Sample OAuth flow

The OAuth authentication supported by Security Access Manager is OAuth version 2.0. The method of providing the access token is through an HTTP header named "`Authorization`". Other forms of providing the access token are not supported. Here is a typical work flow to make use of OAuth authentication.

1. Acquire an access token from the OAuth server.

   Using curl, this could be accomplished as follows:

```
curl -k -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8"
-d "grant_type=client_credentials&client_id=<CLIENT_ID>
&client_secret=<CLIENT_SECRET>&redirect_uri="
https://<WEBSEAL_SERVER>/mga/sps/oauth/oauth20/token
```

Where:

- `CLIENT_ID` is the client ID that is created in the API Protection portion of the appliance LMI.
- `CLIENT_SECRET` is the secret associated with the client created in the API Protection portion of the appliance LMI.

The curl call above would return output resembling the following result:

```
{"expires_in":3599,"access_token":"iCIFH6k7KUq0oP55ZZFd",
"token_type":"bearer","scope":""}
```

Note that the returned result contains the **access_token** and its value, which is obtained from the OAuth server.

2. Access an API-protected resource.

   Using curl, this could be accomplished as follows:

```
curl  -k -c auth.txt -H "Authorization: Bearer iCIFH6k7KUq0oP55ZZFd"
https://<WEBSEAL_SERVER>/<API_protected resource>
```

Notice the HTTP header provided, named “`Authorization`”. The value of this header is the key word **Bearer** followed by the **access_token** that was obtained in Step 1. This access token is fed to the OAuth server and will allow the HTTP request to be satisfied for the API-protected resource.

ws-trust authentication

Authentication details are passed to a server for verification by using the ws-trust protocol. This is the legacy mechanism that is supported by IBM Security Access Manager and is also known as `oauth-auth`.

When OAuth Authentication is enabled, the WebSEAL configuration parameter **session lifetime timeout**, which is controlled by the **timeout** entry in the **[session]** stanza of the WebSEAL configuration file, is ignored. The session lifetime is set to the OAuth token expiry time.

Enabling and disabling OAuth authentication:

The **oauth-auth** stanza entry, located in the **[oauth]** stanza of the WebSEAL configuration file, enables and disables the OAuth authentication method. By default, OAuth authentication is disabled.

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[oauth]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 24. Configuring OAuth authentication

Protocol to support	Configuration file entry
HTTP	oauth-auth = http
HTTPS	oauth-auth = https
Both HTTP and HTTPS	oauth-auth = both
Disable OAuthc authentication	oauth-auth = none

For example, to support both HTTP and HTTPS protocols:

```
[oauth]
oauth-auth = both
```

3. Restart the WebSEAL server.

The user-identity-attribute stanza entry:

OAuth authentication must create a user credential. To do this, OAuth authentication must be provided with a user identity to use when creating this credential. The appliance's implementation of OAuth authentication provides the definition of the user identity through an attribute that is returned by the OAuth server.

The **user-identity-attribute** entry in the **[oauth]** stanza defines the name of the attribute that is returned by the OAuth server. This stanza entry's value is the user identity that is used when creating a credential for the OAuth authentication. By default, this entry has a value of **username**. What that tells the appliance is to take the value of the **username** attribute from the OAuth server response and use that as the user identity for the credential that will be created. By default, the **username** value in the OAuth server response is the client ID of the API protection client. That client ID must exist as an Security Access Manager user for OAuth authentication to be able to create a valid credential.

You can modify the OAuth server to provide the user identity in a different attribute, that is, something other than the **username** attribute. If you do that, modify the **user-identity-attribute** entry in the **[oauth]** stanza of the `webseald.conf` file, to provide that attribute name.

For more information, see the [oauth] stanza documentation in the IBM Knowledge Center.

Allowing external users to perform OAuth authentication:

You can allow external users that do not exist in the Security Access Manager registry to perform OAuth authentication.

OAuth authentication usually requires the identity that is represented by the OAuth token to exist in the Security Access Manager user registry. Support for a user that does not exist in the Security Access Manager registry requires some further configuration. The authorization server might need to update the attributes it puts in the RSTR in order to support external users.

At run time, when the appliance receives the RSTR from the authorization server, the user identity is extracted with the following order of precedence:

1. If the **pac-attribute** entry is present in the **oauth** stanza and the corresponding attribute is found in the RSTR, the PAC is used to authenticate the user. If this entry is not present, the PAC attribute is not used.
2. If the **external-user-identity-attribute** entry is present in the **oauth** stanza and the corresponding attribute is found in the RSTR, the value of this attribute is used as the username for authentication. If this entry is not present, the external user attribute is not used.

   If the **external-group-attribute** entry is present in the **oauth** stanza and the corresponding attribute is found in the RSTR, the group is added. The external group information is only used if authentication is occurring via an external user identity. If this entry is not present, the external group attribute is not used.

3. If no other authentication has already occurred, the username is used for authentication.

The authorization server must be changed to return these attribute values. If the **external-user-identity-attribute** entry is set to username, then external users can be enabled without any authorization server changes.

OAuth Introspection

Authentication details are passed to a server for verification by using an OAuth introspection endpoint.

When OAuth Introspection Authentication is enabled, the WebSEAL configuration parameter sessionlifetime timeout, which is controlled by the timeout entry in the [session] stanza of the WebSEAL configuration file, is ignored. The session lifetime is set to the OAuth token expiry time.

Enabling and disabling OAuth authentication:

The **oauth-introspection-auth** stanza entry, located in the **[oauth-introspection]** stanza of the WebSEAL configuration file, enables and disables the OAuth authentication method. By default, OAuth authentication is disabled.

About this task

Each configuration entry can be customized on a per junction basis by adding the configuration entry to a [oauth-introspection:{jct_id}] stanza, where '{jct-id}' refers to the junction point for a standard junction (include the leading '/'), or the virtual host label for a virtual host junction.

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[oauth-introspection]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 25. Configuring OAuth authentication

Protocol to support	Configuration file entry
HTTP	oauth-introspection-auth = http
HTTPS	oauth-introspection-auth = https
Both HTTP and HTTPS	oauth-introspection-auth = both
Disable OAuth authentication	oauth-introspection-auth = none

For example, to support both HTTP and HTTPS protocols:

```
[oauth-introspection]
oauth-introspection-auth = both
```

3. Restart the WebSEAL server.

Configure the OAuth Introspection capability:

The OAuth introspection capability is configured by using the [oauth-introspection] stanza.

Each configuration entry can be customised on a per junction basis by adding the configuration entry to a [oauth-introspection:{jct_id}] stanza, where '{jct-id}' refers to the junction point for a standard junction (include the leading '/'), or the virtual host label for a virtual host junction.

For more information on `[oauth-introspection]` stanza, see [oauth-introspection] stanza.

OpenID Connect (OIDC) authentication

OpenID Connect is a simple identity protocol and open standard that is built using the OAuth 2.0 protocol. It enables client applications to rely on authentication that is performed by an OpenID Connect Provider (OP) to verify the identity of a user. OpenID Connect uses OAuth 2.0 for authentication and authorization, and then builds identities that uniquely identify users.

WebSEAL provides a native OpenID Connect relying partner (RP) capability that is able to consume an identity token which has been provided by an OpenID Connect Provider in order to establish an authenticated session.

The WebSEAL implementation does not implement the complete specification for OIDC relying parties. The following parts of the specification are not supported by WebSEAL:

Table 26. Unsupported OIDC specifications

Section	Description
3.3	Hybrid Flow
5.3	Retrieving claims from the UserInfo Endpoint
6	Request Parameters as JWTs
8.1	Pairwise Subject Identifier Type
9	Only the client_secret_basic authentication type will be supported.
10.2	JWE - Encryption of the JWT
11	Offline Access
12	Using refresh tokens for authentication
15.3	Dynamic registration will not be supported.

In addition to this, the key identifier (KID) is required to be present in the JSON Web Key Set (JWKS) which is obtained from the OP.

If you need the complete RP capabilities, it is recommended that you instead use the RP that is provided as a part of the Security Access Manager Federation offering.

Landing page

A new landing page has been introduced to WebSEAL to handle the OIDC RP capability: `'/pkmsoidc'`.

This page serves as the redirect URI for the OIDC authentication flow. The format of the redirect URI is: `https://<webseal host>/pkmsoidc`. WebSEAL will use the host header from the HTTP request when constructing the redirect URI used in the authentication flow, unless a static redirect URI has been configured using the redirect-uri configuration entry. This entry only needs to be configured when WebSEAL resides behind a layer-7 load balancer.

The processing which is followed when this landing page is requested will differ based on the parameters supplied, namely:

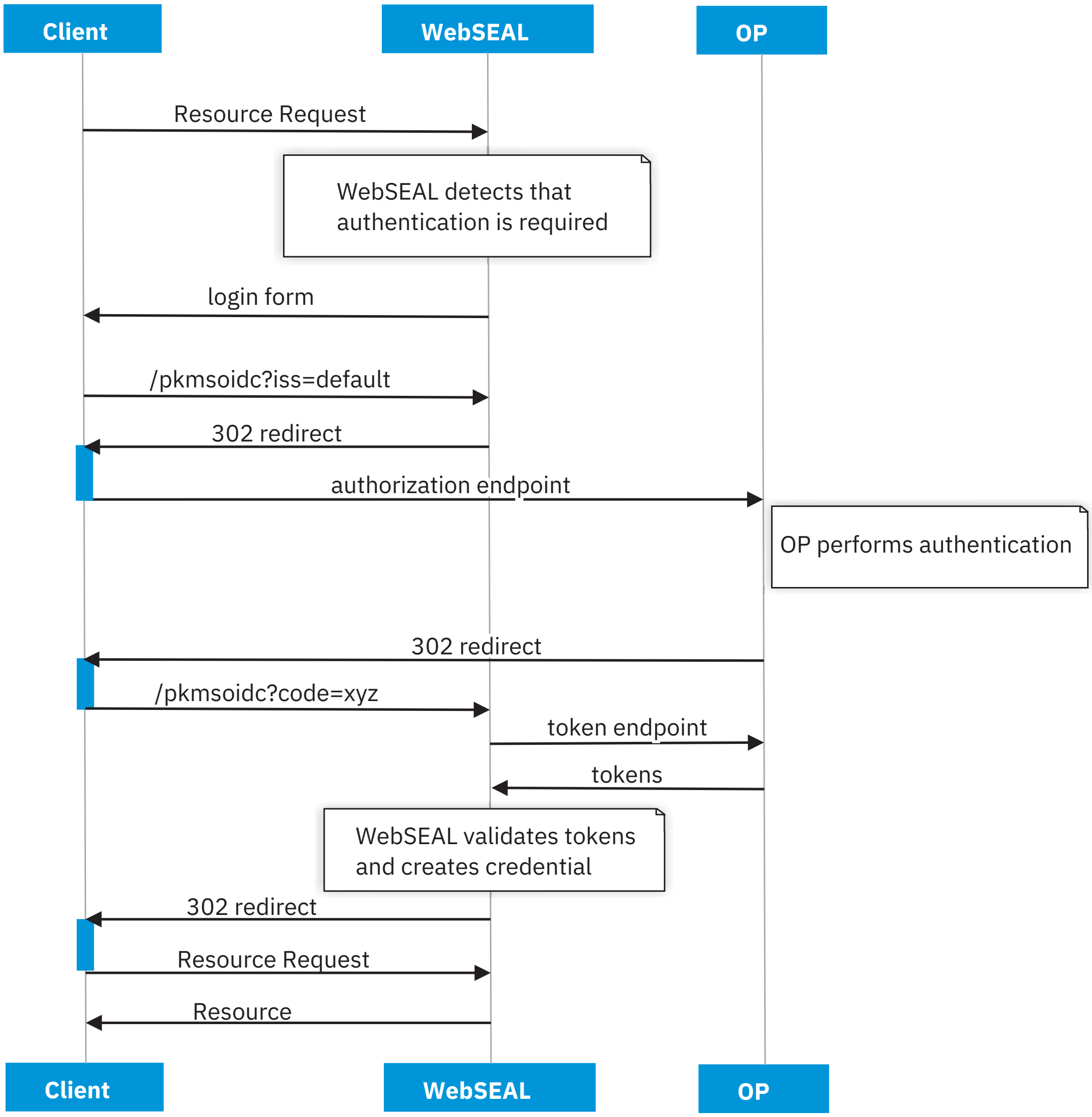
Method	Arguments	Processing
`GET`	None	Return the self-posting 'fragment' form, `oidc_fragment.html`. This file can be found in the management section of the WebSEAL document root. It is used during the OIDC implicit flow to retrieve the OIDC token data from the client.
`GET`	iss	This is the entry point into the OIDC authentication flow. A new authentication request will be generated for the specified OIDC issuer. The default configured issuer will be used if no issuer is specified.
`GET`	code	This is sent as a successful authentication response in the authorization code flow.
`POST`	None	This is sent as a successful authentication response in the implicit code flow.

Authentication flow

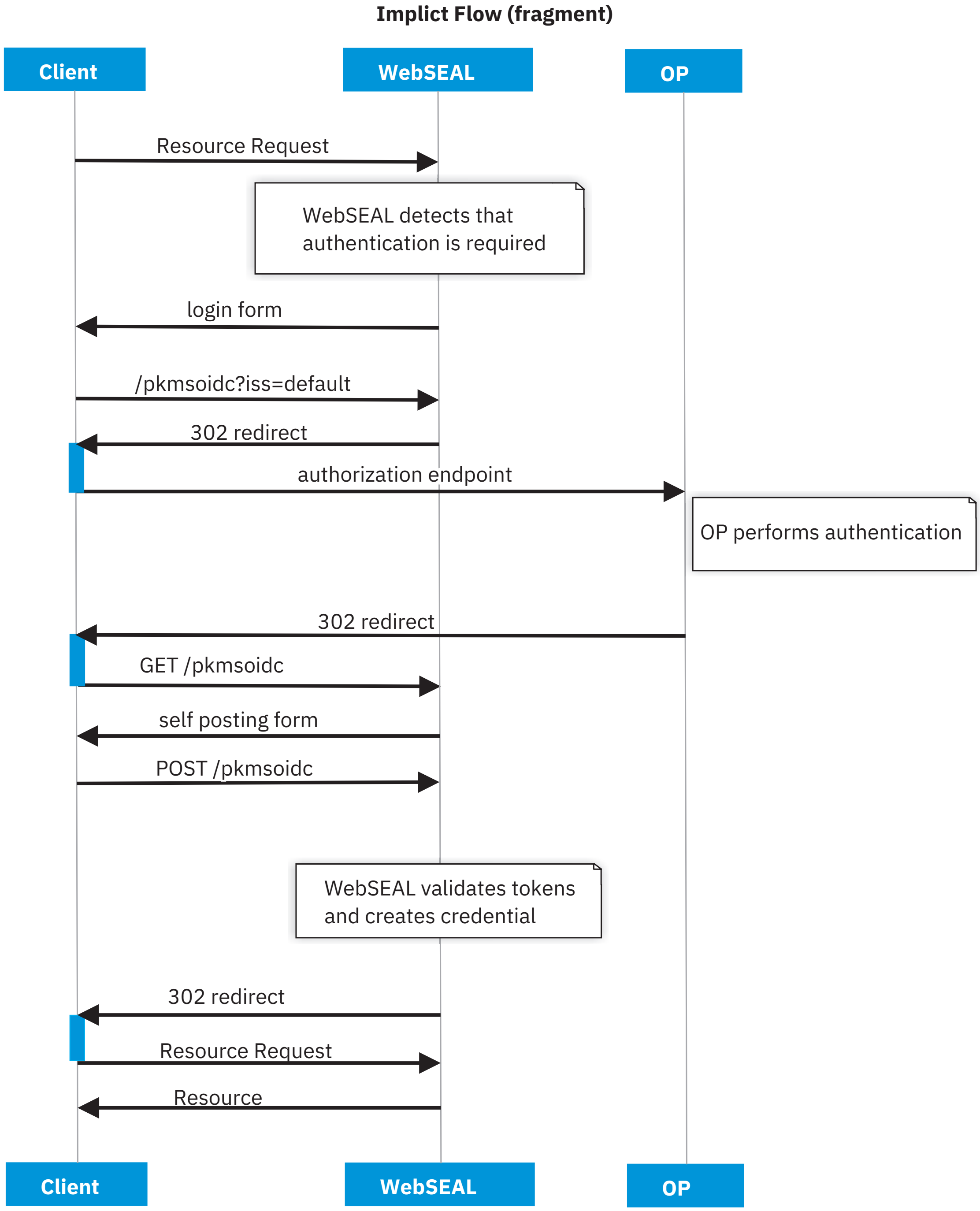
The OIDC specification states that authentication can follow one of three paths: the Authorization Code Flow, the Implicit Flow, or the Hybrid Flow. The flow determines how the ID Token and Access Token are returned to the Client. The Authorization Code Flow and the Implicit Flow are described below (please note that WebSEAL does not support the Hybrid Flow).

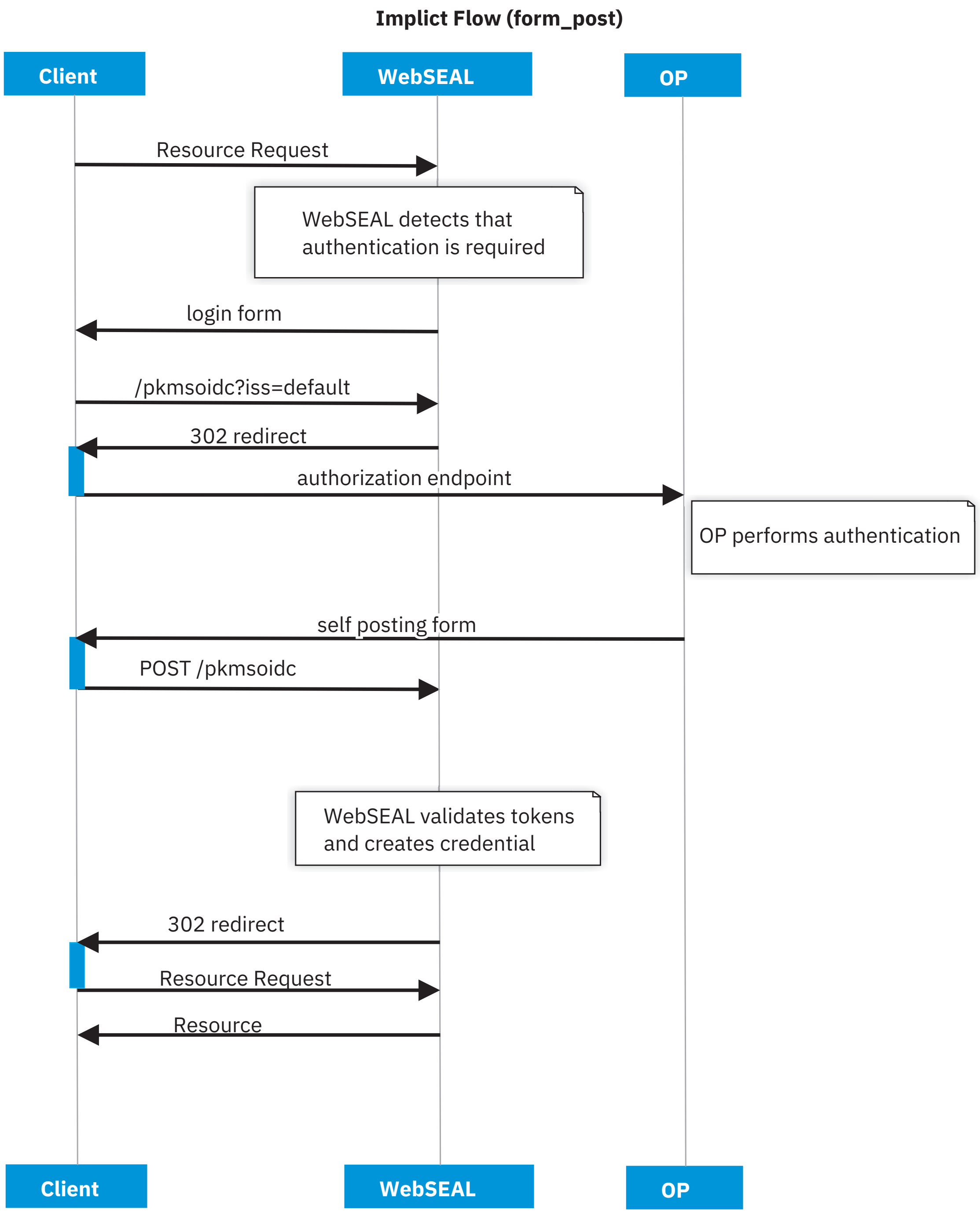
Authorization Code Flow

Authorization Code Flow



Implicit Flow


Implict Flow (fragment)
Client
WebSEAL
OP
Resource Request
WebSEAL detects that authentication is required
login form
/pkmsoidc?iss=default
302 redirect
authorization endpoint
OP performs authentication
302 redirect
GET /pkmsoidc
self posting form
POST /pkmsoidc
WebSEAL validates tokens and creates credential
302 redirect
Resource Request
Resource
Client
WebSEAL
OP



Enabling and disabling OIDC authentication

The `oidc-auth` stanza entry is located in the `[oidc]` stanza of the WebSEAL configuration file. It enables and disables the OIDC authentication method.

About this task

OIDC authentication is disabled by default. To configure OIDC authentication, complete the following steps:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the [oidc] stanza, specify the protocol to support in your network environment. The protocols are shown in the following table.

Table 27. Configuring OIDC authentication

Protocol to support	Configuration file entry
HTTPS	`oidc-auth = https`
Disable OIDC authentication (default)	`oidc-auth = none`

Note: OIDC authentication is not supported over the HTTP protocol.

3. Customize the entries contained within the `[oidc:<op-id>]` stanza, where `'<op-id>'` is a unique identifier for the OP.
4. Set the default OP ID by modifying the `default-op` entry in the `[oidc]` stanza.
5. Restart the WebSEAL server.

Configuring the OIDC RP

The RP functionality is configured using the `'[oidc]'` and `'[oidc:<op-id>]'` stanzas. Multiple OPs may be configured for authentication by creating a separate `'[oidc:<op-id>]'` stanza for each OP, where the `'<op-id>'` qualifier is a unique name for the OP. This qualifier can then be used to distinguish authentication requests for different OPs, using the `'iss'` argument to the `'/pkmsoidc'` landing page. A default OP might also be specified by using the default-op configuration entry, which is used as the default if no OP is specified in the iss argument to the `'/pkmsoidc?iss='` landing page.

Details of the configuration entries for the OP can be located in the stanza reference: [oidc:default] stanza

Error handling

If an error is encountered by the OP during the processing of the authorization grant during an authorization code flow, WebSEAL will return the following error page to the client: `3898342f.html`. The following macro's will be set based on the error response received from the OP:

Macro	Description
`%ERROR_CODE%`	A unique identifier for the error.
`%ERROR_TEXT%`	Human-readable ASCII encoded text description of the error. (optional)
`%ERROR_URL%`	URI of a web page that includes additional information about the error. (optional)

Advanced authentication methods

This chapter contains information that describes advanced WebSEAL authentication functionality.

Topic Index:

Multiplexing proxy agents

This section contains the following topics:

Multiplexing proxy agents overview

Security Access Manager provides solutions for securing networks that use a multiplexing proxy agent (MPA).

Standard Proxy Agents (SPA) are gateways that support per-client sessions between clients and the origin server over SSL or HTTP. WebSEAL can apply normal SSL or HTTP authentication to these per-client sessions.

Multiplexing proxy agents (MPA) are gateways that accommodate multiple client access. These gateways are sometimes known as WAP gateways when clients access using Wireless Access Protocol (WAP). Gateways establish a single authenticated channel to the origin server and "tunnel" all client requests and responses through this channel.

To WebSEAL, the information across this channel initially appears as multiple requests from one client. WebSEAL must distinguish between the authentication of the MPA server and the additional authentication of each individual client.

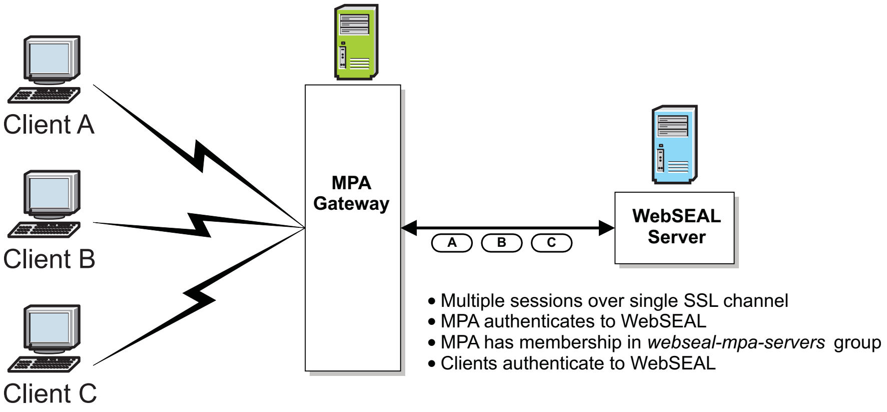

Client A
Client B
Client C
MPA Gateway
A
B
C
WebSEAL Server
• Multiple sessions over single SSL channel
• MPA authenticates to WebSEAL
• MPA has membership in *webseal-mpa-servers* group
• Clients authenticate to WebSEAL

Figure 15. Communication over an MPA Gateway

Because WebSEAL maintains an authenticated session for the MPA, it must simultaneously maintain separate sessions for each client. Therefore, the authentication method and session data used for the MPA must be distinct (different) from the session data and authentication method used by the client.

Valid session data types and authentication methods

Different session data types and authentication methods are valid depending on whether the session is for a Multiplexing proxy agent (MPA) or a client.

The following table lists the valid session types for the MPA and the client:

Table 28. Valid session types

Session Types	MPA-to-WebSEAL	Client-to-WebSEAL
SSL Session ID	Yes	Not valid
HTTP Header	Yes	Yes
IP Address	Yes	Not valid

Table 28. Valid session types (continued)

Session Types	MPA-to-WebSEAL	Client-to-WebSEAL
Session Cookie	Yes	Yes

The session data type used by the MPA to WebSEAL must be distinct from the session data type used by the client to WebSEAL. As an example, if the MPA uses a session cookie for the session data type, the client must use the HTTP Header session data type.

- The client cannot use an SSL session ID as the session data type.
- If MPA support is enabled, the function of **`ssl-id-sessions`** changes. Normally, if `ssl-id-sessions = yes`, only the SSL session ID is used to maintain sessions for HTTPS clients. To allow the MPA to maintain a session with an SSL session ID and have clients maintain sessions by using another method, this restriction is removed. See also “Valid session key data types” on page 311.

The following table lists the valid authentication methods for the MPA and the client:

Table 29. Valid authentication types

Authentication Types	MPA-to-WebSEAL	Client-to-WebSEAL
Basic authentication	Yes	Yes
Forms authentication	Yes	Yes
Certificate	Yes	Not valid
External authentication interface	Yes	Not valid

The authentication method used by the MPA to WebSEAL must be distinct from the authentication method used by the client to WebSEAL. As an example, if the MPA uses basic authentication, the client must use forms authentication.

- Certificates and external authentication interface authentication methods are not valid for use by the client.
- Normally, if forms authentication is enabled for a particular transport, basic authentication is automatically disabled for that transport. If MPA support is enabled, this restriction is removed. The MPA is then allowed to log in, for example, with forms and clients to log in with basic authentication over the same transport.

Authentication process flow for MPA and multiple clients

1. The WebSEAL administrator performs the following preliminary configuration:
   - Enable support for multiplexing proxy agents.
   - Create a Security Access Manager account for the specific MPA gateway.
   - Add this MPA account to the **webseal-mpa-servers** group.
2. Clients connect to the MPA gateway.
3. The gateway translates the request to an HTTP request.
4. The gateway authenticates the client.

5. The gateway establishes a connection with WebSEAL with the client request.
6. The MPA authenticates to WebSEAL (using a method distinct from the client) and an identity is derived for the MPA (which already has a WebSEAL account).
7. WebSEAL verifies the MPA's membership in the **webseal-mpa-servers** group.
8. A credential is built for the MPA and flagged as a special MPA type in the cache.

   Although this MPA credential accompanies each future client request, it is not used for authorization checks on these requests.
9. Now WebSEAL needs to further identify the owner of the request.

   The MPA is able to distinguish the multiple clients for proper routing of login prompts.
10. The client logs in and authenticates using a method distinct from the authentication type used for the MPA.
11. WebSEAL builds a credential from the client authentication data.
12. Session data type used by each client must be distinct from the session data type used by the MPA.
13. The authorization service permits or denies access to protected objects based on the user credential and the object's ACL permissions.

Enabling and disabling MPA authentication

About this task

The **mpa** stanza entry, located in the **[mpa]** stanza of the WebSEAL configuration file, enables and disables MPA authentication:

Procedure

- To enable the MPA authentication method, enter "yes".
- To disable the MPA authentication method, enter "no".

Example

```
[mpa]
mpa = yes
```

Creation of a user account for the MPA

See the Administering Security Access Manager Base administration information to learn how to create an account.

Addition of the MPA account to the webseal-mpa-servers group

Refer to the Configuring topics in the Knowledge Center for information on managing groups.

MPA authentication limitations

- Security Access Manager supports only one MPA per WebSEAL server.
- MPA authentication is not supported with step-up authentication configuration.
- MPA is not supported with `use-same-session = yes`

Switch user authentication

This section contains the following topics:

Overview of the switch user function

The WebSEAL switch user function allows administrators to assume the identity of a user who is a member of the Security Access Manager secure domain. The ability to assume a user's identity can help an administrator in a Help Desk environment to troubleshoot and diagnose problems. Switch user can also be used to test a user's access to resources and to perform application integration testing.

The switch user implementation is similar to the **su** command in UNIX environments. In the WebSEAL environment, the administrator acquires the user's credentials and interacts with resources and back-end applications with exactly the same abilities as the actual user.

The administrator uses a special HTML form to supply switch user information. WebSEAL processes the form and calls a special authentication mechanism that returns the specified user's credential without the requirement of knowing the user's password.

The following sequence describes the switch user process flow:

1. An administrator authenticates to WebSEAL. WebSEAL establishes a session for the administrator, and creates an entry for the administrator in the WebSEAL session cache.

   The session cache entry contains a cache data structure. This data structure stores the administrator's credential. During the switch user process flow, the cache data will be manipulated.

   For more information on WebSEAL session caches, see "WebSEAL session cache structure" on page 281.
2. The administrator requests a pre-configured switch user HTML form, and completes the form. On the form, the administrator specifies:
   - The name of the user identity that the administrator needs to assume.
   - A destination URL.
   - An authentication method.

   This action results in a POST request being sent to **/pkmssu.form**.

   The contents of the switch user HTML form can be modified before making it available for use by WebSEAL. See "Configuring the switch user HTML form" on page 177.

   You can also extend the capabilities of the form. See "Designing additional input forms" on page 178.

   Note: The `pkmssu.form` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.
3. WebSEAL determines whether to allow the switch user request by performing the following checks:
   a. WebSEAL examines the membership of the Security Access Manager **su-admins** group to determine if the administrator has permission to invoke the switch user function.

      Administrators requesting use of switch user authentication must be members of the su-admins group. Membership in this group must be configured before switch user can be used. For more information, see "Configuring user access" on page 176.

b. WebSEAL examines the membership of the Security Access Manager **su-admins** > **securitygroup** > **su-excluded** groups to ensure that the user identity supplied in the switch user form is not a member of one of these groups.

   User identities that belong to any of these groups cannot be accessed by the switch user function. The WebSEAL administrator must configure memberships in these groups before administrators use the switch user function. For configuration instructions and more information on these groups, see "Configuring user access" on page 176

4. When WebSEAL decides to allow the switch user request, WebSEAL calls the appropriate switch user module to perform the special switch user authentication.

   WebSEAL supports a variety of authentication mechanisms. Each authentication mechanism has a corresponding switch user authentication mechanism. WebSEAL provides built-in modules that contain the special switch user function.
5. When authentication of the designated user succeeds, the switch user module returns a valid credential for the user—without requiring the user password for input.
6. WebSEAL manipulates the contents of the appropriate entry in the WebSEAL session cache by:
   a. Removing the administrator's WebSEAL session cache data and storing it in a separate location.
   b. Inserting the switched-to user's cache data, including the user's credential, in place of the administrator's cache data.

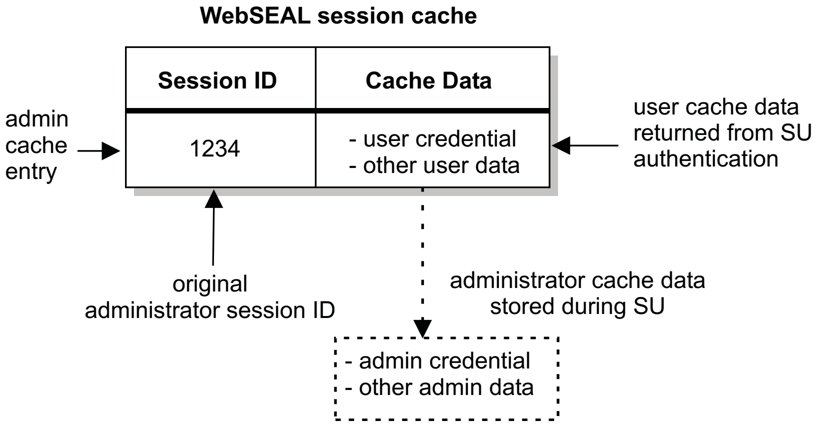


Figure 16. Swapping administrator and user cache data during switch user

7. WebSEAL sends a redirect to the browser for the destination URL supplied in the switch user form.

   The request is processed normally, using the user's credential.
8. The administrator can continue to make other requests. All authorization decisions for these requests are based on the credential of the user.

   When using switch user functionality, administrators might need to establish and manage sessions with additional applications. These sessions need to be established using the identity of the new user. To enable this, the new user credential also contains a new User Session ID. This User Session ID is used, for example, when troubleshooting the user's ability to access and use additional Web resources.

For more information on WebSEAL session caches, see "WebSEAL session cache configuration" on page 286 and "WebSEAL session cache structure" on page 281.

9. The administrator ends the switch user session using the standard Security Access Manager **/pkmslogout** utility. Upon successful log out:
   a. The user's cache data is deleted.
   b. The administrator's original cache data (and credential) is restored.
   c. The administrator is returned to the original page from which the switch user form was requested.

   The authorization service uses the original credential of the administrator for all subsequent requests.

Configuration of switch user authentication

The WebSEAL administrator must complete several configuration steps before administrators can use the switch user functionality. To configure switch user, complete the instructions in each of the following sections:

Configuring user access:

About this task

During WebSEAL installation, the WebSEAL configuration process automatically creates several groups for use by the switch user functionality. The WebSEAL administrator controls switch user capability by adding users to the groups.

To configure user access, complete the following steps:

Procedure

1. Add users to the **su-admins** group.
   To use switch user function, a user must be a member of a special administrative group called **su-admins**. This group is automatically created by default during installation of a WebSEAL server. There are no users in this group by default. The WebSEAL administrator must manually add users to this group. Typically, only administrative users are added to this group.
   Users who have been granted membership in **su-admins** can switch user to most other user identities, but cannot switch to the identity of any other user that is also a member of the **su-admins** group. Therefore, as soon as an administrator is granted switch user privileges by being added to **su-admins**, the administrator's account is protected from access by any other user that gains switch user privileges.
2. Add users to the **su-excluded** group
   This group contains the names of users whose identities should not be accessed through the switch user capability. During WebSEAL installation, the WebSEAL configuration process automatically creates this group. There are no users in this group by default. A WebSEAL administrator typically adds to this group the names of users who are not members of the administrative group **su-admins**, but for whom switch user access should still be blocked

Results

When switch user is used, WebSEAL also checks the memberships of the Security Access Manager group called **securitygroup**. This group contains the name of the

Security Access Manager administrative user **sec_master**, plus a number of WebSEAL processes that must be excluded from access through switch user capability.

The **securitygroup** group is automatically created by default during installation of a WebSEAL server. The following identities are automatically added to this group during installation:

- **sec_master** — the Security Access Manager administrator
- **acld** — the Security Access Manager authorization server daemon
- **webseald** — the WebSEAL daemon

WebSEAL administrators should *not* add any users to the **securitygroup** group. To control user access to switch user, use either **su-admins** or **su-excluded**.

Configuring the switch user HTML form:

WebSEAL provides a default HTML form that the administrator accesses to use the switch user function. The default form can be used without modification. Optionally, you can edit the form for customized appearance and functionality.

About this task

This step is optional.

The default form is named `switchuser.html`. You can modify the name of this file.

You can use the LMI to access this file in the `management/`*`lang`* directory. The value of the *`lang`* directory is specific to the locale. For example, the *`lang`* directory for a US English locale is called "`C`".

Form contents

The form contains requests for:

- User name

  The name of the user whose credentials the administrator wants to access.
- Destination URL

  This page displays after a successful switch user operation.
- Authentication method

  The authentication method stanza entries specify which authentication mechanism WebSEAL uses to build the user credential.

Each of these entries is required. WebSEAL verifies that all required data is present in the submitted form. If data is missing, the form is returned to the administrator with a descriptive message. When all required data is present, WebSEAL submits data from the switch user form data to the **/pkmssu.form** action URL.

Note: Only members of the **su-admins** group can invoke the form. An ACL is not required on this file. WebSEAL performs an internally hardcoded group membership check. WebSEAL returns a **`404 "Not Found"`** error when the group membership check fails.

Customizing the HTML form

To customize the switch user form, open the form for editing, and complete the following steps:

Procedure

1. Specify the location and contents of the destination URL.
   You can configure this URL as hidden input, which contains an appropriate home page or a successful switch user confirmation page.
2. Specify the authentication methods.
   You can configure this field as hidden input. Valid values for the authentication method include:
   ```
   su-ba su-forms su-certificate su-http-request
   su-cdsso
   ```
   The methods in this list map directly to authentication mechanisms specified in the WebSEAL configuration file. Note, however, that the **su-ba** > **su-forms** methods both map to the **su-password** authentication mechanism. Both basic authentication (ba) and forms authentication (forms) use the **su-password** authentication module. Note that a WebSEAL deployment can support basic authentication without supporting forms authentication. Therefore separate configuration values are maintained for each authentication type (**su-ba** > **su-forms**).

Designing additional input forms:

About this task

This step is optional.

You can design additional forms to validate or process data to be submitted to **/pkmssu.form**. These forms can be used to assist the administrator by populating some of the entries on the switch user form.

Some examples are:

- An administrator might have chosen to have different destination URLs, to be accessed based on the user identity. Another form could be written to build and present a list of these URLs, from which the administrator could select the appropriate entry.
- A form could be developed to call another program, such as a CGI script, to supply a list of user identities for whom switch user is allowed. This list could help administrators determine if access to a user identity through switch user is allowed.
- A form could be developed to display a list of user identities for whom switch user is not allowed. This list would be based on the memberships of the su-excluded and securitygroup groups.

Stopping and restarting WebSEAL:

About this task

To activate the new configuration changes you must stop and restart WebSEAL. This enables WebSEAL to use the new values that were specified to the WebSEAL configuration file in “Configuring user access” on page 176.

The methods for stopping and restarting the WebSEAL server are described in “WebSEAL instance management” on page 16.

Using switch user

About this task

When the configuration steps in the previous section have been completed, WebSEAL administrators can use the switch user function.

To use the switch user function, complete the following steps:

Procedure

1. Log in as a user who has permission to access the switch user function.

   This function is usually accessed by administrators. The user must be a member of the **su-admins** group.
2. Request the switch user HTML form.

   The default file name is `switchuser.html`. For information about this file, see "Configuring the switch user HTML form" on page 177.
3. On the form, specify:
   - The name of the user identity that you want to assume.
   - A destination URL.
   - An authentication method.

   This action results in a POST request being sent to **/pkmssu.form**. WebSEAL sends a redirect to the browser for the destination URL supplied in the switch user form. The request is processed using the user's credential, and the URL is accessed.

   Note: The `pkmssu.form` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.
4. Make other requests as necessary.

   All authorization decisions for these requests are based on the credential of the user.
5. When finished, end the switch user session by using the standard Security Access Manager **/pkmslogout** utility.

Results

For more information on how the switch user function works, see "Overview of the switch user function" on page 174.

Additional switch user feature support

This section describes switch user support for additional features such as reauthentication, user session management, and auditing.

Support for session cache timeout:

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

The inactivity timer continues to be reset while the administrator performs requests as the "switched-to" user. When the administrator ends the switch user session, the inactivity is still valid for the re-established administrator session.

The lifetime value is not extended because of a switch user operation. It is possible for the lifetime timeout of the session cache entry to expire during a switch user operation. If this timeout occurs, the session cache is deleted and the administrator is logged off. The administrator must reauthenticate and begin the switch user operation again.

Support for reauthentication:

WebSEAL reauthentication functionality is recognized by the switch user operation.

If reauthentication is required during a switch user operation, the administrator must authenticate as the "switched-to" user.

Note: The administrator must know the "switched-to" user's password to successfully reauthenticate.

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 sessions task and terminate all user sessions task perform as expected.

Support for tag-value:

The tag-value capability often used by custom authentication modules is recognized and supported by the switch user functionality.

Support for auditing:

It is possible to audit the administrator during a switch user operation. The switch user functionality adds an extended attribute to the "switch-to" user credential that identifies the administrator. The extended attribute, as stored in the credential, is called **tagvalue_su-admin**:

```
tagvalue_su-admin = su-admin-name
```

This extended attribute is available to any auditing mechanism.

Reauthentication

This section contains the following topics:

Reauthentication concepts

Security Access Manager WebSEAL can force a user to perform an additional login (reauthentication) to ensure that a user who is accessing a protected resource is the same person who initially authenticated at the start of the session. Forced reauthentication provides additional protection for sensitive resources in the secure domain.

Reauthentication can be activated by:

- A protected object policy (POP) on the protected object.

- Expiration of the inactivity timeout value of a WebSEAL session cache entry.

Reauthentication is supported by the following WebSEAL authentication methods:

- Forms (user name and password) authentication
- External authentication interface

In addition, a custom user name and password module can be written to support reauthentication.

Reauthentication assumes that the user has initially logged in to the secure domain and that a valid session (credential) exists for the user. The **reauth-at-any-level** option in the **[reauthentication]** stanza of the WebSEAL configuration file determines how WebSEAL handles a reauthentication operation:

- If the value for this option is no, the user must login using the same identity, authentication method, and authentication level that generated the existing credential. WebSEAL preserves the user's original session information, including the credential, during reauthentication. The credential is not replaced during reauthentication.
- If the value for this option is yes, the user must use the same identity but can be authenticated using a different authentication method or level from that which is currently held by the user. In this case, the user's credential can change one or more times during the lifetime of the user's session, and the user's credential is updated upon successful reauthentication. Note that this might have several consequences that must be carefully considered. If the credential changes during the course of an established session, then operations that utilize credential attributes, such as authorization decisions and auditing, might return different results mid-session. Care must be taken to ensure a consistent user experience and to account for these types of changes in audit records.

During reauthentication, WebSEAL also caches the request that prompted the reauthentication. Upon successful reauthentication, the cached data is used to rebuild the request. See "Server-side request caching" on page 235.

If reauthentication fails, WebSEAL returns the login prompt again. If reauthentication succeeds, but the ACL check fails for that resource, a 403 error ("Forbidden") is returned and the user is denied access to the requested resource.

In either case, the user is not logged off (the exception to this outcome is when the **max-login-failures** policy limit has been reached). Using a still valid credential, the user can terminate the reauthentication process (by requesting another URL) and still participate in the secure domain by accessing other resources that do not require reauthentication.

A configuration option exists that requires WebSEAL to remove the user's session cache entry and log the user out when the reauthentication attempts reach the **max-login-failures** policy limit.

Another configuration option is available to reset the lifetime timer of WebSEAL session cache entries. In addition, a grace period can be configured to allow sufficient time for the reauthentication process to complete before the lifetime timeout of a session cache entry expires.

Reauthentication based on security policy

Reauthentication based on security policy is activated by a specific extended attribute in a POP that protects the requested resource object. The POP can be directly attached to the object, or the object can inherit the POP conditions from a parent object.

Reauthentication POP: creating and applying

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 to create and apply the reauthentication POP. The following example illustrates creating a POP called "secure" with the **reauth** extended attribute and attaching it to an object (`budget.html`):

```
pdadmin> pop create secure
pdadmin> pop modify secure set attribute reauth true
pdadmin> pop attach /WebSEAL/hostA/junction/budget.html secure
```

Anyone attempting to access `budget.html` is forced to reauthenticate using the same identity and authentication method that generated the existing credential.

If the user requesting the resource is unauthenticated, the POP forces the user to authenticate. No reauthentication is necessary for this resource after successful initial login.

Details about the **pdadmin pop** commands can be found in the Command reference topics in the Knowledge Center.

Reauthentication based on session inactivity

Reauthentication based on session inactivity is enabled by a configuration stanza entry and is activated by the expiration of the inactivity timeout value of a session cache entry.

A user's session is normally regulated by a session inactivity value and a session lifetime value. When WebSEAL is configured for reauthentication based on session inactivity, the user's session cache entry is "flagged" whenever the session inactivity timeout value expires. The session cache entry (containing the user credential) is not removed. The user can proceed to access unprotected resources. However, if the user requests a protected resource, WebSEAL sends a login prompt. After successful reauthentication, the inactive session "flag" is removed and the inactivity timer is reset.

If reauthentication fails, WebSEAL returns the login prompt again. The session cache entry remains "flagged" and the user can proceed to request unprotected resources until the session cache entry lifetime value expires.

Two other conditions can end a user session: the user can explicitly log out or an administrator can terminate a user session. See “Terminating user sessions” on page 596.

Enabling of reauthentication based on session inactivity

To configure WebSEAL to "flag" inactive sessions rather than remove them from the session cache, set the value for the **reauth-for-inactive** stanza entry to yes in the **[reauthentication]** stanza of the webseald.conf configuration file:

```
[reauthentication]
reauth-for-inactive = yes
```

The default value for this stanza entry is no.

For information about enabling reauthentication based on session inactivity for the distributed session cache, see "Adjustment of the last access time update frequency for the distributed session cache" on page 343.

Resetting of the session cache entry lifetime value

The user's session cache entry has a limited lifetime, as specified by the **timeout** stanza entry in the **[session]** stanza of the webseald.conf configuration file. The default value, in seconds, is 3600 (1 hour):

```
[session]
timeout = 3600
```

Regardless of session activity or inactivity, the session cache entry is removed when the lifetime value is reached, at which point the user is logged off.

However, you can configure the lifetime of the session cache entry to be reset whenever reauthentication occurs. With this configuration, the user session no longer has a single maximum lifetime value. Each time reauthentication occurs, the lifetime value of the session cache entry is reset.

You can configure session cache entry lifetime reset with the **reauth-reset-lifetime** stanza entry in the **[reauthentication]** stanza of the webseald.conf configuration file:

```
[reauthentication]
reauth-reset-lifetime = yes
```

The default value is "no".

Extension of the session cache entry lifetime value

It is possible for the lifetime value of a session cache entry to expire while the user is performing a reauthentication. This situation occurs under the following conditions:

- The user requests a resource protected by a reauthentication POP
- The user's session cache entry lifetime value is very near expiration

The lifetime of a session cache entry can expire after the reauthentication login form is sent to the user and before the completed login form is returned. When the session cache entry lifetime value expires, the session cache entry is deleted. When the login form is returned to WebSEAL, there is no longer a session for that user. In addition, all cached user request data is lost.

You can configure a time extension, or "grace period," for the session cache entry lifetime value if the session cache entry lifetime expires during reauthentication.

The **reauth-extend-lifetime** stanza entry in the **[reauthentication]** stanza of the webseald.conf configuration file provides this time extension, in seconds. For example (5 minutes):

```
[reauthentication]
reauth-extend-lifetime = 300
```

The default value, "0", provides no extension to the session cache entry timeout value.

The **reauth-extend-lifetime** stanza entry applies to users with existing session cache entries and who are required to reauthenticate. For example:

- Users performing reauthentication resulting from POP security policy
- Users performing reauthentication resulting from session cache inactivity
- Users performing step-up authentication

The **reauth-extend-lifetime** option is intended to be used in conjunction with the **reauth-reset-lifetime=yes** option.

Prevention of session removal when the session lifetime expires

The session cache entry lifetime value usually determines the maximum session length. It is possible for a user to remain active for the full duration of a session lifetime. When the session lifetime value expires, the session cache entry is normally removed and the user is logged off, regardless of activity.

To prevent this sudden session termination, you can configure WebSEAL to allow the user to reauthenticate after the session timeout value has expired. After successful reauthentication, the lifetime value of the session cache entry is reset.

WebSEAL allows resetting of the session lifetime value, after it has expired, under the following conditions:

- Reauthentication based on inactivity policy is enabled (reauth-for-inactive=yes)
- The session lifetime value (timeout) has expired
- The time extension ("grace period") for the session lifetime is enabled and set to a reasonable value (for example, reauth-extend-lifetime=300)
- The user activates the reauthentication prompt by requesting a protected resource before the time extension ("grace period") expires

  (WebSEAL does not allow repeated additions of the time extension to an end of session lifetime event.)
- Resetting the session cache lifetime is configured to be true (reauth-reset-lifetime=yes)

At the occurrence of a session lifetime expiration, WebSEAL checks the conditions listed above. If all conditions are met, the lifetime timeout is extended by the **reauth-extend-lifetime** value and the user's session cache entry is "flagged" as extended. The session cache entry (containing the user credential) is not removed and the user can proceed to access unprotected resources. When the user requests a protected resource, WebSEAL prompts the user to reauthenticate.

The **reauth-extend-lifetime** value should be set to a reasonable value so the user has enough time to trigger the reauthentication prompt. Note that if the user does not access a protected object during the "grace period", the reauthentication process is not activated. In this case, it is possible for the **reauth-extend-lifetime** value to expire, in which case the session cache entry is removed.

Typically, however, reauthentication policy is implemented to secure an application that is serving predominantly protected resources. A time extension ("grace period") of 5–10 minutes should be adequate time to allow an active user to trigger the reauthentication process, and therefore reset the session lifetime value.

Removal of a user session at login failure policy limit

If a reauthentication attempt fails, WebSEAL normally returns the login prompt again. Because the user still has a valid session and credential, the user can terminate the reauthentication process (by requesting another URL) and still participate in the secure domain by accessing other resources that do not require reauthentication.

However, the reauthentication process is impacted by the login failure policy (**max-login-failures**) if the user continues failed attempts to reauthenticate. When the number of failed reauthentication attempts reaches or exceeds the **max-login-failures** limit, WebSEAL responds according to the **terminate-on-reauth-lockout** configuration.

The **terminate-on-reauth-lockout** stanza entry is located in the **[reauthentication]** stanza of the WebSEAL configuration file. The purpose of this stanza entry is to control whether or not the user's session cache entry is completely removed upon reaching the **max-login-failures** policy limit.

The default setting is "yes". When the maximum number of failed login attempts (specified by the **max-login-failures** policy) is reached during reauthentication, the user is logged out of the original session and the user's session cache entry is removed. For example:

```
[reauthentication]
terminate-on-reauth-lockout = yes
```

Now the user no longer has a valid session and credential. Although the user can still access unprotected resources, the user is required to login again for any request made to any protected resource.

A value of "no" for the **terminate-on-reauth-lockout** stanza entry is provided as backward compatibility for versions of WebSEAL prior to version 6.0.

```
[reauthentication]
terminate-on-reauth-lockout = no
```

With the "no" setting, the user is not logged out and the initial login session is still valid. The user can still access other resources that are not protected by a **reauth** POP.

When the maximum number of failed login attempts (specified by the **max-login-failures** policy) is reached during reauthentication, the user is locked out from accessing that resource as specified by the **disable-time-interval** policy setting, and notified of the lockout as specified by the **late-lockout-notification** configuration setting.

For both values of **terminate-on-reauth-lockout**, the specific response to the user is governed by the **disable-time-interval** and **late-lockout-notification** settings.

If the **disable-time-interval** policy is set to a number of seconds, the error message indicates that the account is temporarily locked out.

If the **disable-time-interval** policy is set to "disable", the error message indicates that the account has been disabled and that an administrator is required to reset (unlock) the account.

For complete details on the login failure policy mechanism, see “Login failure policy ("three strikes" login policy)” on page 238.

Customization of login forms for reauthentication

WebSEAL supports reauthentication for both forms authentication methods.

By default, forms authentication uses the `login.html` page to request user name and password information from the client (see “Static server response pages” on page 58). This default login page is also used during reauthentication.

It is possible to have the user name field in these login pages automatically filled in during reauthentication by using the USERNAME macro (see “Macro resources for customizing response pages” on page 66). The user needs to complete only the password (passcode) field.

For example, modify the following line in the `login.html` page:

```
<TD><INPUT NAME="username" SIZE="15"></TD>
```

to include the USERNAME macro:

```
<TD><INPUT NAME="username" SIZE="15" VALUE="%USERNAME%"></TD>
```

During an initial login (unauthenticated user), the value for the USERNAME macro is empty and the user name text field displayed on the login page appears with the entry "unknown".

For a reauthenticating client, the USERNAME macro would contain the value of the client user name. The user name text field on the login page appears with the user's name automatically provided.

Authentication strength policy (step-up)

This section contains the following topics:

Authentication strength concepts

WebSEAL supports many authentication methods. These include basic authentication, forms authentication, certificate authentication, and others. Any client that accesses a WebSEAL server has an authentication state, such as *unauthenticated* or *certificate*, which indicates the method by which the client last authenticated with WebSEAL.

WebSEAL provides a feature that enables administrators to assign a ranking or level to some of the supported authentication methods. Administrators can define an ordered list that ranks each authentication method from lowest to highest. This hierarchical ranking can be arbitrarily tailored to each individual WebSEAL deployment.

There is no absolute ranking between the authentication methods. No one authentication method is inherently better or stronger than another method. The ranking is simply a method for an administrator to define a relative level for each authentication method for use with a specific Security Access Manager WebSEAL protected object namespace. The only rule governing the assignment of levels is that the *unauthenticated* level is always lower than all other authenticated levels.

This set of authentication levels can be used to implement an authentication strength policy. Authentication strength is sometimes called *step-up authentication*. Note, however, that step-up authentication is not a unique authentication method like forms authentication or certificate authentication. Instead, it is a defined process for requiring users to change their current authentication method to another authentication method.

The concept of changing the authentication method is useful as a way of providing additional protection for selected resources in the WebSEAL protected object namespace. For example, a user can log in using certificate authentication, and then access many resources that are protected by Security Access Manager security. When the user attempts to access a more sensitive resource, which has been marked to require a higher level of access, the user is prompted to log in to a different authentication level.

Note that when a user activates authentication strength by attempting to access a protected object, the user does not have to log out first. Instead, the user is presented with a login prompt, and simply logs in again to the higher level.

Users can change authentication strength multiple times per authentication session. The authentication level specified in the controlling POP governs the level at which the user must be authenticated.

The following authentication methods can be assigned an authentication level:

- Unauthenticated
- Password authentication

  Password authentication is limited to forms authentication. Basic authentication is not supported as a step-up authentication level.
- Certificate authentication
- External authentication interface

Authentication strength is supported over both HTTP and HTTPS, with the exception of certificate authentication. Because certificates are valid only over an SSL connection, it is not possible to step up to certificates over HTTP. If an object that requires certificate authentication is requested over HTTP, an error page is served, as specified by the **cert-stepup-http** stanza entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file.

Administrators apply an authentication level to a protected resource by declaring and attaching a standard Security Access Manager protected object policy (POP) to the resource object. Authentication strength policy is set and stored in a POP attribute called an *IP Endpoint Authentication Method*. The attribute takes an integer value that represents the authentication level. The lowest level, *unauthenticated*, is always 0. Each level increases the integer index up to the total number of authentication methods that have been assigned a level.

When clients first authenticate to WebSEAL, the initial authentication method used is stored as an extended attribute in the client's credential. The Security Access Manager authorization service compares the authentication method (level) in the credential against the authentication level for the requested resource, as specified in the POP. When the level in the POP exceeds the level in the credential, the user is prompted to authenticate at the higher authentication strength level.

The IP Endpoint Authentication Method attribute can also optionally be used to restrict access to a resource, based on the network address of the client that sent the access request. The access can be restricted based on an individual network (IP) address, or a range of network addresses.

WebSEAL uses the following algorithm to process the conditions in a POP:
1. Check the IP endpoint authentication method policy on the POP.
2. Check ACL permissions.
3. Check time-of-day policy on the POP.
4. Check the audit level policy on the POP.

Authentication strength configuration task summary

To configure authentication strength levels, complete the instructions in each of the following sections:
1. "Establishing an authentication strength policy"
2. "Specifying authentication levels" on page 189
3. "Specifying the authentication strength login form" on page 190
4. "Creating a protected object policy" on page 191
5. "Specifying network-based access restrictions" on page 192
6. "Attaching a protected object policy to a protected resource" on page 195
7. "Enforcing user identity match across authentication levels" on page 195
8. "Controlling the login response for unauthenticated users" on page 196
9. "Stepping up authentication at higher levels" on page 196

Establishing an authentication strength policy

About this task

This section consists of planning steps to be taken before specifying authentication strength settings in the WebSEAL configuration file.

Complete the following steps:

Procedure

1. Compile a list of protected objects for which access will be limited only to users who have successfully authenticated through a specific authentication method. For each protected object, specify the authentication method that applies.
2. Compile a complete list of all authentication mechanisms that will be active (enabled) on the WebSEAL server system.
3. Determine a hierarchy (ranking) for the active authentication mechanisms. Order the mechanisms from weakest to strongest.
4. Determine if, during authentication strength level step-up, the user identity must be identical across the increased authentication level.
5. Determine if any protected resources require access restriction based on the network address of the requesting client.
6. Stop the WebSEAL server.

Specifying authentication levels

About this task

Complete the following steps to specify authentication levels.

Procedure

1. Edit the **[authentication-levels]** stanza in the WebSEAL configuration file. For each authentication method to be used for authentication level step-up, add an entry to the stanza. The supported authentication methods are described in the following table:

Table 30. Authentication methods supported for authentication strength

Authentication Method	Configuration File Entry
None	`level = unauthenticated`
Forms authentication	`level = password`
Certificate authentication	`level = ssl`
External authentication interface	`level = ext-auth-interface`
Lightweight Third-Party Authentication (LTPA)	`level = ltpa`
OpenID Connect (OIDC)	`level = oidc`

The default entries are:

```
[authentication-levels]
level = unauthenticated
level = password
```

The following entry must always be the first in the list: `level = unauthenticated`. Additional entries can be placed in any order. For example, to enable authentication strength levels for certificate authentication at the highest level, the completed stanza entry is:

```
[authentication-levels]
level = unauthenticated
level = password
level = ssl
```

2. Verify that each authentication method listed in **[authentication-levels]** is enabled. To determine if an authentication method is enabled, check the appropriate entries in the WebSEAL configuration file. To review the necessary entries and access the authentication configuration instructions, see the following sections:
   - "Enabling and disabling basic authentication" on page 125
   - "Enabling and disabling forms authentication" on page 126
   - "Enabling certificate authentication" on page 131
   - "Enabling and disabling OIDC authentication" on page 169

   Note: Basic authentication is enabled by default.

Using multiple authentication levels:

You can associate more than one authentication level with a particular authentication mechanism.

Authentication mechanisms can set the authentication level, which results from a successful authentication, directly into the credential as an attribute. If so, this overrides the level set in the credential by the placement of the authentication mechanism in the **[authentication-levels]** stanza. You can use this method to specify the authentication level that will be set in the resulting Security Access Manager credential. Use it when you want to associate more than one level with a particular authentication mechanism.

The reasons to enter an authentication mechanism more than once in the **[authentication-levels]** stanza are:

1. To control the prompt that is displayed to a user when they are required to step-up.
2. To satisfy the Policy Server requirement that a method exists that can handle every configured POP level.

This is likely to be a consideration when using an External Authentication Interface (EAI) server to perform authentication. It is common to want the EAI to handle multiple authentication levels. An EAI server can return an authentication level either as an attribute in a Privilege Attribute Certificate (PAC), if it returns one, or as an extended attribute header if it returns a User Id. The EAI authentication mechanism can then be specified in multiple lines of the **[authentication-levels]** stanza. For example, if an EAI server is configured to handle authenticating users at levels 2 and 3, while Forms authentication is used to authenticate users at level 1, the **[authentication-levels]** stanza would contain the following entries:

```
[authentication-levels]
#----------------------
# STEP UP
#----------------------
# authentication levels
#
# Syntax:
# level = <method-name>
#
# Valid method names are:
#   unauthenticated
#   password
#   ssl
#   ext-auth-interface
#   ltpa
#   oidc
#
level = unauthenticated
level = password
level = ext-auth-interface
level = ext-auth-interface
```

Specifying the authentication strength login form

About this task

When an authenticated client attempts to access a protected resource, and is required to reauthenticate to a higher authentication strength level, WebSEAL presents a special HTML form. The client uses the form to supply the information needed for the type of authentication required.

WebSEAL supplies a default login form. Administrators can either use the default login form or customize it to fit the local WebSEAL deployment.

The location of the default login form is specified in the WebSEAL configuration file:

```
[acnt-mgt]
stepup-login = stepuplogin.html
```

Complete the following steps:

Procedure

1. Specify the name of the authentication strength login form.

   To use the default location for the form, verify that the WebSEAL configuration file stanza entry, **stepup-login**, contains the default value, `stepuplogin.html`.
2. Optionally, customize the contents of the authentication strength login form.

   This file contains macros, in the form of `%TEXT%` sequences, which are replaced with the appropriate values. This substitution occurs within WebSEAL's template file processing functions and allows the form to be used for the supported authentication methods with correct formatting. It also allows other information, such as error message and authentication method name, to be supplied in the form for the user.

   For more information on using macros, see "Macro resources for customizing response pages" on page 66.
3. Restart the WebSEAL server.

Results

The configuration of authentication strength levels is now complete.

Creating a protected object policy

About this task

Complete the following steps:

Procedure

1. Create a POP. For example, use **pdadmin** to create a new POP named `test`:

   ```
   pdadmin> pop create test
   ```
2. Display the contents of the new POP:

   ```
   pdadmin> pop show test
   ```

   The new POP contains new settings similar to the following:

   ```
   pdadmin> pop show test
          Protected object policy:  test
          Description:
          Warning:  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
   ```
3. Note the default values in the POP for the attribute `IP Endpoint Authentication Method Policy`.

   ```
   ...
   ...
   IP Endpoint Authentication Method Policy
             Any Other Network 0
   ...
   ```

   The IP Endpoint Authentication Method Policy attribute is used to specify two different attributes:

- Authentication strength level.

  The default value is `0`.
- Network-based access policy.

  The default value is `Any Other Network`.

4. Use **pdadmin pop modify** to modify the IP Endpoint Authentication Method Policy attribute to specify the authentication strength level that you want to apply to the resources identified in "Establishing an authentication strength policy" on page 188. The syntax is:

   ```
   pdadmin> pop modify pop-name set ipauth anyothernw level-index
   ```

   The value *level-index* is an integer. The default value is `0`. The default value maps to the authentication strength level `unauthenticated`.

   Specify the index that corresponds to the necessary authentication strength level. To determine the correct *level-index*, examine the **[authentication-level]** stanza in the WebSEAL configuration file.

   ```
   For example:[authentication-levels]
   level = unauthenticated
   level = password
   level = ssl
   ```

   For the above entry, the index values are described in the following table:

Table 31. Example integer values for authentication strength levels

Authentication method	Index value
unauthenticated	0
password	1
ssl	2

   For example, to add the password authentication strength level (index value 1) to the *test* POP, enter:`pdadmin> pop modify test set ipauth anyothernw 1` To verify the modification, display the POP:

   ```
   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 1
   ```

   Note: In this example, the only valid index values are: 0,1,2. If any other index value is configured, WebSEAL presents an error page whenever a client requests any object with that has the POP attached.

Specifying network-based access restrictions

About this task

Security Access Manager supports an optional POP configuration setting that enables the application of authentication strength levels to client requests originating from specified network addresses. The network addresses can be defined as either a single IP address, or as a range of IP addresses.

Note: In most deployments, user access is not restricted based on the IP address within POPs. In most deployments, this configuration section can be skipped.

The **pdadmin pop modify set ipauth** command is used to specify IP addresses. Note that this is the same **pdadmin** command used to specify authentication levels.

The default usage of **pdadmin pop modify set ipauth** does not impose any network-based access restrictions. This usage consists of specifying the command line argument `anyothernw` as the value for the IP Endpoint Authentication Method Policy attribute. This setting applies to all user access, regardless of the IP address of the requestor, and requires all users to authenticate at the specified level.

The syntax is:

```
pdadmin> pop modify pop-name set ipauth anyothernw level_index
```

For example, in "Creating a protected object policy" on page 191 above, the following command created a POP that required all users to authenticate at authentication level 1, and did not impose any network-based access requirements:

```
pdadmin> pop modify test set ipauth anyothernw 1
```

Procedure

The following network-based access restrictions can be applied:

- **Require a specific authentication strength level when the IP address of the requesting client is within a defined range of IP addresses.**

  Syntax:

  ```
  pdadmin> pop modify pop_name set ipauth add network netmask level_index
  ```

  Note that the **pdadmin pop modify set ipauth add** command specifies both the network addresses and the required authentication level in the IP Endpoint Authentication Method attribute.

  For example, to require users from IP address range 9.1.2.[0–255] to use authentication strength level 1:

  ```
  pdadmin> pop modify test set ipauth add 9.1.2.1 255.255.255.0 1
  ```

  Note that the value specified for the netmask determines the range of network addresses affected. The number 0 in the netmask serves as a wildcard to mean all IP addresses for that subnet. See the example that follows.

Table 32. Examples of using netmask to specify a network range (IPv4)

IPv4 Address	Netmask	Network range affected
9.1.2.3	255.255.255.0	9.1.2.[0–255]
9.1.2.3	255.255.0.0	9.1.[0–255].[0–255]
9.1.2.3	255.0.0.0	9.[0–255].[0–255].[0–255]

Table 33. Examples of using netmask to specify a network range (IPv6)

IPv6 Address	Netmask	Network range affected
fec0::1	fff0::	fec[0-f]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]
fec0:ffff::1	ffff:fff0::	fec0:fff[0-f]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]:[0-ffff]

- **Require requests from one specific IP address to use a specified authentication strength level.**

For example, to require requests from IP address 9.1.2.3 to use authentication strength level 1:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 1
```

To require requests from all IP addresses on subnet 9.1.2.x to use authentication strength level 1:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 1
```

- **Disable use of authentication strength level step-up by all requests from a range of network addresses.**

  The syntax is:

  ```
  pdadmin> pop modify pop_name set ipauth remove network netmask
  ```

  For example, to disable all requests from the range of IP addresses on the 9.1.2.x subnet:

  ```
  pdadmin> pop modify test set ipauth remove 9.1.2.1 255.255.255.0
  ```

- **Allow access to the protected resource based solely on IP address, or range of IP addresses, regardless of the authentication strength level.**

  This restriction is enforced by specifying the IP address or addresses, and assigning an authentication level of zero (0). For example, to allow requests from IP address 9.1.2.3, regardless of authentication strength level:

  ```
  pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 0
  ```

  Likewise, to allow requests from all IP addresses on the 9.1.2.x subnet, regardless of authentication strength level:

  ```
  pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 0
  ```

- **Deny access based solely on IP address, or range of IP addresses, regardless of authentication strength level.**

  This restriction is enforced by using the key word `forbidden` as the final parameter.

  For example, to restrict only the client at IP address 9.1.2.3 from accessing the protected resource:

  ```
  pdadmin> pop modify test set ipauth 9.1.2.3 255.255.255.255 forbidden
  ```

  Likewise, to restrict requests from all IP addresses on the 9.1.2.x subnet from accessing the resource:

  ```
  pdadmin> pop modify test set ipauth 9.1.2.3 255.255.255.0 forbidden
  ```

- **Prevent requests from all IP addresses from accessing the protected object, unless the IP address has been enabled by a previous** pop modify set ipauth add **command.**

  For example, in a use case above, a range of IP addresses were required to access the protected resource by using authentication strength level 1:

  ```
  pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.0 1
  ```

  The administrator can, in addition, specify that requests from all other IP addresses will be denied, regardless of authentication strength level, in the following **pdadmin** command:

  ```
  pdadmin> pop modify test set ipauth anyothernw forbidden
  ```

  The option **anyothernw** means *any other network address*, and the option **forbidden** enforces the denial policy.

Attaching a protected object policy to a protected resource

About this task

After a protected object policy (POP) has been defined and created, it must be attached to the protected resources to which it applies. The syntax for attaching a POP is:

```
pdadmin pop attach object_name pop_name
```

For example, an authentication policy for a WebSEAL deployment could be defined as follows:

- The deployment will use forms authentication and certificate authentication. Forms authentication is the first authentication strength level (1) and certificate authentication is the second (stronger) authentication level (2).
- Users must authenticate using forms authentication or stronger to access the following protected resource (a WebSEAL junction):
  ```
  /WebSEAL/hostA/junction
  ```
- Users must authenticate using certificate authentication to access the following protected resource (an application):
  ```
  /WebSEAL/hostA/junction/applicationA
  ```

To implement this policy, the following configuration steps must take place.

Procedure

1. Modify the WebSEAL configuration file to grant forms authentication an authentication strength of 1 and certificate authentication a strength of 2:
   ```
   [authentication-levels]
   level = unauthenticated
   level = password
   level = ssl
   ```
2. Create a POP for authentication level 1 (forms authentication).
   ```
   pdadmin> pop create test1
   pdadmin> pop modify test1 set ipauth anyothernw 1
   ```
3. Create a POP for authentication level 2 (certificate authentication).
   ```
   pdadmin> pop create test2
   pdadmin> pop modify test2 set ipauth anyothernw 2
   ```
4. Attach the POP `test1` to `/WebSEAL/hostA/junction`.
   ```
   pdadmin> pop attach /WebSEAL/hostA/junction test1
   ```
5. Attach the POP `test2` to `/WebSEAL/hostA/junction/application`.
   ```
   pdadmin> pop attach /WebSEAL/hostA/junction/applicationA test2
   ```

Results

Note: For more information on the administration of POPs, see the Administering Security Access Manager Base administration information. For information on **pdadmin** syntax, see the Command Reference topics in the Knowledge Center.

Enforcing user identity match across authentication levels

About this task

By default, WebSEAL requires the user identity that performs the authentication strength (step-up) operation to match the user identity used to perform the initial authentication operation.

WebSEAL verifies that the user identity in the new user credential matches the user identity in the original credential. If the user identities do not match, WebSEAL denies the authentication step-up, logs an error and returns an error page to the user.

Procedure

This function is enabled by default.

To disable this function, edit the WebSEAL configuration file, and set the value of **verify-step-up-user** to `no`:

```
[step-up]
verify-step-up-user = yes
```

Controlling the login response for unauthenticated users

About this task

You can control the login prompt response for an unauthenticated user who requests an object protected by a step-up authentication POP attribute.

By default, WebSEAL presents only the login prompt for the specific authentication level required by the POP. The **show-all-auth-prompts** stanza entry in the **[step-up]** stanza of the WebSEAL configuration file controls this response. The default value is "no":

```
[step-up]
show-all-auth-prompts = no
```

Procedure

In previous versions of WebSEAL, multiple login prompts—one for each enabled authentication method—were presented to the unauthenticated user on one login page. To support this previous behavior, set the value of the **show-all-auth-prompts** stanza entry to "yes":

```
[step-up]
show-all-auth-prompts = yes
```

Note: The **show-all-auth-prompts** function is triggered only by a POP on an object. If an unauthenticated user is asked to authenticate for reasons that do not involve a POP on an object, the functionality of **show-all-auth-prompts** is not used.

Stepping up authentication at higher levels

About this task

You can configure WebSEAL to accept authentication mechanisms that are configured at a higher level than the level specified in the POP. With this configuration, the user can authenticate directly at the higher level.

Procedure

- To accept higher authentication levels during step-up operations, you must set the value of the **step-up-at-higher-level** stanza entry to "yes".

  ```
  [step-up]
  step-up-at-higher-level = yes
  ```

- To disallow higher authentication levels during step-up operations, set the value of the **step-up-at-higher-level** stanza entry to "no".

```
[step-up]
step-up-at-higher-level = no
```

Results

The default value is "no" if you do not configure this configuration entry. That is, by default WebSEAL does not accept higher authentication levels.

External authentication interface

Security Access Manager provides an external authentication interface that enables you to extend the functionality of the WebSEAL authentication process. The external authentication interface allows third-party systems to supply an authenticated identity to WebSEAL and Web-server plug-ins. The identity information is then used to generate a credential.

This extended authentication functionality is similar to the existing custom authentication module capability provided by the Web security external authentication C API. However, the external authentication interface allows the user identity to be supplied in HTTP Response headers rather than through the authentication module interface.

For complete information on configuring external authentication interface authentication, refer to “External authentication interface” on page 260.

External authentication interface for mobile applications

The standard EAI flow can involve multiple Moved Temporarily (302) operations, which are not mobile application friendly. WebSEAL provides configuration options to help reduce the number of 302 operations.

The following topics illustrate the various scenarios based on the authentication information transport mechanism.

Authentication through 401 WWW-Authenticate:

Use this scenario as an example where the authentication information is provided as an HTTP header, for example, basic-authentication.

This flow uses Local Response Redirects where requests to any page that requires an authenticated user is intercepted and redirected to the junctioned EAI application for authentication.

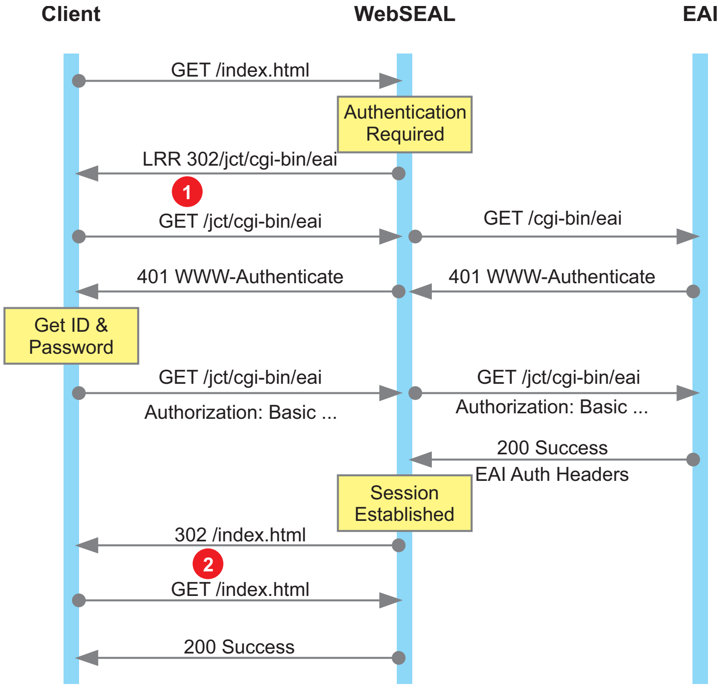
The junction that provides this EAI application must be created with **HTTP Basic Authentication Header** set to **ignore**.

Assume that the following configuration entries have been set in the WebSEAL configuration file.

```
[ba]
ba-auth = none

[eai]
eai-auth = https

[eai-trigger-urls]
trigger = /jct/cgi-bin/eai

[acnt-mgt]
enable-local-response-redirect = yes
```

```
[local-response-redirect]
local-response-redirect-uri = [login] /jct/cgi-bin/eai
```

The traditional flow is shown as follows:



Note: The `/index.html` file is just an example. It can be any document from WebSEAL or its junctions that require an authenticated session for access.

To configure WebSEAL to internally process 302 operations, first specify the maximum number of 302 operations it can sequentially follow. A value of 2 is suitable for typical scenarios:

```
[server]
maximum-followed-redirects = 2
```

Secondly, configure WebSEAL to process 302 redirects internally for any request that results in a Local Response Redirect with the following entry:

```
[server]
follow-redirects-for = !LRR!
```

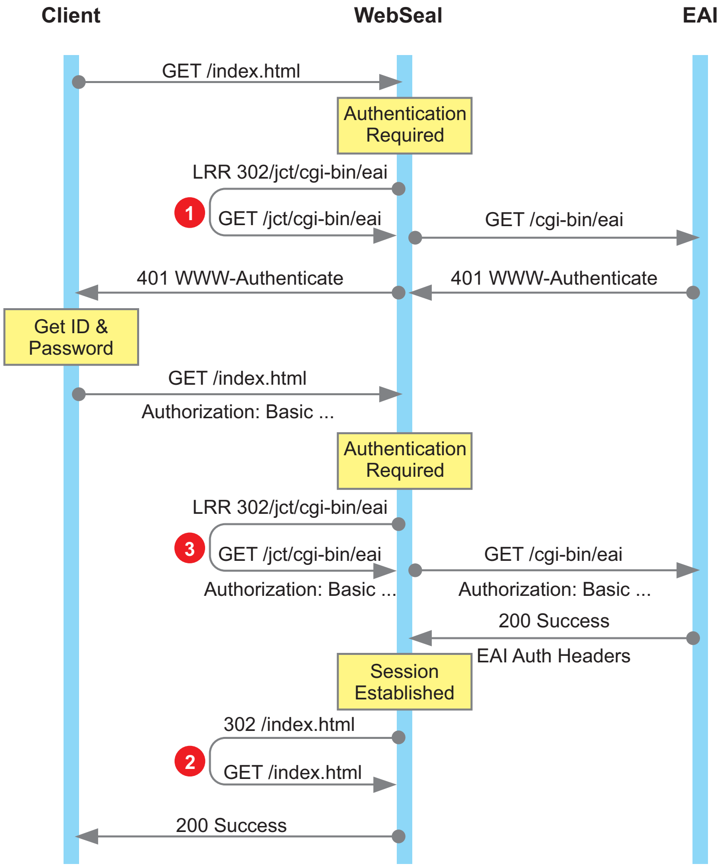
This configuration avoids the initial Local Response Redirect 302 (shown at point 1). The client now receives the 401 WWW-Authenticate header for `/index.html` instead of a redirect to the EAI URL. So the client sends the authorization header with a **GET** for `/index.html`.

Thirdly, set WebSEAL to process redirects internally for any request that would result in a 302 redirect back to the same URL that was originally requested (shown at point 2). To achieve this, add the following configuration entry:

```
[server]
follow-redirects-for = GET /jct/cgi-bin/eai*
```

The original request from the client was for `/index.html`. In this flow, WebSEAL internally redirects the client to `/index.html`. So the resulting page content that is returned from WebSEAL matches what the client is expecting to receive. Relative URLs in this page will operate as expected.

The final flow is shown as follows:



Note: Two sequential 302 operations are internally processed by WebSEAL (shown at 3 and 2) in the flow, which is why the value of 2 was used for the **maximum-followed-redirects** entry.

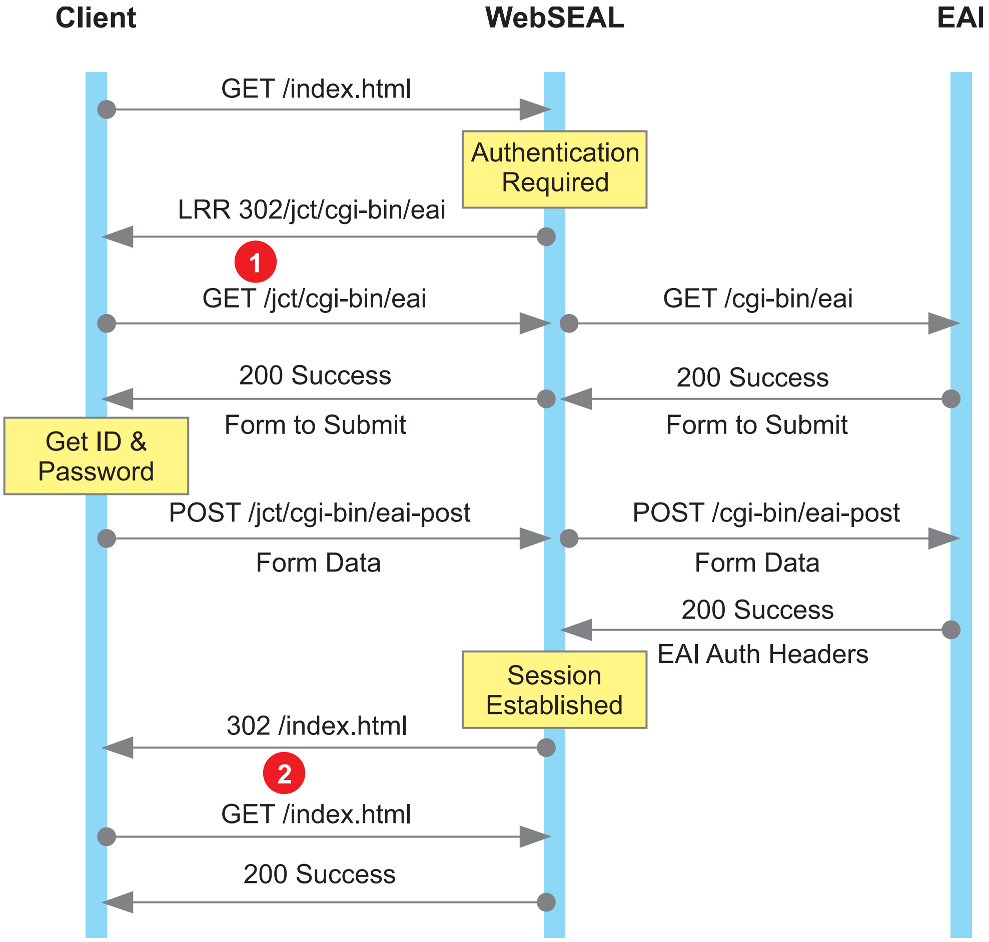
Authentication through forms:

Use this flow as an example to reduce 302 operations for forms authentication.

Assume that the following configuration entries have been set in the WebSEAL configuration file.

```
[eai]
eai-auth = https

[eai-trigger-urls]
trigger = /jct/cgi-bin/eai-post

[acnt-mgt]
enable-local-response-redirect = yes

[local-response-redirect]
local-response-redirect-uri = [login] /jct/cgi-bin/eai
```

The traditional flow is as follows:



Note: The `/index.html` file is just an example. It can be any document from WebSEAL or its junctions that require an authenticated session for access.

To configure WebSEAL to internally process 302 operations, first specify the maximum number of 302 operations it can sequentially follow. A value of 2 is suitable for typical scenarios:

```
[server]
maximum-followed-redirects = 2
```

Secondly, configure WebSEAL to process 302 redirects internally for any request that results in a Local Response Redirect with the following entry:

```
[server]
follow-redirects-for = !LRR!
```

This configuration avoids the initial Local Response Redirect 302 (shown at point 1).

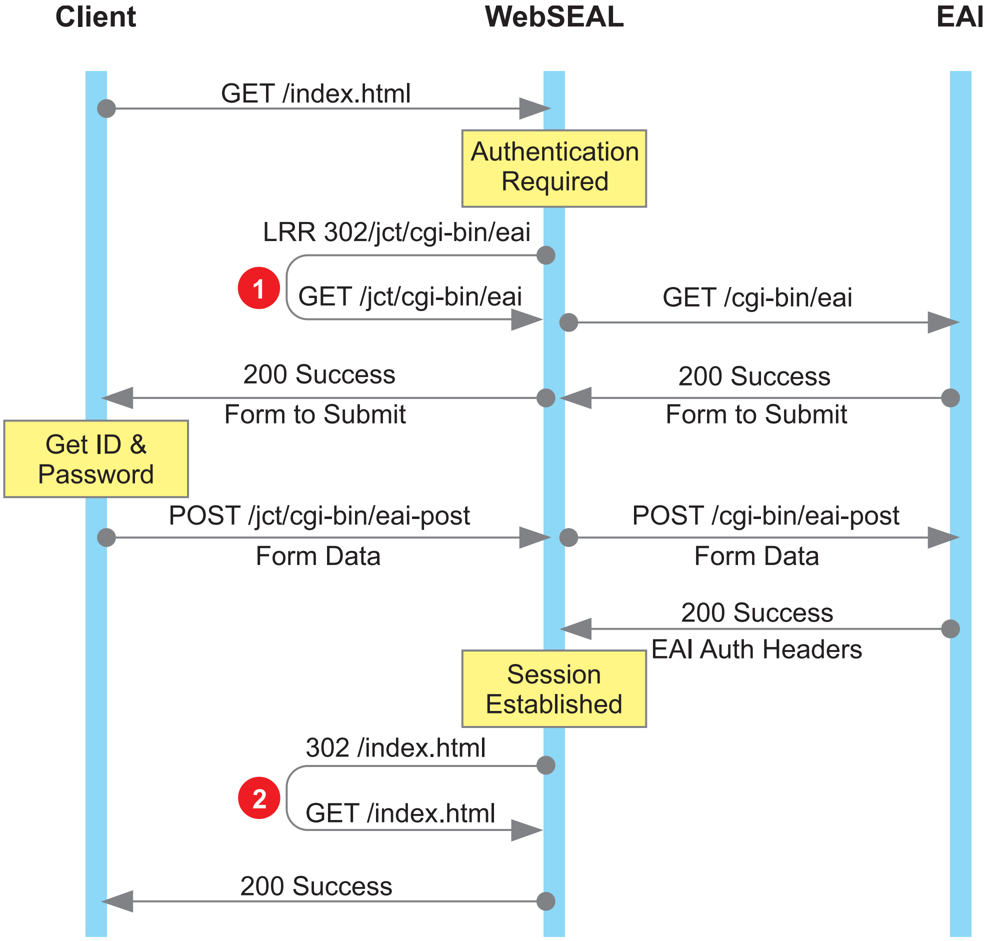
Note: Ensure that the browser does not cache the form that is returned from the EAI against the resource that is being accessed (`/index.html`). Also, as the browser believes that the request for `/index.html` returned the form, all page-relative URLs in the form are relative to `/index.html`, not`/jct/cgi-bin/eai`.

Thirdly, set WebSEAL to process redirects internally for any request that would result in a 302 redirect back to the same URL that was originally requested (shown at point 2). To achieve this, add the following configuration entry:

```
[server]
follow-redirects-for = POST /jct/cgi-bin/eai-post*
```

Note: WebSEAL returns the content of `/index.html` for what the browser believes to be the resource `/jct/cgi-bin/eai-post`. All page-relative URLs in `index.html` or any other URLs that were intercepted by the authentication are relative to `/jct/cgi-bin/eai-post` from the browser's point of view. Thus do not use any page-relative URLs if you use this technique. To solve this issue, it might be necessary to not remove the second 302 (shown at point 2).

The final flow is shown as follows:



Client Certificate User Mapping

The Client Certificate User Mapping functionality will be deprecated in a future release in favor of the Authenticated User Mapping functionality.

This section introduces the advanced Client Certificate User Mapping functionality, describes key concepts and components, and details how to deploy the functionality within a Security Access Manager environment.

Introduction

WebSEAL uses the Cross Domain Authentication Service (CDAS) to authenticate a user and provide a Security Access Manager user identity.

The client certificate user-mapping CDAS provides a mechanism by which WebSEAL can use the details of a client certificate to determine the corresponding Security Access Manager user identity. The rules that govern the mapping of the client certificate are defined in XSL style notation.

Note: If no rules file is provided, by default the Security Access Manager user identity is determined by the Subject DN from the certificate.

The CDAS supports all user registries that Security Access Manager supports.

The rules evaluation can return an LDAP search string. This string representation of the LDAP search filter must be in accordance with the format described in RFC 2254.

Example Rules:

The new CDAS gives the user more flexibility in mapping attributes contained within the certificate to the Security Access Manager user identity.

The following list illustrates some of the mapping functionality supported by this CDAS.

1. If issuer DN = X, and subject DN = Y, then Security Access Manager DN also = Y.
2. The certificate itself is stored as a **userCertificate** attribute on the **inetOrgPerson** entry, and a search is done for the Base64 encoded version of the certificate within the user registry.
3. Take the issuer DN and the subject DN from the certificate, and combine them to look like this:

   ```
   <certDN>subjectName</certDN><issuerDN>issuerName</issuerDN>
   ```

   Then look for an entry with this value for the attribute:

   ```
   ibm-certificateSubjectAndIssuer
   ```
4. If issuer DN = X, the **subjectAltName** is the same as the DN of the **inetOrgPerson** entry.
5. If issuer DN = X, the **serialNumber** maps to the **secCertSerialNumber** attribute of the **inetOrgPerson**.
6. If issuer DN = X, the cn from the **subjectDN** field will map to the cn of the **inetOrgPerson** entry.
7. If issuer DN = X, the **subjectDN** maps to **secCertDN** in the **inetOrgPerson** entry.

Certificate User Mapping Rule language:

Extensible Style Language (XSL) is the language used to specify rules, while Extensible Markup Language (XML) is the language used for the data that forms an input to the rules. The combination of XML and XSL provides a platform-independent way to express both the inputs to the rules evaluator and the rules themselves.

XML also provides the ability to express complex data types in a structured and standard manner in text format. This text format allows rules for processing the XML data to be written without having to cater to platform and programming language specifics.

XSL is a functional stylesheet language that can be used to perform simple or complex tasks, depending on your needs. XSL possesses an inherent ability to analyze and evaluate XML data, which is becoming the standard for data representation. XSL is built on other XML-based standards such as XPath, which is the expression language at the core of a Certificate User Mapping Rule.

To implement the user mapping rules, it is necessary to impose a number of constraints on the XSL rules, including the requirements that the output of the rule evaluation be simple text, and that the output conforms to one of a known set of result strings. For more information about the format and constraints of user mapping rules, see "Format and constraints of rules" on page 207.

UMI XML document model:

The Universal Management Infrastructure XML document model (or UMI XML model) is a set of restrictions placed on the XSL/XML model by the user mapping rules implementation, which enables the interface to be simple and yet functional for certificate purposes. The model constrains the certificate rules to function within a predetermined XML document format, with the same top-level XML document element for all rules. The XML UMI that is imported by the rules evaluator from certificate attributes must be inserted into this XML document before the data can be used by the certificate. Similarly, to simplify the process of defining rules, the certificate rules must operate within the confines of the UMI XML model.

The UMI XML model requires the XML document to contain the following top-level XML element, into which all target UMI for a particular rule evaluation is inserted. The `XMLUMI` element is created automatically as part of the rule evaluation process by the user mapping engine.

```
<XMLUMI>
<!--XML formatted UMI are inserted here.    -->
</XMLUMI>
```

As a result of this restriction, the XPath to the data used in a Certificate User Mapping Rule must include the prefix `/XMLUMI` in order to access a particular data element within the model. For example, if a UMI item of `stsuuser:STSUniversalUser` is added to the document, you must specify the XPath `/XMLUMI/stsuuser:STSUniversalUser` in order to access the data contained in the XML object `stsuuser:STSUniversalUser`.

An XPath is the path to a particular child element within the hierarchy of a structured XML data object. Much like a directory path on a hard drive is used to access a specific file, an XPath designation starts from the root of the document (in this case `/XMLUMI`) and traces a path from this root down through its child elements to the specific element that is being referenced. For example, using the example entitlement `stsuuser:STSUniversalUser` in the "XML certificate model" on page 204 as a reference, you would use the following XPath to access the `Version` element of `/XMLUMI/stsuuser:STSUniversalUser`:

```
"/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:
Attribute[@name='Version']/stsuuser:Value"
```

XPaths like this example are the means by which user mapping rules access the UMI data values that are needed to make attribute-based user mapping decisions.

Because all data elements are restricted to work within the UMI XML model, the user mapping rules must also be restricted to operate on or match XPaths within the model. Therefore, XSL template match statements are also restricted to matching XPaths starting from /XMLUMI within the UMI XML document. For additional information, see "Format and constraints of rules" on page 207.

Containers and XML UMI container names:

When data is requested from a resource manager, the granularity of the XML data returned is at the level of a single container of information. The container is normally also the smallest data element (for example, elements that might be considered for billing purposes). This convention also applies to the UMI XML model. The UMI that is used in user mapping rules is also defined and manipulated as containers of XML data. For example, the stsuuser:STSUniversalUser XML object defined in "XML certificate model" is an example of a UMI container.

The topmost element in the definition of an item of UMI is referred to as the *container name* of that item of UMI. When defining a Certificate User Mapping Rule, the XPath to the XML definition of data in any UMI container must always be referenced using the name of the container as the first element following /XMLUMI in the XPath specification for the data element.

Returning to the example UMI item stsuuser:STSUniversalUser, to access any element within the stsuuser:STSUniversalUser container, the XPath specification must be prefixed with stsuuser:STSUniversalUser. For example, "/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='SerialNumber']/stsuuser:Value" refers to the SerialNumber value. To access this information from within a Certificate User Mapping Rule, this XPath must also be prefixed by the top-level element of the XML target UMI input document, which is XMLUMI (for example, "/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='SerialNumber']/stsuuser:Value").

A template match statement can be used to remove the need to completely specify the entire path. The example rule file contains a template match statement of /XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList, which means that all attributes of the certificate can be specified without a prefix. For example, "/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='SerialNumber']/stsuuser:Value" is the same as "stsuuser:Attribute[@name='SerialNumber']/stsuuser:Value". For additional information, see "Format and constraints of rules" on page 207.

XML certificate model:

The following UMI XML document shows the data that is passed to the XSL processor from the rules evaluator during the evaluation of a Certificate User Mapping Rule.

The document contains one container named stsuuser. The attribute value of the container stsuuser:STSUniversalUser is defined in XML.

The certificate evaluator automatically encompasses all of the data under the XML top-level node declaration XMLUMI when the UMI XML document is created, so this top-level element is added for clarity.

The XML document is automatically created by the CDAS, based on the attributes available within the client certificate. The XML document that is passed to the evaluation routines by the user mapping rules evaluator is as follows:

```
<XMLUMI>
<stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
<stsuuser:Principal>
<stsuuser:Attribute name="name">
<stsuuser:Value>
-- Subject DN from certificate --
</stsuuser:Value>
</stsuuser:Attribute>
</stsuuser:Principal>
<stsuuser:AttributeList>
<stsuuser:Attribute name="--attr-name--" type="urn:ibm:security:gskit">
<stsuuser:Value>--attr-value--</stsuuser:Value>
</stsuuser:Attribute>
...
</stsuuser:AttributeList>
</stsuuser:STSUniversalUser>
</XMLUMI>
```

For example:

```
<?xml version="1.0" encoding='UTF-8'?>

<XMLUMI>
<stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
<stsuuser:Principal>
<stsuuser:Attribute name="name">
<stsuuser:Value>
CN=testuser,O=ibm,C=au
</stsuuser:Value>
</stsuuser:Attribute>
</stsuuser:Principal>
<stsuuser:AttributeList>
<stsuuser:Attribute name="SubjectDN" type="urn:ibm:security:gskit">
<stsuuser:Value>CN=testuser,O=ibm,C=au</stsuuser:Value>
</stsuuser:Attribute>

<stsuuser:Attribute name="IssuerDN" type="urn:ibm:security:gskit">
<stsuuser:Value>CN=ca,O=ibm,C=au</stsuuser:Value>
</stsuuser:Attribute>

<stsuuser:Attribute name="ValidFromEx" type="urn:ibm:security:gskit">
<stsuuser:Value>00:29:26 08-06-2009</stsuuser:Value>
</stsuuser:Attribute>

</stsuuser:AttributeList>
</stsuuser:STSUniversalUser>
</XMLUMI>
```

For a full list of available attributes, see “Valid certificate attributes” on page 210.

When referencing a particular UMI item within the XMLUMI document available to a rule, the XPath path specifier can begin from the container name of the XML element (for example, `stsuuser:STSUniversalUser`) . If the callers want to specify their own template match statement explicitly, they can do so.

For additional information, see “Format and constraints of rules” on page 207.

User mapping rules evaluator

The user mapping rules evaluator evaluates user mapping rules within the constraints that are required by the user mapping engine. Pre-configured rules are supplied in a configuration file to the new CDAS.

The user mapping rules evaluator takes the rule policy along with the XML representation of the certificate and passes this to the XSL processor for evaluation.

The input for the transformation is the XML version of the client certificate (as defined above). XSL transformation rules decide how the Security Access Manager user name is mapped from the supplied certificate information. Two inputs are used when making the decision:

- the XML representation of the client certificate, and
- the XSL rule, which determines how the XML is interpreted.

The output from the decision is a single string which is used to determine the Security Access Manager user identity.

The user mapping engine expects the rules evaluation to result in the return of one of the string identifiers listed below. These identifiers ensure uniqueness in the event that an XSL rule is written incorrectly and the evaluation returns incorrect information. Delimiting the identifiers with an exclamation point (!) enables the evaluator to identify errant cases.

The string must conform to one of the following definitions:

!*free format text***!**

Free format text, which could also include elements from the source XML. This string will be used as the Security Access Manager user identity. For example:

```
!cn=testuser,o=ibm,c=au!
!<xsl:value-of select="stsuuser:Attribute[@name='SerialNumber']/
stsuuser:Value"/>!
```

!userreg base='%*base***%' attr='%***name***%'!%ldap-search-filter%!**

Indicates that the user registry should be searched for the Security Access Manager user identity, based on the supplied search string. The **attr** value is used to define the name of the LDAP attribute which holds the Security Access Manager user identity. The search string should conform to RFC 2254. For example:

```
!userreg base='o=ibm,c=au' attr='cn'! (&amp;(objectClass=ePerson)
(serialNum=<xsl:value-of select="stsuuser:Attribute[@name=
'SerialNumber']/stsuuser:Value"/>))!
```

!no-matching-rule!

Indicates that no matching rule was found for the supplied client certificate. If this string is returned from the rule evaluation the CDAS will return an error. For example:

```
!no-matching-rule!
```

Note: The ampersand (&) character cannot be used in XSLT documents or it will produce errors when processed. This character must be transcribed as **&amp;**.

Format and constraints of rules:

A Certificate User Mapping Rule must be defined as an XSL template in an XSL stylesheet. The rule must be written in a valid XSL template rule format. It must return a text document that contains one of the string identifiers shown in “User mapping rules evaluator” on page 206.

The identifiers must be the only text in the output document, although they can be surrounded by white space. If a value other than the defined values or an empty document is returned, the user mapping fails and an error code is returned to the CDAS to indicate that the rule is not compliant.

The result of the XSL transformation performed by an XSL Certificate User Mapping Rule must be a text output document that contains only one of the supported string identifiers.

The following Certificate User Mapping Rule example references the XML data item that is defined in `stsuuser:STSUniversalUser`. The condition that the rule evaluates is expressed as follows:

```
<?xml version="1.0" encoding='UTF-8'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:stsuuser=
"urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

<!-- Required to constrain output of rule evaluation -->
<xsl:output method="text" omit-xml-declaration="yes" encoding='UTF=8' indent=
"no"/>

<!-- Need this to ensure default text node printing is off -->
<xsl:template match="text()"></xsl:template>

<!-- Let's make it easier by matching the constant part of our XML name -->
<xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList">

<!-- If this certificate was issued by our CA, just use the subject dn -->
<xsl:when test='stsuuser:Attribute[@name="IssuerDN"]/stsuuser:Value =
"cn=ca,o=ibm,c=au"'>
!<xsl:value-of select="stsuuser:Attribute[@name='SubjectDN']/
stsuuser:Value"/>!
</xsl:when>

<!-- If this certificate was issued by the tivoli CA, search for the
certificate serial number -->
<xsl:when test='stsuuser:Attribute[@name="IssuerDN"]/stsuuser:Value =
"cn=ca,o=tivoli,c=au"'>
!userreg base='o=ibm,c=us' attr='cn'!secCertSerialNumber=<xsl:
value-of select="stsuuser:Attribute[@name='SerialNumber']/
stsuuser:Value"/>!
</xsl:when>

<!-- Otherwise we don't have a matching rule.'no-matching-rule' is a
special string. -->
<xsl:otherwise>
!no-matching-rule!
</xsl:otherwise>

</xsl:template>

</xsl:stylesheet>
```

Note: Everything up to and including the template match is static for all rules. Remaining parts of the XSL rule can be customized.

To reference any data item in the document, the XPath to each node must include the XMLUMI node. When a rule is built, the rule writer must understand what the correct XPath is from the current point in the tree, in order to access the XML data nodes and subnodes. The current point in the tree is selected by using the template match statement. The template match statement allows an XSL programmer to shorten the XPath to each data element by specifying that the XPath processing must occur further down the XML document tree.

The `<xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList">` statement tells the XSL processor that all relative XPaths within the bounds of the template statement should be assumed to be relative to the node `"/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList"`. For example, `"/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='IssuerDN']/stsuuser:Value"` can be referred to as simply `"stsuuser:Attribute[@name='IssuerDN']/stsuuser:Value"`.

Examples of user mapping rules:

This section provides two examples of output XSLT evaluation, first using a free format text string and then searching the user registry for the user DN.

In the first example, the rule works on a string from the client certificate: if the string matches then it selects the Distinguished Name (DN) from the client certificate.

Rule:

```
<!-- Test a valid 'free format' string. -->
<xsl:when test='stsuuser:Attribute[@name="SubjectDN"]
/stsuuser:Value = "cn=testuser,o=ibm,c=au"'>
!<xsl:value-of select="stsuuser:Attribute[@name='SubjectDN']/
stsuuser:Value"/>!
</xsl:when>
```

Details from Client Certificate:

```
<stsuuser:STSUniversalUser xmlns:stsuuser=
"urn:ibm:names:ITFIM:1.0:stsuuser">
<stsuuser:Principal>
<stsuuser:Attribute name="name">
<stsuuser:Value>
CN=testuser,O=ibm,C=au
</stsuuser:Value>
</stsuuser:Attribute>
</stsuuser:Principal>
<stsuuser:AttributeList>
<stsuuser:Attribute name="SubjectDN" type=
"urn:ibm:security:gskit">
<stsuuser:Value>CN=testuser,O=ibm,C=au</stsuuser:Value>
</stsuuser:Attribute>
...
</stsuuser:AttributeList>
</stsuuser:STSUniversalUser>
```

String returned by CDAS:

```
CN=testuser,O=ibm,C=au
```

In the second example, the rule searches the user registry for an attribute and returns the user Common Name (CN) from the registry. In this case the search of the registry is on the e-mail address from the client certificate.

Rule:

```
<!-- Test a matching 'userreg' string. -->
<xsl:when test='stsuuser:Attribute[@name="SubjectDN"]
/stsuuser:Value = "cn=testuser3,o=ibm,c=au"'>
!userreg base='o=ibm,c=au' attr='cn'!(description=<xsl:value-of
select="stsuuser:Attribute[@name='SubjectEmail']/
stsuuser:Value"/>)!
</xsl:when>
```

Details from Client Certificate:

```
<stsuuser:STSUniversalUser xmlns:stsuuser=
"urn:ibm:names:ITFIM:1.0:stsuuser">
<stsuuser:Principal>
<stsuuser:Attribute name="name">
<stsuuser:Value>
cn=testuser3,o=ibm,c=au
</stsuuser:Value>
</stsuuser:Attribute>
</stsuuser:Principal>
<stsuuser:AttributeList>
<stsuuser:Attribute name="SubjectDN" type=
"urn:ibm:security:gskit">
<stsuuser:Value>cn=testuser3,o=ibm,c=au</stsuuser:Value>
</stsuuser:Attribute>
<stsuuser:Attribute name="SubjectEmail" type=
"urn:ibm:security:gskit">
<stsuuser:Value>testuser3@ibm.com</stsuuser:Value>
</stsuuser:Attribute>
...
</stsuuser:AttributeList>
</stsuuser:STSUniversalUser>
```

String returned from user registry:

```
cn=testuser3
```

How to manage the CDAS

This section describes how to add the CDAS into WebSEAL, and how to configure the CDAS.

Enabling the CDAS functionality:

You can enable the extended CDAS functionality.

About this task

To enable the extended CDAS functionality:

Procedure

1. You must update the **[cert-map-authn]** stanza in the WebSEAL configuration file as follows:

   ```
   [cert-map-authn]
   rules-file = file
   debug-level = level
   ```

   where:

   file The name of the rules file for the certificate mapping CDAS to use.

   level Controls the trace level for the module.

   For example:

   ```
   [cert-map-authn]
   rules-file = cert-rules.txt
   debug-level = 5
   ```

Note: The *level* variable indicates the trace level, with *1* designating a minimal amount of tracing and *9* designating the maximum amount of tracing. You can also use the Security Access Manager **pdadmin** trace commands to modify the trace level by using the trace component name of `pd.cas.certmap`. This trace component is only available after the first HTTP request is processed.

2. You can use the Local Management Interface (LMI) to modify the rules file (for example, `cert-rules.txt`) as required:
   a. Select **Secure Web Settings** > **Global Settings** > **Client Certificate Mapping** from the top menu. The Client Certificate Mapping management page displays.
   b. *(Optional)* If no rules files exist, you can click **New** to create a new rules file. Enter a name for the new file such as `cert-rules.txt` and click **Save**. A new file is generated that is based on the default template.
   c. Click the file that you want to manage, such as `cert-rules.txt`, from the available list of File Names.
   d. Click **Edit**.
   e. Update the file.
   f. Click **Save**.

Valid certificate attributes:

The client certificate attributes that are made available to the mapping rules are defined by the GSKit toolkit.

For WebSEAL, this can be any of the following attributes:

```
* Base64Certificate
* SerialNumber
* SubjectCN
* SubjectLocality
* SubjectState
* SubjectCountry
* SubjectOrganization
* SubjectOrganizationalUnit
* SubjectDN
* SubjectPostalCode
* SubjectEmail
* SubjectUniqueID
* IssuerCN
* IssuerLocality
* IssuerState
* IssuerCountry
* IssuerOrganization
* IssuerOrganizationUnit
* IssuerDN
* IssuerPostalCode
* IssuerEmail
* IssuerUniqueID
* Version
* SignatureAlgorithm
* ValidFrom
* ValidFromEx
* ValidTo
* ValidToEx
* PublicKeyAlgorithm
* PublicKey
* PublicKeySize
* FingerprintAlgorithm
* Fingerprint
* BasicConstraintsCA
* BasicConstraintsPathLength
* DerCertificate
* CertificatePolicyID
* CRLDistributionPoints
* DerSubjectDN
* DerIssuerDN
* KeyUsage
* AlternativeDirectoryName
* AlternativeDNSName
* AlternativeIPAddress
* AlternativeURI
* AlternativeEmail
```

For more information about GSKit, see the IBM *Secure Sockets Layer Introduction and iKeyman User's Guide*.

Configuring WebSEAL to use the certificate mapping module

Procedure

1. Check the `accept-client-certs` entry within the `[certificate]` stanza. The value of this entry should be `required`, `optional`, or `prompt_as_needed`. This determines if and when a user is prompted to provide a client certificate by the browser.

   For more details about the different values for this entry, see "Client-side certificate authentication modes" on page 128.
2. Save the configuration file.
3. Restart WebSEAL to implement the updates.

Constructing the XSLT rules file:

You can use the mapping module to define flexible rules that allow mapping of certificate attributes to a user identity. The user identity can be a Security Access Manager user ID. For example, `testuser`. Alternatively, the user identity can be the user DN as found in the registry. For example, `cn=testuser, o=ibm,c=au`.

Upon receiving a user certificate, the module creates an XML document that lists all of its attributes. The XML document conforms to the Universal Management Infrastructure (UMI) XML document model. For example, the module could create a document that looks like this:

```
<?xml version="1.0" encoding='UTF-8'?>

<XMLUMI>
  <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
    <stsuuser:Principal>
      <stsuuser:Attribute name="name">
        <stsuuser:Value>
          CN=testuser,O=ibm,C=au
        </stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:Principal>
    <stsuuser:AttributeList>
      <stsuuser:Attribute name="SubjectDN" type="urn:ibm:security:gskit">
        <stsuuser:Value>CN=testuser,O=ibm,C=au</stsuuser:Value>
      </stsuuser:Attribute>

      <stsuuser:Attribute name="IssuerDN" type="urn:ibm:security:gskit">
        <stsuuser:Value>CN=ca,O=ibm,C=au</stsuuser:Value>
      </stsuuser:Attribute>

      <stsuuser:Attribute name="ValidFromEx" type="urn:ibm:security:gskit">
        <stsuuser:Value>00:29:26 08-06-2009</stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:AttributeList>
  </stsuuser:STSUniversalUser>
</XMLUMI>
```

The XSLT file defines how to transform the XML document. The result of the transformation must be in one of these forms:

- *!identifier!*

  A Security Access Manager user ID or user DN. This form is used when no registry search is required, such as when the identifier can be retrieved directly from the certificate.
- `!userreg base='`*baseDN*`' attr='`*attrName*`'!` *ldapSearchFilter !*

  baseDN is the base distinguished name, *attrName* is the LDAP attribute name that corresponds to a user identity, and *ldapSearchFilter* is the LDAP search filter.

  This form is used when certificate information is used to search the registry for the corresponding user. You can also use the module with Active Directory.
- `!no-matching-rule!`

  This form indicates that you cannot use any rule to find the required information to authenticate the user.

The following example is the `cert-rules-template.txt` file installed in the `$WEBRTE_HOME/etc/ directory`:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

    <!-- Required to constrain output of rule evaluation -->
    <xsl:output method="text" omit-xml-declaration="yes" encoding='UTF=8'
indent="no"/>

    <!-- Need this to ensure default text node printing is off -->
    <xsl:template match="text()"></xsl:template>

    <!-- Let's make it easier by matching the constant part of our XML name -->
    <xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:
AttributeList">
        !<xsl:value-of select="stsuuser:Attribute[@name='SubjectDN']/stsuuser:
Value"/>!
    </xsl:template>

</xsl:stylesheet>
```

This XSLT document transforms the UMI XML document created by the authentication module and outputs the subject DN of the certificate it receives between ! characters. For example, `!cn=testuser,o=ibm,c=au!`.

In this case, no user registry search is performed. This `<xsl:output>` element is required to indicate that text, not an XML document, is the output of the transformation.

This first `<xsl:template>` element ensures that any remaining text nodes in the document are not copied to the output.

The next example shows how to direct the mapping module to perform a user registry, for example, LDAP, search with data from the certificate. It extracts the value of the SubjectEmail attribute from the certificate and searches for a user with an LDAP mail attribute equal to this address.

The base DN for the search is o=ibm,c=au. The value of the cn attribute is printed in the output. In this case, the result will probably be a Security Access Manager user ID, rather than a user DN.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

    <!-- Required to constrain output of rule evaluation -->
    <xsl:output method="text" omit-xml-declaration="yes" encoding='UTF=8'
indent="no"/>

    <!-- Need this to ensure default text node printing is off -->
    <xsl:template match="text()"></xsl:template>

    <!-- Let's make it easier by matching the constant part of our XML name -->
    <xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:
AttributeList">
        !userreg base='o=ibm,c=au' attr='cn'!(mail=<xsl:value-of
    select="stsuuser:Attribute[@name='SubjectEmail']/stsuuser:
Value"/>)!
    </xsl:template>

</xsl:stylesheet>
```

Note: Because the & character is used in XML to demarcate the start of an entity, it cannot be used as is in the LDAP search filter. If an LDAP query contains multiple terms that need to be joined, the &amp; entity needs to be used instead.

Validating a successful module mapping:

To confirm a successful module mapping for users, ensure that a Security Access Manager policy is set for a protected resource to refuse unauthenticated users and allow authenticated ones.

Before you begin

Before accessing this resource in a browser, ensure that the client certificate is imported into the browser. See your browser help for instructions on how to import.

Procedure

1. When attempting to access a protected resource, the browser prompts you to select a client certificate. Select the client certificate which you just imported into the browser. The WebSEAL log contains trace messages pertinent to the mapping module, indicating success or failure, assuming that the debugging level was set appropriately in the WebSEAL configuration file as described in "Configuring WebSEAL to use the certificate mapping module" on page 211.
2. The result of the XSLT transformation dictates whether the mapping module must perform a user registry search or not. The mapping module conducts a user registry search if the result is in the following form:

   `!userreg base='`*`baseDN`*`' attr='`*`attrName`*`' ! `*`ldapSearchFilter`*`!`

   Otherwise, the mapping module does not conduct a user registry search. The result of the search is a Security Access Manager user ID or a DN of a user.

If the mapping is successful and a search was performed in the user registry, the following message displays in the WebSEAL log. The WebSEAL log is typically at /var/pdweb/log/msg_webseal-*instance*.log on UNIX machines. It is at C:\Program Files\Tivoli\PDWeb\log\msg_webseal-*instance*.log on Windows machines:

```
2012-06-07-16:37:11.113+10:00I----- thread(2) trace.pd.cas.certmap:5
/sandbox/amwebrte611/src/pdwebrte/authn/modules/certmapauthn/
AMWCertLDAPUserRegistry.cpp:146: ISAM user identity: testuser
```

If no search was performed in the registry, only a message similar to the following is displayed:

```
2012-06-07-18:34:29.200+10:00I----- thread(2) trace.pd.cas.certmap:3
/sandbox/amwebrte611/src/pdwebrte/authn/modules/certmapauthn/
AMWCertRulesEngine.cpp:219: result: CN=testuser,O=IBM,C=AU
```

Authenticated User Mapping

You can use the authenticated user mapping module to map an authenticated user name to a different Security Access Manager user identity.

During the authentication process, Security Access Manager takes an XML representation of the authentication data and then evaluates the data against an XSLT rule to produce the appropriate user identity. The result is either a static user identity or an LDAP search string that you can use to locate the user identity. In addition, one or more attributes might be added to the generated credential for the user.

Note: The authenticated user mapping module cannot be invoked if an EAI authentication takes place, where a privileged attribute certificate (PAC) is supplied as the authentication data.

Authenticated user mapping rule language

Extensible Style Language (XSL) is the language that specifies rules. Extensible Markup Language (XML) is the language for the data that forms an input to the rules. The combination of XML and XSL provides a platform-independent way to express both the inputs to the rules evaluator and the rules themselves.

XML expresses complex data types in a structured and standard manner in text format. Using this text format, you can write processing rules for the XML data that is independent of operating systems and programming languages.

XSL is a functional style sheet language that can perform simple or complex tasks. XSL possesses an inherent ability to analyze and evaluate XML data, which is becoming the standard for data representation. XSL is built on other XML-based standards such as XPath, which is the expression language at the core of an authenticated user mapping rule.

To implement the user mapping rules, it is necessary to impose some constraints on the XSL rules. These constraints include the requirements that the output of the rule evaluation conforms to one of a known set of result strings. For more information about the format and constraints of user mapping rules, see "Format and constraints of rules" on page 218.

UMI XML document model

The Universal Management Infrastructure XML document model (UMI XML model) is a set of restrictions that are placed on the XSL or XML model by the user mapping rules implementation. This model enables the interface to be simple and yet functional for user mapping purposes.

The model constrains the rules to function in a predetermined XML document format with the same top-level XML document element for all rules. The XML UMI that is imported by the rules evaluator from user attributes must be inserted into this XML document before the data can be used. Similarly, to simplify the process of defining rules, the rules must operate in the confines of the UMI XML model.

The UMI XML model requires the XML document to contain the following top-level XML element into which all target UMI for a particular rule evaluation is inserted. The `XMLUMI` element is created automatically as part of the rule evaluation process by the user mapping engine.

```
<XMLUMI>
<!--XML formatted UMI are inserted here.    -->
</XMLUMI>
```

As a result of this restriction, the XPath to the data in a rule must include the prefix `/XMLUMI` to access a particular data element within the model. For example, if you add a UMI item of `stsuuser:STSUniversalUser` to the document, you must specify the XPath `/XMLUMI/stsuuser:STSUniversalUser` to access the data in the XML object `stsuuser:STSUniversalUser`.

An XPath is the path to a particular child element in the hierarchy of a structured XML data object. It is similar to the mechanism that a directory path on a hard disk uses to access a specific file. An XPath designation starts from the root of the document and traces a path from this root through its child elements to the specific element that is referenced. For example, to use the example entitlement `stsuuser:STSUniversalUser` in the "XML user mapping model" on page 216, the following XPath accesses the `address` element of `/XMLUMI/stsuuser:STSUniversalUser`:

```
"/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:
Attribute[@name='address']/stsuuser:Value"
```

XPaths like this example are the means by which user mapping rules access the UMI data values for attribute-based user mapping decisions.

Because all data elements are restricted to work in the UMI XML model, the user mapping rules must also be restricted to operate on or match XPaths in the model. Therefore, XSL template match statements are also restricted to matching XPaths starting from `/XMLUMI` in the UMI XML document. For more information, see "Format and constraints of rules" on page 218.

Containers and XML UMI container names

When data is requested from a resource manager, the granularity of the returned XML data is at the level of a single container of information. The container is normally also the smallest data element. For example, elements that might be considered for billing purposes.

This convention also applies to the UMI XML model. The UMI that is used in user mapping rules is also defined and manipulated as containers of XML data. For example, the `stsuuser:STSUniversalUser` XML object that is defined in "XML user mapping model" on page 216 is an example of a UMI container.

The topmost element in the definition of an item of UMI is the *container name* of that item. When you define an authenticated user mapping rule, the XPath to the XML definition of data in any UMI container must be referenced with the name of the container as the first element after /XMLUMI.

To access any element in the stsuuser:STSUniversalUser container in the UMI item stsuuser:STSUniversalUser example, prefix the XPath specification with stsuuser:STSUniversalUser. For example, "/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='username']/stsuuser:Value" refers to the username value.

To access this information from an authenticated user mapping rule, prefix this XPath with the top-level element of the XML target UMI input document, which is XMLUMI. For example, "/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='username']/stsuuser:Value".

The example rule file contains a template match statement of /XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList, which means that you can specify all attributes of the user mapping file without a prefix. For example, "/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:Attribute[@name='username']/stsuuser:Value" is the same as "stsuuser:Attribute[@name='username']/stsuuser:Value". For more information, see “Format and constraints of rules” on page 218.

XML user mapping model

The following UMI XML document shows the data that is passed to the XSL processor from the rules evaluator during the evaluation of an authenticated user mapping rule.

The document contains one container that is named stsuuser. The attribute value of the container stsuuser:STSUniversalUser is defined in XML.

The evaluator automatically encompasses all of the data under the XML top-level node declaration XMLUMI when the UMI XML document is created, so this top-level element is added for clarity.

The XML document is automatically created by Security Access Manager with the attributes that are in the authentication request. The XML document that is passed to the evaluation routines by the user mapping rules evaluator is as follows:

```
<?xml version="1.0" encoding='UTF-8'?>

<XMLUMI>
  <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
    <stsuuser:Principal>
      <stsuuser:Attribute name="name">
        <stsuuser:Value>
          - authenticated user identity -
        </stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:Principal>
    <stsuuser:AttributeList>
      <stsuuser:Attribute name="-attrname-">
        <stsuuser:Value>-attrvalue-</stsuuser:Value>
      </stsuuser:Attribute>
      ...
    </stsuuser:AttributeList>
  </stsuuser:STSUniversalUser>
</XMLUMI>
```

For a full list of available attributes, see “Valid user mapping attributes” on page 222.

When you reference a particular UMI item in the XMLUMI document available to a rule, the XPath path specifier can begin from the container name of the XML element (for example, `stsuuser:STSUniversalUser`). If the callers want to specify their own template match statement explicitly, they can do so.

For more information, see “Format and constraints of rules” on page 218.

User mapping rules evaluator

The user mapping rules evaluator evaluates user mapping rules in the constraints that are required by the user mapping engine. A configuration file that you specify supplies the pre-configured rules to the authenticated user mapping module.

The user mapping rules evaluator sends the rule policy with the XML representation of the authentication request to the XSL processor for evaluation.

The input for the transformation is the XML version of the authentication request. XSL transformation rules decide how the Security Access Manager user name is mapped from the supplied user attributes information. Two inputs are used for decision making:

- The XML representation of the authentication request.
- The XSL rule, which determines how the XML is interpreted.

The output from the decision determines the Security Access Manager user identity and attributes.

Table 34. Evaluation output

XML Element	Description	Example
`<identity>`	Contains the new user identity. The original user identity, as determined by the authentication module, is used if no identity container is provided. If an identity container is provided but the identity cannot be determined, it is an authentication error. An example of this situation is a failed LDAP search.	`<identity>cn=testuser,o=ibm,c=us</identity>`

Table 34. Evaluation output (continued)

XML Element	Description	Example
`<attribute>`	Contains the name and value of an attribute that is added to the constructed credential. You can supply multiple attributes in the response. **Note:** Use a unique name for each attribute that you want to add to the user credential. If you add more than one attribute with the same name in the XSLT rules, only one of the attributes with that name is added to the user credential.	`<attribute name=qop>test-qop</attribute>`

The value for the identity or attribute can either be a static string or an LDAP search query. The LDAP search query uses the following format:

```
<userreg base='%base%' attr='%name%'>%ldap-search-filter%</userreg>
```

Format and constraints of rules:

Define an authenticated user mapping rule as an XSL template in an XSL style sheet.

Write the rule in a valid XSL template rule format by using XSLT version 1.0. It must return a text document that contains the string identifiers in "User mapping rules evaluator" on page 217.

The following authenticated user mapping rule example references the XML data that is defined in `stsuuser:STSUniversalUser` to add an attribute that is called `authn_method` to the user credential.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:stsuuser=
"urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

<!-- Required to constrain output of rule evaluation -->
<xsl:output method="text" omit-xml-declaration="yes" encoding='UTF=8' indent="no"/>

<!-- Need this to ensure default text node printing is off -->
<xsl:template match="text()"></xsl:template>

<!-- Let's make it easier by matching the constant part of our XML name -->
<xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList">
    <attribute name='authn_method'><xsl:value-of select="stsuuser:Attribute
    [@name='method']/stsuuser:Value"/></attribute>
</xsl:template>

</xsl:stylesheet>
```

Sample user mapping rule:

This example XSLT rule sets a new user identity that is derived from an LDAP search and two new attributes.

```
<?xml version="1.0" encoding='UTF-8'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

<!-- Required to constrain output of rule evaluation -->
<xsl:output method="xml" omit-xml-declaration="yes" encoding='UTF=8' indent="no"/>

<!-- Need this to ensure default text node printing is off -->
<xsl:template match="text()"></xsl:template>

<!-- Let's make it easier by matching the constant part of our XML name -->
<xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList">
 <attribute name='method'><xsl:value-of select="stsuuser:Attribute
 [@name='method']/stsuuser:Value"/></attribute>
 <attribute name='user_email'><userreg base='dc=iswga' attr='email'>
 (cn=<xsl:value-of select="stsuuser:Attribute[@name='username']/stsuuser:Value"/>)
 </userreg></attribute>
 <identity><userreg base='dc=iswga' attr='dn'>
 (cn=<xsl:value-of select="stsuuser:Attribute[@name='username']/stsuuser:Value"/>)
 </userreg></identity>
</xsl:template>

</xsl:stylesheet>
```

The first attribute `method` contains the method that was used for authentication. This information is extracted from the XML representation of the authentication data.

The second attribute that the rule adds to the user credential is `user_email`. This attribute is populated by a user registry search. A search is performed, where the `CN` in the user registry is matched against the supplied user name. The `email` attribute is then returned for the matched user.

An LDAP search is also performed, where the `CN` in the user registry is matched against the supplied user name. The `dn` attribute of the matched user is then used as the Security Access Manager user identity.

Troubleshooting user mapping rule problems:

An error in the XSLT rule might cause the authentication to fail. The following table details some of the common problems in the XSLT rules and the corresponding failure scenarios.

Table 35. Typical user mapping rule problems and failure scenarios

Rule problem	Example	Expected failure
Invalid XSLT	A missing slash (/) in the closing tag	`HPDIA0110E An authentication mechanism module specific error occurred.` The logged details also include the output from the `SAXParseException`. For example, `SAXParseException: Expected end of tag 'identity'.`

Table 35. Typical user mapping rule problems and failure scenarios (continued)

Rule problem	Example	Expected failure
Referencing an XML attribute that does not exist	<attribute name='TESTAttr'> <xsl:value-of select= "stsuuser:Attribute[@name= '**notexist**']/stsuuser :Value"/> </attribute> <identity> <xsl:value-of select= "stsuuser:Attribute [@name='**invalid**']/ stsuuser:Value"/> </identity>	Any references to XML attributes that do not exist are disregarded by the user mapping module. In this example, the authentication proceeds as the original authenticated user. No identity mapping takes place because the identity container references a non-existent XML attribute. Similarly, the user credential does not contain an attribute that is called **TESTAttr** because the attribute container references an XML attribute that does not exist.
Referencing an unknown LDAP attribute in the rule	<identity> <userreg base='dc=iswga' attr='dn'> (**unknownLDAPattr** =<xsl:value-of select= "stsuuser:Attribute [@name='username'] /stsuuser:Value"/>) </userreg> </identity>	1 2014-07-29-13:28:23.110+10:00I----- 0x1005B308 webseald ERROR acl mgmt AMWUserMapLDAPUserRegistry.cpp 157 0x7f830cc74700 -- HPDAC0776E The DN specified was not found in the registry. 2 2014-07-29-13:28:23.111+10:00I----- 0x1005B3B5 webseald ERROR acl authzn usermapauthn.cpp 482 0x7f830cc74700 -- HPDAC0949E Validation of the rule text for rule object "usermapauthn" failed. Error code 0x1005b3b4 was returned along with error message "<identity xmlns:stsuuser="urn:ibm :names:ITFIM:1.0:stsuuser"> <userreg base="dc=iswga"attr="dn"> (unknownLDAPattr=userA)</userreg> </identity>". 3 2014-07-29-13:28:23.111+10:00I----- 0x13212065 webseald WARNING ias general pdauthn.cpp 1813 0x7f830cc74700 -- HPDIA0101E An unexpected error code was encountered.
Returning an LDAP search string that does not have a matching entry in LDAP	<identity> <userreg base='dc=iswga' attr='cn'>(description= **DescThatDoesNotExistInLDAP**) </userreg> </identity>	1 2014-07-29-13:31:58.256+10:00I----- 0x1005B308 webseald ERROR acl mgmt AMWUserMapLDAPUserRegistry.cpp 157 0x7f61fc4c5700 --HPDAC0776E The DN specified was not found in the registry. 2 2014-07-29-13:31:58.256+10:00I----- 0x1005B3B5 webseald ERROR acl authzn usermapauthn.cpp 482 0x7f61fc4c5700 -- HPDAC0949E Validation of the rule text for rule object "usermapauthn" failed. Error code 0x1005b3b4 was returned along with error message "<identity xmlns:stsuuser="urn:ibm :names:ITFIM:1.0:stsuuser"> <userreg base="dc=iswga" attr="cn"> (description=DescThatDoesNotExistInLDAP) </userreg></identity>". 3 2014-07-29-13:31:58.257+10:00I----- 0x13212065 webseald WARNING ias general pdauthn.cpp 1813 0x7f61fc4c5700 -- HPDIA0101E An unexpected error code was encountered.

Table 35. Typical user mapping rule problems and failure scenarios (continued)

Rule problem	Example	Expected failure
Return a static string identity that is not a valid IBM Security Access Manager user	`<identity>`**`does-not-exist`**`</identity>`	`HPDIA0219W   An unknown user, <login username>, was presented to Security Access Manager.` In this instance, the `login username` that is mentioned in the log might be a valid IBM Security Access Manager user. However, the XSLT rules attempt to map the user to an unknown IBM Security Access Manager user. So the authentication fails.

Enabling authenticated user mapping

The authenticated user mapping module is disabled by default. You must enable it before you can map an authenticated user name to a different Security Access Manager user identity.

Procedure

1. Access the local management interface to configure an XSLT rules file to define the rules for the authenticated user mapping module. The following steps use `auth-rules.xsl` as an example.
2. Select **Secure Web Settings** > **Global Settings** > **User Name Mapping** from the top menu. The User Name Mapping management page displays.
3. Take one of the following actions:
   - If rules files exist, select the file that you want to enable, such as `auth-rules.xsl`, from the available list of **File Names**.
   - If no rules files exist:
     a. Click **New** to create a new rules file.
     b. Enter a name for the new file such as `auth-rules.xsl`.
     c. Click **Save**. The system generates a new file that is based on the default template.
4. Click **Edit**.
5. Update the file to reflect the rules you want to set.
6. Click **Save**.
7. Access the WebSEAL configuration file for your instance.
8. Update the **`[user-map-authn]`** stanza in the WebSEAL configuration file as follows:

   ```
   [user-map-authn]
   rules-file = file
   debug-level = level
   ```

   where:

   file Specifies the name of the rules file for the authenticated user mapping module.

   Note: If you rename the rules file that WebSEAL is configured to use, you must manually update this *rules-file* configuration entry to match the new name of the rules file. Otherwise, WebSEAL cannot locate and use the rules file after the rename operation.

   level Controls the trace level for the module.

```
[user-map-authn]
rules-file = auth-rules.xsl
debug-level = 5
```

Notes:

- The *level* variable indicates the trace level; 1 designates a minimal amount of tracing, and 9 designates the maximum. The Security Access Manager **pdadmin** trace command also modifies the trace level by using the trace component name of pd.cas.usermap. This trace component is only available after the first HTTP request is processed.
- If you do not specify a debug level, 0 is used by default.
- If an invalid numerical value is configured, the module defaults to a level of 0.
- If a non-numeric value is configured for the entry, the authentication mechanism module fails with an error in the WebSEAL log, which indicates that the debug-level is invalid.

Valid user mapping attributes

A list of authenticated user mapping attributes can be used in the mapping rules.

The following table lists the available attributes.

Table 36. Valid user attributes

Attribute	Description
address	The address of the client that originates the authentication request.
qop	A string that represents the quality-of-protection of the incoming request.
browser	An identifier for the browser that originates the request.
method	A string that identifies the method that is used to authenticate the user.
attr:*<xxx>*	Any extended attributes that are provided by the authentication mechanism.

If the selected authentication method requires a user name and password, the following extended attributes are available:

Table 37. Extended attributes - user name and password

Attribute	Description
username	The name of the user during the authentication.
password	The password that is used during the authentication. The value of this attribute is masked in the associated logged output for security reasons.

Note: You can configure an External Authentication Interface (EAI) for WebSEAL authentications. If the EAI returns a username value, you can use the authenticated user mapping function. However, EAIs that return an Extended Privilege Attribute Certificate (EPAC) cannot use this function.

If the selected authentication method requires an SSO token, the following extended attributes are available.

Table 38. Extended attributes - SSO token

Attribute	Description
query	The query string from the request.
referer	The referer header from the request.
token_type	The type of token. The value can be auth, ecc, vft.

If the selected authentication method requires a certificate, the following extended attributes are available.

Table 39. Extended attributes - certificate

Attribute	Description
x509.base64_certificate	A base64 encoded representation of the certificate.
x509.basic_constraints_ca	The constraints that are associated with the CA who issued the certificate.
x509.basic_constraints_path_len	The depth of valid certification paths that include this certificate.
x509.certificate_policy_id	An identifier that names the policy that is acceptable to the certificate user.
x509.crl_distribution_points	The distribution points for the CRL information.
x509.der_certificate	A DER encoded representation of the certificate.
x509.fingerprint	The fingerprint that is associated with the certificate.
x509.fingerprint_algorithm	The algorithm that is used to generate the fingerprint that is associated with the certificate.
x509.issuer_cn	The common name of the issuer of the certificate.
x509.issuer_country	The country identifier that is associated with the issuer of the certificate.
x509.issuer_dn	The full domain name of the issuer of the certificate.
x509.issuer_dn_der	A DER encoded representation of the domain name of the issuer of the certificate.
x509.issuer_email	The email address that is associated with the issuer of the certificate.
x509.issuer_locality	The locality that is associated with the issuer of the certificate.
x509.issuer_org	The name of the organization that is associated with the issuer of the certificate.
x509.issuer_org_unit	The name of the organizational unit that is associated with the issuer of the certificate.
x509.issuer_postal_code	The postal code of the issuer of the certificate.
x509.issuer_state	The name of the state that is provided by the issuer of the certificate.

Table 39. Extended attributes - certificate (continued)

Attribute	Description
`x509.issuer_unique_id`	A unique identifier for the issuer of the certificate.
`x509.key_usage`	Defines the purpose of the key that is contained in the certificate.
`x509.public_key`	The public key that is used by the certificate.
`x509.public_key_algorithm`	The key algorithm that is used by the certificate.
`x509.public_key_size`	The size of the public key.
`x509.serial_number`	The serial number that is associated with the certificate.
`x509.signature_algorithm`	The algorithm that is used to generate the certificate signature.
`x509.subject_alternative_dirname`	A directory name that is associated with the subject of the certificate.
`x509.subject_alternative_dnsname`	A DNS name that is associated with the subject of the certificate.
`x509.subject_alternative_email`	The email address that is associated with the subject of the certificate.
`x509.subject_alternative_ipaddr`	The IP address that is associated with the subject of the certificate.
`x509.subject_alternative_uri`	A URI that is associated with the subject of the certificate.
`x509.subject_cn`	The common name of the subject of the certificate.
`x509.subject_country`	The country identifier that is associated with the subject of the certificate.
`x509.subject_dn`	The full domain name of the subject of the certificate.
`x509.subject_dn_der`	A DER encoded representation of the domain name of the subject of the certificate.
`x509.subject_email`	The email address that is associated with the subject of the certificate.
`x509.subject_locality`	The locality that is associated with the subject of the certificate.
`x509.subject_org`	The name of the organization that is associated with the subject of the certificate.
`x509.subject_org_unit`	The name of the organizational unit that is associated with the subject of the certificate.
`x509.subject_postal_code`	The postal code of the subject of the certificate.
`x509.subject_state`	The name of the state that is provided by the subject of the certificate.
`x509.subject_unique_id`	A unique identifier for the subject of the certificate.
`x509.valid_from`	The date from which the certificate is valid. The date is the number of seconds since epoch.

Table 39. Extended attributes - certificate (continued)

Attribute	Description
x509.valid_from_ex	The date from which the certificate is valid. The date format is hh:mm:ss dd-mm-yyyy.
x509.valid_to	The date to which the certificate is valid. The date is the number of seconds since epoch.
x509.valid_to_ex	The date to which the certificate is valid. The date format is hh:mm:ss dd-mm-yyyy.
x509.version	The certificate version number.
x509.ext.xxx	Each of the attributes that are contained in the x509 certificate extension is included. They are prefixed with the name x509.ext.

Notes:

- The x509 data, except for the x509 extensions, is included in the constructed XML document only if it is required by the rule. This design decreases the size of the constructed XML document, which improves performance.
- All data is XML encoded. Non-printable data is encoded as `\x`*hh*`;`, where *hh* is the code point in hexadecimal form.

If the selected authentication method is Kerberos authentication, an extended attribute that represents the security identifier (SID) is available in the XML representation of the authentication data. The name of the attribute is `attr:<spnego-sid-attr-name>`, which corresponds to the **spnego-sid-attr-name** configuration entry in the **[spnego]** stanza.

External user mapping

Use the HTTP Callout Mapping Module to invoke a web service to perform the STSUU enrichment and return the modified content back into the flow of the federation.

The HTTP Callout Mapping Module can be optionally used by the runtime during federation mapping operations. The module can call out to external services in one of these formats: XML, JSON, and WS-Trust. The following code shows an example of the JSON configuration data:

```
{
 "uri": "http:mywebserver.com",
 "sslKeyStore" : "keystore",
 "autType": "BASIC, NONE, CERTIFICATE",
 "basicauthusername": "username",
 "basicauthpassword": "password",
 "messageFormat":"XML, JSON, WSTRUST",
 "certKeyStore": "keystore",
 "certKeyAlias": "keyalias",
 "issuerUri":"urn:itfim:wstrust:issuer",
 "appliesTo" : "urn:itfim:wstrust:tamToken",
 }
```

If the message format is WSTRUST, the SOAP body of the message is the XML representation of STSUU. If the message format is XML or JSON, then those formats represent the request body. The response has the same message format of the request.

Password strength

The password strength module validates the strength of new passwords.

This module is started by a password change operation, which is initiated by the **/pkmspasswd** command. It evaluates the new password against an XSLT rule to determine whether the new password meets the configured criteria.

Security Access Manager uses the password strength rules that are configured by this module in addition to the password policy rules that are set in **pdadmin** to evaluate each password. The new password must meet both the password policy settings and the XSLT rules that are imposed by this module.

Password strength validation rule language

Extensible Style Language (XSL) specifies rules. Extensible Markup Language (XML) is the language for the data. It forms an input to the rules. The combination of XML and XSL provides a platform-independent method of expressing both the inputs to the rules evaluator and the rules themselves.

XML expresses complex data types in a structured and standard manner in text format. Using this text format, you can write processing rules for the XML data that are independent of operating systems and programming languages.

XSL is a functional style sheet language that can accomplish simple or complex tasks. XSL possesses an inherent ability to analyze and evaluate XML data, which is becoming the standard for data representation. XSL is built on other XML-based standards, such as XPath, which is the expression language at the core of a password strength validation rule.

To implement the password strength validation rules, it is necessary to impose some constraints on the XSL rules. The output of the rule evaluation must conform to one of a known set of result strings. For more information about the format and constraints of password strength validation rules, see “Format and constraints of rules” on page 229.

UMI XML document model

The Universal Management Infrastructure XML document model (UMI XML model) is a set of restrictions that are placed on the XSL or XML model by the password strength validation rules implementation. This model enables the interface to be both simple and functional for password strength validation purposes.

The model constrains the rules to function in a predetermined XML document format with the same top-level XML document element for all rules. The XML UMI that is imported by the rules evaluator from user attributes must be inserted into this XML document before the data can be used. To simplify the process of defining rules, the rules must operate in the confines of the UMI XML model.

The UMI XML model requires the XML document to contain the following top-level XML element into which all target UMI for a particular rule evaluation is inserted. The `XMLUMI` element is created automatically as part of the rule evaluation process by the password strength validation engine.

```
<XMLUMI>
<!--XML formatted UMI are inserted here.    -->
</XMLUMI>
```

As a result of this restriction, the XPath to the data in a rule must include the prefix /XMLUMI to access a particular data element in the model. For example, if you add a UMI item of stsuuser:STSUniversalUser to the document, you must specify the XPath /XMLUMI/stsuuser:STSUniversalUser to access the data in the XML object stsuuser:STSUniversalUser.

An XPath is the path to a particular child element in the hierarchy of a structured XML data object. It is similar to the mechanism that a directory path on a hard disk uses to access a specific file. An XPath designation starts from the root of the document and traces a path from this root through its child elements to the specific element that is referenced. For example, to use the example entitlement stsuuser:STSUniversalUser in the "XML password strength validation model" on page 228, the following XPath accesses the address element of /XMLUMI/stsuuser:STSUniversalUser:

```
"/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/stsuuser:
Attribute[@name='password']/stsuuser:Value"
```

XPaths like this example are the means by which password strength validation rules access the UMI data values for attribute-based password strength validation decisions.

Because all data elements are restricted to work in the UMI XML model, the password strength validation rules must also be restricted to operate on or match XPaths in the model. Therefore, XSL template match statements are also restricted to matching XPaths that start from /XMLUMI in the UMI XML document. For more information, see "Format and constraints of rules" on page 229.

Containers and XML UMI container names

When data is requested from a resource manager, the granularity of the returned XML data is at the level of a single container of information. The container is normally also the smallest data element, for example, elements that might be considered for billing purposes.

This convention also applies to the UMI XML model. The UMI that is used in password strength validation rules is also defined and manipulated as containers of XML data. For example, the stsuuser:STSUniversalUser XML object that is defined in "XML password strength validation model" on page 228 is an example of a UMI container.

The topmost element in the definition of an item of UMI is the *container name* of that item. When you define a password strength validation rule, the XPath to the XML definition of data in any UMI container must be referenced with the name of the container as the first element after /XMLUMI.

To access any element in the stsuuser:STSUniversalUser container in the UMI item stsuuser:STSUniversalUser example, prefix the XPath specification with stsuuser:STSUniversalUser. For example, "/stsuuser:STSUniversalUser/ stsuuser:AttributeList/stsuuser:Attribute[@name='password']/stsuuser:Value" refers to the password value.

To access this information from a password strength validation rule, prefix this XPath with the top-level element of the XML target UMI input document, which is XMLUMI. For example, "/XMLUMI/stsuuser:STSUniversalUser/ stsuuser:AttributeList/stsuuser:Attribute[@name='password']/stsuuser:Value".

The example rule file contains a template match statement of /XMLUMI/ stsuuser:STSUniversalUser/stsuuser:AttributeList, which means that you can specify all attributes of the password strength validation file without a prefix. For example, "/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList/ stsuuser:Attribute[@name='password']/stsuuser:Value" is the same as "stsuuser:Attribute[@name='password']/stsuuser:Value". For more information, see "Format and constraints of rules" on page 229.

XML password strength validation model

The following UMI XML document shows the data that is passed to the XSL processor from the rules evaluator during the evaluation of a password strength validation rule.

The document contains one container that is named stsuuser. The attribute value of the container stsuuser:STSUniversalUser is defined in XML.

The evaluator automatically encompasses all of the data under the XML top-level node declaration XMLUMI when the UMI XML document is created, so this top-level element is added for clarity.

The XML document is automatically created by Security Access Manager with the attributes that are in the password change request. The XML document that is passed to the evaluation routines by the password strength validation rules evaluator is as follows:

```
<?xml version="1.0" encoding='UTF-8'?>

<XMLUMI>
  <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
    <stsuuser:Principal>
      <stsuuser:Attribute name="name">
        <stsuuser:Value>
          - user identity -
        </stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:Principal>
    <stsuuser:AttributeList>
      <stsuuser:Attribute name="old-password">
        <stsuuser:Value>-attrvalue-</stsuuser:Value>
      </stsuuser:Attribute>
      <stsuuser:Attribute name="password">
        <stsuuser:Value>-attrvalue-</stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:AttributeList>
  </stsuuser:STSUniversalUser>
</XMLUMI>
```

When you reference a particular UMI item in the XMLUMI document for a rule, the XPath path specifier can begin from the container name of the XML element, for example, stsuuser:STSUniversalUser. If the callers want to specify their own template match statement explicitly, they can.

For more information, see "Format and constraints of rules" on page 229.

Password strength rules evaluator

The evaluator evaluates password strength rules in the constraints that are required by the password strength validation engine. A configuration file that you specify supplies the pre-configured rules to the password strength validation module.

The password strength validation rules evaluator sends the rule policy with the XML representation of the password change request to the XSL processor for evaluation.

The input into the rule evaluation is an XML representation of the change password data. For example:

```
<?xml version="1.0" encoding='UTF-8'?>

<XMLUMI>
  <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
    <stsuuser:Principal>
      <stsuuser:Attribute name="name">
        <stsuuser:Value>
          testuser
        </stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:Principal>
    <stsuuser:AttributeList>
      <stsuuser:Attribute name="old-password">
        <stsuuser:Value>passw0rd</stsuuser:Value>
      </stsuuser:Attribute>
      <stsuuser:Attribute name="password">
        <stsuuser:Value>newPassw0rd</stsuuser:Value>
      </stsuuser:Attribute>
    </stsuuser:AttributeList>
  </stsuuser:STSUniversalUser>
</XMLUMI>
```

The output of the evaluation is as follows:

```
<valid>result</valid>
```

where *result* is either `true` or `false`.

If the XSLT rules do not generate a result of either `true` or `false`, the password change fails, and an authentication error is printed in the WebSEAL log. For example:

```
HPDAC0949E   Validation of the rule text for rule object "pwdstrengthauthn" failed.
Error code 0x1005b3b4 was returned along with error message "<valid xmlns:stsuuser="
urn:ibm:names:ITFIM:1.0:stsuuser">invalidResponse</valid>".
50        2014-07-29-17:52:01.670+10:00I----- 0x132120DD webseald WARNING ias authsvc
pdauthn.cpp 1497 0x7f9c811a8700 -- HPDIA0221W   Authentication for user userA failed.
You have used an invalid user name, password or client certificate.
```

Format and constraints of rules:

Define a password strength rule as an XSL template in an XSL style sheet.

Write the rule in a valid XSL template rule format with XSLT version 1.0. It must return a text document that contains the string identifiers in "User mapping rules evaluator" on page 217.

Sample password strength rule:

This example XSLT rule ensures that a password is at least 8 characters long.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser" version="1.0">

    <!-- Required to constrain output of rule evaluation -->
    <xsl:output method="xml" omit-xml-declaration="yes" encoding='UTF=8' indent="no"/>

    <!-- Need this to ensure default text node printing is off -->
    <xsl:template match="text()"></xsl:template>

    <!-- Let's make it easier by matching the constant part of our XML name -->
    <xsl:template match="/XMLUMI/stsuuser:STSUniversalUser/stsuuser:AttributeList">A
      <xsl:choose>
        <xsl:when test="string-length(stsuuser:Attribute
        [@name='password']/stsuuser:Value) >= 8">
          <valid>true</valid>
        </xsl:when>
        <xsl:otherwise>
          <valid>false</valid>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>

</xsl:stylesheet>
```

Enabling password strength validation

The password strength validation module is disabled by default. You must enable it before you can validate whether a new password meets the configured criteria.

Procedure

1. Access the local management interface to configure an XSLT rules file to define the password strength rules. The following steps use `password-rules.xslt` as an example.
2. Select **Secure Web Settings** > **Global Settings** > **Password Strength** from the top menu. The Password Strength management page displays.
3. Take one of the following actions:
   - If rules files exist, select the file that you want to enable, such as `password-rules.xslt`, from the available list of **File Names**.
   - If no rules files exist:
     a. Click **New** to create a new rules file.
     b. Enter a name for the new file such as `password-rules.xslt`.
     c. Click **Save**. The system generates a new file that is based on the default template.
4. Click **Edit**.
5. Update the file to reflect the rules you want to set.
6. Click **Save**.
7. Access the WebSEAL configuration file for your instance.
8. Update the **`[password-strength]`** stanza in the WebSEAL configuration file as follows:

   ```
   [password-strength]
   rules-file = file
   debug-level = level
   ```

   where:

   file Specifies the name of the rules file for the password strength validation module.

   level Controls the trace level for the module.

   ```
   [password-strength]
   rules-file = password-rules.xslt
   debug-level = 5
   ```

   Note: The *level* variable indicates the trace level; 1 designates a minimal amount of tracing, and 9 designates the maximum. The Security Access

Manager **pdadmin** trace command also modifies the trace level by using the trace component name of pd.cas.pwdstrength. This trace component is only available after the first change password operation is processed.

Password strength validation attributes

You can use these attributes in the password strength validation rules.

Table 40. Valid password strength validation attributes

Attribute	Description
old-password	The original password for the user.
password	The new password for the user.

Post-authentication processing

This chapter discusses supplemental post-authentication processes.

Topic Index:

Automatic redirection after authentication

This section contains the following topics:

Overview of automatic redirection

When a user makes a request for a resource in a WebSEAL domain, WebSEAL sends the resource to the user upon successful authentication and policy checks. As an alternative to this standard response, you can configure WebSEAL to automatically redirect the user to a specially designated home or welcome page.

The customized redirection can be further configured with a range of macros specifying the user's authorization level, username, host name and so forth.

This forced redirection after login is appropriate, for example, when users enter the WebSEAL domain through a portal page. Automatic redirection also overrides user attempts to directly access specific pages within the domain by selecting user bookmarks.

The automatic redirection process flow is as follows:

1. The user sends a request and successfully authenticates.
2. WebSEAL builds a custom response and returns it to the browser as a redirect.

   This redirect response contains the URL value specified by the **login-redirect-page** stanza entry in the WebSEAL configuration file.
3. The browser follows the redirect response (containing the configured URL).
4. WebSEAL returns the page located at the configured URL.

Automatic redirection after login is enabled and disabled independently for each authentication method. Automatic redirection is supported for the following authentication methods:

- Forms authentication
- Basic authentication
- External authentication interface

Enabling automatic redirection

About this task

To configure automatic redirection, complete the following steps:

Procedure

1. Open the WebSEAL configuration file for editing.
2. Enable automatic redirection for each of the applicable authentication methods by uncommenting the entry for each method in the **[enable-redirects]** stanza:
   ```
   [enable-redirects]
   redirect = forms-auth
   redirect = basic-auth
   redirect = cert-auth
   redirect = ext-auth-interface
   redirect = oidc
   ```

   The example above enables automatic redirection for forms authentication, basic authentication, certificate authentication, and EAI authentication.
3. Specify the URL to which the user is redirected after login. The URL can be expressed as either an absolute URL or a server-relative URL, with or without an embedded macro. For example:
   ```
   [acnt-mgt]
   login-redirect-page = http://www.ibm.com
   ```
   or:
   ```
   [acnt-mgt]
   login-redirect-page = /jct/intro-page.html
   ```
   or:
   ```
   [acnt-mgt]
   login-redirect-page = /jct/intro-page.html?level=%AUTHNLEVEL%&url=%URL%
   ```
4. Stop and restart the WebSEAL server.

Disabling automatic redirection

About this task

To disable automatic redirection, complete the following steps:

Procedure

1. Open the WebSEAL configuration file for editing.
2. Disable automatic redirection for each of the applicable authentication methods by commenting or removing the entry for each authentication method in the **[enable-redirects]** stanza:
   ```
   [enable-redirects]
   #redirect = forms-auth
   #redirect = basic-auth
   #redirect = cert-auth
   #redirect = ext-auth-interface
   #redirect = oidc
   ```
   Note that the hash character (#) is added to the start of each line. The example above disables automatic redirection for forms authentication, basic authentication, certificate authentication, and EAI authentication.
3. Stop and restart the WebSEAL server.

Limitations

WebSEAL does not support automatic redirection at login under the following conditions:

- During reauthentication.
- When the browser is reopened while using basic authentication.

  Redirection works as expected the first time a user visits a page with a browser and authenticates with a valid user name and password. However, if that instance of the browser is closed and another opened, the redirected page is not displayed after the user is authenticated.

Macro support for automatic redirection

Automatic redirection provides support for a subset of the macros provided by WebSEAL to customize the static redirect URL. Macros allow dynamic substitution of information from WebSEAL.

Macros are specified as an argument in the query string of the redirection URL. Specific characters in the macro values are URI-encoded (see “Encoding of macro contents” on page 234).

Valid WebSEAL macros for use in automatic redirection are:

Macro	Description
`AUTHNLEVEL`	Authentication level required by authentication strength policy (step-up).
`HOSTNAME`	Fully qualified host name.
`PROTOCOL`	The client connection protocol used. Can be HTTP or HTTPS.
`URL`	The URL requested by the client.
`USERNAME`	The name of the logged in user. The value "unauthenticated" is used for users who are not logged in. (See also “Customization of login forms for reauthentication” on page 186.)
`HTTPHDR{name}`	Used to include the contents of a specified HTTP header. If the specified HTTP header does not exist within the request, the macro contains the text: `Unknown`. For example, the macro name to include the "Host" HTTP header is `HTTPHDR{Host}`.
`CREDATTR{name}`	Used to include the contents of a specified attribute in the user credential. If the specified credential attribute does not exist in the request, the macro contains the text: 'Unknown'. For example, use the following macro name to include the **tagvalue_session_index** attribute, which contains the secret token for the session: `CREDATTR{tagvalue_session_index}`.

Encoding of macro contents:

Some macro content contains user-provided data such as the requested URI or the Referer header of that request. It is important for security reasons to ensure that reserved, or special characters in client-supplied data are encoded.

WebSEAL URI encodes macro contents to ensure that the content does not return reserved, or special characters back to the client. URI encoding is an international standard that allows you to map the wide range of characters used worldwide into the limited character-set used by a URI.

Notes on encoding macro contents:

- WebSEAL always applies URI encoding to macro contents, even if the original data has already been encoded.
- Encoded macro contents must be decoded using standard URI decoding rules.
- URI encoding increases the string length of macro content, and therefore the Location header (where the content is embedded in the query string). For a discussion of Location header length issues, see "Macro content length considerations."

Macro content length considerations:

Information supplied by macros increases the string length of the Location URI header. URI encoding of macro content further increases this string length.

Some client applications (such as WAP browsers on cellular phones) have URI length limitations due to the small memory capacity of the device. If a URI exceeds the length limitation on such a client device, errors can occur and the link will likely fail.

WebSEAL does not impose any length restrictions on the Location URI header. Therefore, when configuring macros for local response redirection, you must carefully consider the possible limitations of client devices that access your site. You can estimate the length of the Location header by determining the fixed lengths of the URI and the TAM_OP values, and factor in expected sizes of any macros used in the query string.

The following table provides information about the possible lengths of the content provided by the macros used for local response redirection:

Macro	Size of Content
`AUTHNLEVEL`	No more than 10 characters.
`HOSTNAME`	The length of the HOST header of the corresponding request, or the fully qualified host name of the WebSEAL system if the HOST header is not present.
`PROTOCOL`	No more than 10 characters.
`URL`	Length of the request URI.
`USERNAME`	Maximum length defined by user name length policy for this implementation of WebSEAL.
`HTTPHDR{name}`	Length of the specified HTTP header.
`CREDATTR{name}`	Length of the contents for the specified attribute in the user credential.

Server-side request caching

This section contains the following topics:

Server-side request caching concepts

In past versions of WebSEAL, WebSEAL created a cache entry for the URL of a user request whenever authentication was required. Upon successful authentication, WebSEAL sent an HTTP redirect to the browser that included this URL. The browser then followed the redirect to the original resource location.

A limitation of this implementation became apparent when, for example, a POST request was interrupted by a session timeout that required the user to login again. Because WebSEAL only cached the URL of the original request, the POST data (including the Method and Message-body) were lost during the HTTP redirect. The user had to rebuild the POST request.

WebSEAL now caches a more complete set of request data and uses this cached data to rebuild the request during the HTTP redirect whenever the request processing is interrupted and the user is required to login again. This solution particularly benefits POST and PUT requests, because these requests types can include a message body in the request.

Server-side request caching is supported for forms, external authentication interface, e-community single signon, and certificate authentication methods whenever the request processing is interrupted by a login requirement, a reauthentication requirement, or an authentication strength (step-up) requirement.

Process flow for server-side request caching

When an additional authentication requirement interrupts a request, the user is prompted to login again.

After successful authentication, WebSEAL sends a redirect to the browser for the original resource. Upon receiving this request, WebSEAL rebuilds the request using the cached data and processes the request with that data.

Cached request data includes URL, Method, Message-body, query strings, and all other HTTP headers (including cookies). This data is temporarily stored in the WebSEAL session cache.

The following diagram illustrates a typical server-side request caching process flow:

1. The user successfully logs in and submits an HTTP request for a resource involving a CGI-generated data form. WebSEAL creates a session cache entry for the user.
2. The back-end application server returns the form to the user.
3. During the time it takes the user to fill in the form, the configured session timeout for the user expires. WebSEAL removes the user's cache entry (including credentials) and session ID.
4. Not aware of the session timeout, the user eventually submits the completed form (POST). WebSEAL finds no session cache entry for the user and creates a new cache entry.

5. Because WebSEAL finds no credentials for this user, the user must authenticate. WebSEAL temporarily caches the complete information contained in the POST request and sends a login form to the user.
6. The user submits the completed login form to WebSEAL. Authentication is successful. The cache now contains the user's credentials, as well as the data from the originally cached request.
7. WebSEAL returns a redirect response to the browser containing the URL of the originally requested resource.
8. The browser follows the redirect. WebSEAL intercepts the redirect and rebuilds the original request (the CGI-generated data form) using the cached POST data. The restored form is delivered to the URL destination.

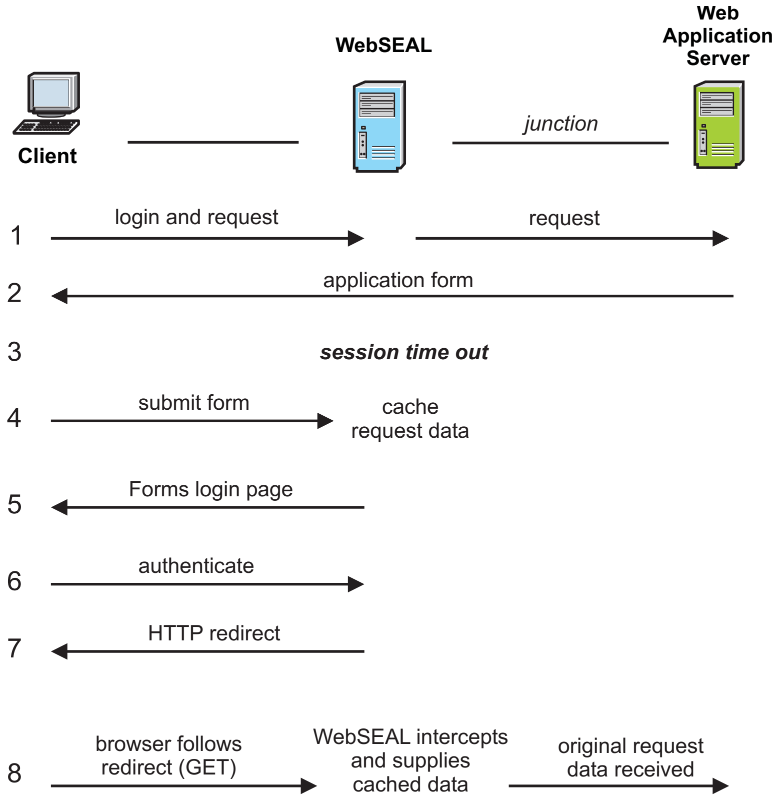


Figure 17. Example WebSEAL request caching process flow

Configuration of server-side caching

You can modify settings in the **[server]** stanza of the WebSEAL configuration file to specify limits to the size of the requests that WebSEAL reads and caches.

Usage notes for server-side request caching:

- Server-side caching helps protect WebSEAL from denial of service attack types that could cause WebSEAL to cache more data than it can handle.
- Server-side request caching does not function correctly if the user session time out value expires during the login process. In this situation, the cache entry is lost.

- Server-side request caching can cause limitations with the browser's ability to manipulate the resource. The browser is unaware that WebSEAL has rebuilt the HTTP redirect. Therefore the browser's reload/refresh function and caching ability can be hindered.

The following sections describe the settings that you can modify:

Modification of request-body-max-read:

The **request-body-max-read** stanza entry specifies the maximum number of bytes of content to read from the body of POST requests.

This configuration is used for dynurl, authentication, and request caching. The **request-body-max-read** stanza entry affects the request body only. It does not impose limits on other components of a request, such as request line and headers. See the **max-client-read** configuration entry in the Web Reverse Proxy Stanza Reference topics in the Knowledge Center.

The value of the **request-body-max-read** stanza entry affects the amount of data that WebSEAL caches for users who must authenticate before their request can be fulfilled. For example, a user name and password submitted with a login form must fit into the **request-body-max-read** limit. This stanza entry affects all requests that have body content, such as POST and PUT requests.

This stanza entry impacts forms authentication, because it limits the size of the POST data that is processed when performing such authentication. To maintain a request body size sufficient for forms authentication, WebSEAL sets an absolute minimum of 512 bytes on **request-body-max-read**. If you specify a value below that minimum, the setting is ignored and the value 512 is used. There is no maximum value limit.

This stanza entry also impacts dynamic URL processing because the query portion of a POST request URI is contained in the request body.

Note: This setting does not limit the maximum POST size. The maximum POST size is unlimited.

The default value is 4096 bytes:

```
[server]
request-body-max-read = 4096
```

When the server-side cache setting for **request-body-max-read** is exceeded during a request, WebSEAL ends the request caching process. WebSEAL returns a Request Caching Failed error message to the browser, and writes the error to the log file. You can customize this error message. See “Guidelines for customizing response pages” on page 65.

The value of **request-body-max-read** also affects the value specified for **request-max-cache**. See “Modification of request-max-cache.”

Modification of request-max-cache:

When a user is prompted to authenticate before a request can be fulfilled, the data from that request is cached for processing after the completion of the authentication. The maximum amount of data cached per request is specified by the **request-max-cache** stanza entry.

To ensure that you cache the full value of **request-body-max-read**, you must account for the maximum size of all the other request components in this value. For example, if you want to cache 2048 bytes of request body content, and you anticipate that the maximum size of all request headers and cookies is 4096 bytes:

1. Set **request-body-max-read** = 2048
2. Set **request-max-cache** = 2048 + 4096 = 6144

The default value for **request-max-cache** is 8192.

```
[server]
request-max-cache = 8192
```

When the server-side cache setting for **request-max-cache** is exceeded during a request, WebSEAL ends the request caching process. WebSEAL returns a Request Caching Failed error message to the browser, and writes the error to the log file. You can customize this error message. See “Guidelines for customizing response pages” on page 65.

There is no maximum size for this value other than the maximum imposed by the data type. However, increasing the size can possibly adversely affect performance and system security. Allocating larger buffers increases memory usage and therefore could possibly decrease performance. More importantly, allocating very large buffers increases the risk of a successful denial-of-service attack by a malicious user. The risk is increased simply because WebSEAL is loading and holding more data into memory, which provides the user with a larger buffer from which to attempt an attack.

Password processing

This chapter discusses password processing options available during WebSEAL authentication.

Topic Index:

Login failure policy ("three strikes" login policy)

This section contains the following topics:

Login failure policy concepts

The login failure ("three strikes") policy, available for Security Access Manager installations using an LDAP-based user registry, enables you to specify a maximum number of failed login attempts (n) and a penalty lockout time (x), such that after "n" failed login attempts a user is locked out for "x" seconds (or, alternatively, the account is disabled).

The login failure policy can help prevent computer password attacks. The policy creates a condition where a user must wait a period of time before making additional login attempts. For example, a policy could dictate 3 failed attempts followed by a 180 second lockout penalty. This type of login policy can prevent random computer-generated login attempts that occur many times a second.

The login failure policy requires the joint contribution of two policy settings:

- Maximum number of failed login attempts:

  max-login-failures

- Penalty for reaching or exceeding the failed login attempt setting:

  disable-time-interval

  The penalty setting can include a temporary account lockout time interval or a complete disabling of the account.

WebSEAL returns a server response error page (`acct_locked.html`) that notifies the user of the penalty. The **late-lockout-notification** stanza entry in the **[server]** stanza of the WebSEAL configuration file specifies whether this notification occurs when the user reaches the **max-login-failures** limit, or at the next login attempt after reaching the limit.

See also “Removal of a user session at login failure policy limit” on page 185.

Setting the login failure policy

About this task

Login failure policy controls the maximum number of failed login attempts allowed before an account lockout penalty is imposed.

Procedure

- Use the **pdadmin policy** command to set the login failure policy. Use the following syntax to set the login failure policy:

  ```
  policy set max-login-failures {number|unset} [-user username]
  ```

- Use the following syntax to display the current login failure policy setting:

  ```
  policy get max-login-failures [-user username]
  ```

  The *number* argument specifies the number of failed login attempts allowed before the penalty is applied. By default, the policy is enabled with a setting of 10 login attempts. For example:

  ```
  pdadmin> policy get max-login-failures
  Maximum login failures: 10
  ```

  The `unset` argument disables the policy. With this setting, the policy contains no value and the policy is not checked or enforced.

- You can apply **max-login-failures** policy to a specific user or apply the policy globally to all users listed in the user registry.

Example

Example global setting:

```
pdadmin> policy set max-login-failures 3
```

Example user-specific setting:

```
pdadmin> policy set max-login-failures 5 -user laura
```

The account lockout penalty value is specified by the **disable-time-interval** policy. See “Setting the account disable time interval.”

Setting the account disable time interval

About this task

Login failure policy controls the maximum number of failed login attempts allowed before an account lockout penalty is imposed.

Procedure

- Use the **pdadmin policy** command to set the penalty time interval for the login failure policy. Use the following syntax to set the penalty time interval:

  ```
  policy set disable-time-interval {number|unset|disable} [-user username]
  ```

- Use the following syntax to display the current penalty time interval setting:

  ```
  policy get disable-time-interval [-user username]
  ```

  The *number* argument specifies the number of seconds that an account is locked out if the maximum number of failed login attempts is reached or exceeded. By default, the lockout time interval is 180 seconds. For example:

  ```
  pdadmin> policy get disable-time-interval
  Disable time interval: 180
  ```

  The `unset` argument disables the policy. With this setting, the policy contains no value and the policy is not checked or enforced.

  The `disable` argument permanently locks the user out of the account after reaching or exceeding the login attempt limit and the LDAP **account valid** attribute for this user is set to "no". An administrator can re-enable the account by using the Web Portal Manager or **pdadmin** utility.

  Note: Setting the **disable-time-interval** to "disable" results in additional administration overhead, because the account must be manually re-enabled by the administrator. After the account is re-enabled, the updated **account valid** LDAP attribute information might not be immediately available. This situation can occur when using WebSEAL with an LDAP environment that includes replicated LDAP servers. In this case, the updated information is propagated to the LDAP replicas according to the LDAP configuration settings that specify the time interval for performing updates.

  You can apply **disable-time-interval** policy to a specific user or apply the policy globally to all users listed in the user registry.

Example

Example global setting:

```
pdadmin> policy set disable-time-interval 60
```

Example user-specific setting:

```
pdadmin> policy set disable-time-interval disable -user laura
```

The **late-lockout-notification** stanza entry in the **[server]** stanza of the WebSEAL configuration file specifies whether this account lockout notification occurs when the user reaches the **max-login-failures** limit, or at the next login attempt after reaching the limit. See "Configuring the account disable notification response."

Configuring the account disable notification response

About this task

WebSEAL returns a server response error page (`acct_locked.html`) that notifies the user of the penalty for reaching or exceeding the **max-login-failures** limit.

The **late-lockout-notification** stanza entry in the **[server]** stanza of the WebSEAL configuration file specifies whether this error page is returned when the user reaches the **max-login-failures** limit, or at the next login attempt after reaching the limit.

The action of account lockout or account disable does not remove the session cache entry of the user, but it does prevent future logins by that user until the account is unlocked.

Procedure

- The default **late-lockout-notification** setting for new installations of WebSEAL is "no". Upon reaching the maximum value set by the **max-login-failures** policy, WebSEAL immediately sends the account disabled error page to the user. For example:

```
[server]
late-lockout-notification = no
```

- The default setting for migrated installations of WebSEAL is "yes". Upon reaching the maximum value set by the **max-login-failures** policy, WebSEAL returns another login prompt to the user. WebSEAL does not send the account disabled error page to the user until the next login attempt. This setting represents the pre-version 6.0 behavior for the **max-login-failures** policy. For example:

```
[server]
late-lockout-notification = yes
```

- If the **disable-time-interval** policy is set to a number of seconds, the error message indicates that the account is temporarily locked out.
- If the **disable-time-interval** policy is set to "disable", the error message indicates that the account has been disabled and that an administrator is required to reset (unlock) the account.

Login failure policy with replicated WebSEAL servers

You use the login failure policy to ensure that an account is locked after a specified number of failed login attempts. This policy performs as expected in a configuration involving one WebSEAL server. In a configuration involving multiple front-end WebSEAL servers with a load-balancing mechanism, the results of the policy are affected by the fact that each WebSEAL server maintains its own local count of failed login attempts by default.

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 available replicated WebSEAL server. Now the client has three more opportunities to attempt a successful login.

For "n" attempts configured on each WebSEAL server, and "m" front-end replicated WebSEAL servers, you are guaranteed an initial account lock on one server after "n" attempts. You are also guaranteed "n" x "m" total attempts to log in across all configured servers. However, after "n" attempts, it is not clear whether subsequent authentication failures are due to the lock on a particular server, or due to continuing incorrect login attempts across the remaining replicated servers.

The "n" x "m" calculation provides a fixed maximum upper limit on the total number of consecutive login attempts before a complete lockout occurs. A case can be made that this number is still probably far less than the number of attempts statistically required to "break" a password.

If your business security solution requires a login failure policy, you should understand the implications of a load-balanced, multiple front-end WebSEAL configuration on this policy.

Decreasing the number of possible login attempts:

Use the **login-failures-persistent** stanza entry to decrease the upper limit on the total number of consecutive login attempts.

About this task

The **login-failures-persistent** entry is located in the **[ldap]** stanza of the WebSEAL configuration file. This entry controls whether login failures are tracked in the local cache of the WebSEAL system or in the LDAP registry. If the failures are tracked in the registry, all replicas that share the registry share the count. This configuration reduces the maximum number of attempts to “n” instead of “n” x “m”. There is a minor performance impact as a result of enabling **login-failures-persistent** because a write operation to the LDAP server has to occur for each login attempt.

Password strength policy

Password strength policy refers to the stipulations placed on the construction of a password by password policy rules.

This section contains the following topics:

Password strength policy concepts

You can control the password strength policy.

Security Access Manager provides five **pdadmin policy** commands to control the password strength policy. See the Command Reference topics in the Knowledge Center for information on these **pdadmin policy** commands.

Password strength policies

The five password strength policies implemented through the **pdadmin policy** command include:

- Minimum password length (**min-password-length**)
- Minimum alphabetic characters (**min-password-alphas**)
- Minimum non-alphabetic characters (**min-password-non-alphas**)
- Maximum repeated characters (**max-password-repeated-chars**)
- Spaces allowed (**password-spaces**)

These policies are enforced when you create a user with the Web Portal Manager or **pdadmin** utility, and when a password is changed with the Web Portal Manager or **pdadmin** utility, or the **pkmspasswd** utility.

Syntax for password strength policy commands

The **pdadmin policy** commands, used to set password strength policy, are appropriate for use only with an LDAP type of user registry.

The **unset** option disables this policy attribute; that is, the policy is not enforced.

```
policy set min-password-length {number|unset} [-user username]
policy get min-password-length [-user username]
```

Manages the policy that controls the minimum length of a password.

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.

The default setting is 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.

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.

The default setting is 4.

policy set min-password-non-alphas {*number*|unset} [-user *username*]
policy get min-password-non-alphas [-user *username*]

Manages the policy controlling the minimum number of non-alphabetic (numeric) characters allowed in a password.

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.

The default setting is 1.

policy set max-password-repeated-chars {*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.

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.

The default setting is 2.

policy set password-spaces {yes|no|unset} [-user *username*]
policy get password-spaces [-user *username*]

Manages the policy controlling whether a password can contain spaces.

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry.

The default setting is unset.

Default password strength policy values

The following table lists the password strength policies and the default values:

Policy	Default Value
min-password-length	8
min-password-alphas	4
min-password-non-alphas	1
max-password-repeated-chars	2
password-spaces	not set

To create the password policy behavior found in earlier releases of Security Access Manager, apply the **unset** option to each of the five password policies listed above.

Valid and not valid password examples

The following table illustrates several password examples and the results based on the default values for the five password strength policies:

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.

Specifying user and global settings

About this task

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 a policy (with the **unset** argument). The policy contains no value and the policy is not checked or enforced.

Example

A global minimum password length policy of 8 characters is created. As an exception to this policy, user **matt** is given a minimum password length policy of 4 characters.

```
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
```

***

The specific minimum password length policy for user **matt** is unset. User **matt** is now governed by the global minimum password length policy of 8 characters.

```
pdadmin> policy set min-password-length unset -user matt
pdadmin> policy get min-password-length -user matt
Minimum password length: 8
```

***

The global minimum password length policy is unset. All users, including user **matt**, now have no minimum password length policy.

```
pdadmin> policy set min-password-length unset
pdadmin> policy get min-password-length
Minimum password length: unset
```

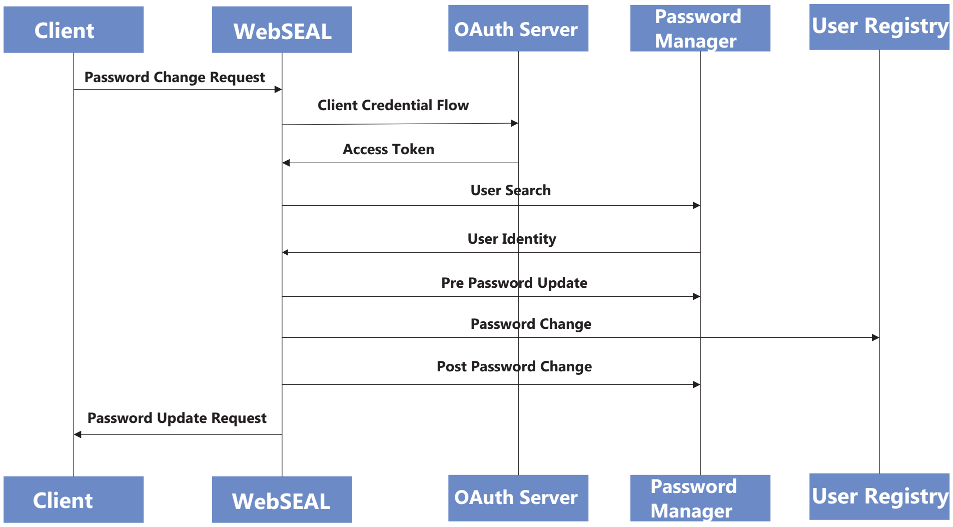
Password Callouts

Password Callouts refers to the ability to invoke external REST services during password updates to handle things such as password strength checking and reverse password synchronisation.

A REST callout can be configured before the password is updated by ISAM and/or after the password has been updated by ISAM.

Password Update Flow

The password update flow is depicted in the following diagram.



Note:

- The **Client Credential Flow** is optional and provides a mechanism by which to obtain an OAuth identity token which can be used in the subsequent callouts. If the **Authentication Request** callout is not configured authentication take places using Basic Authentication
- The **User Search** is optional and provides the ability to map the IBM Security Access Manager user name into an identity which is understood by the password manager. If the **User Search** callout is not configured the IBM Security Access Manager user name is used as the user identity in the subsequent callouts.
- The pre and post password update calls are both optional.

Authentication to the REST services

There are two different methods which can be used to handle authentication to the REST services.

Client Credential Flow

If an authentication endpoint is configured, a callout is made to an OAuth 2.0 service by using the Client Credential flow (see OAuth 2.0 RFE 6749, section 4.4). The authenticated identity information which is used is obtained from the configured `client-id` and `client-secret`. The identity token which is returned from this service is then passed as an authorization header in subsequent calls.

For example:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Contro: no-store
Pragma: no-cache

{
      "access_token":"2YotnFZFEjr1zCsicMWpAA",
      "token_type":"example",
      "expires_in":3600,
      "example_parameter":"example_value"

}
```

Note: IBM Security Access Manager only uses the "access_token" which is returned in the response. It ignores any other fields which are contained in the response.

If an error occurs during authentication a corresponding error page is returned to the client.

Basic Authentication

If an authentication endpoint has not been configured a Basic Authentication token is generated from the configured client-id and client-secret and then passed as an authorization header in subsequent calls.

User Search Operation

The User Search callout provides the ability to map the IBM Security Access Manager user name which is contained in the password update request into a user identity which is known to the configured pre and post password update REST services.

The User Search callout conforms to the querying resources using POST endpoint of the 'System for Cross-domain Identity Management: Protocol' RFC (RFC 7644) : section 3.4.3 (https://tools.ietf.org/html/rfc7644#section-3.4.3).

The filter which is used in the search operation is configurable and conforms to section 3.4.2.2 (https://tools.ietf.org/html/rfc7644#section-3.4.2) of the RFC.

An example request can be:

```
POST /Users/.search
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:SearchRequest"],
   "attributes": ["id"],
   "filter": "personCode eq \"smith\"",
   "count": 1
}
```

An example response can be:

```
HTTP/1.1 200 OK
Content-Type: application/scim+json

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
   "totalResults":1,
   "itemsPerPage":1,
   "startIndex":1,
   "Resources":[
     { "id":"2819c223-7f76-413861904646 }
   ]
}
```

An error page is returned to the client in the event that one of the following scenarios occur:

- an error occurs during the search;
- more than one user is located by the search;
- no user is found by the search

Pre-Password Update Callout

The pre-password update callout is invoked just prior to updating the password within IBM Security Access Manager.

This callout conforms to the `PasswordValidateRequest` section within the draft-hunt-scim-password-mgmt-00 RFC: section 2.5 (https://tools.ietf.org/html/draft-hunt-scim-password-mgmt-00#section-2.5). The user identity is contained within the '$ref' field and is generated by appending the supplied user identity with the configured identity prefix.

An example request can be:

```
POST /PasswordValidateRequestsHTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Accept-Language: en-US
Content-Length: ...

{
   "schemas":
["urn:ietf:params:scim:schemas:core:2.0:password:PasswordValidateRequest"],
   "$ref": "/Users/2819c223-7f76-453a-919d-413861904646",
   "password":"badpwd!"
}
```

If the call is successful an empty '200' response is returned:

```
HTTP/1.1 200 OK
```

If the call is unsuccessful a SCIM error response is returned. The 'detail' field contained in the response is then sent back to the client. The 'Accept-Language' header from the request should be used by the REST service to determine the locale which is to be used when generating the response.

An example response can be:

```
HTTP/1.1 400 Bad Request
Content-Type: application/scim+json

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
   "scimType":"tooLong"
   "detail": "The password must be at least 8 characters in length.",
   "status":"400"
}
```

Post-Password Update Callout

The post-password update callout will be invoked immediately after a successful password update in IBM Security Access Manager.

This callout conforms to the 'Modifying with Patch' section of the 'System for Cross-domain Identity Management: Protocol' RFC (RFC 7644) : section 3.5.2 (https://tools.ietf.org/html/rfc7644#section-3.5.2). The supplied user identity is appended to the configured post-password update endpoint.

An example of the request can be:

```
PATCH /Users/2819c223-7f76-453a-919d-413861904646
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
   "Operations": [
      {
         "op":"replace",
         "value": { "password": "newPassw0rd" }
      }
   ]
}
```

If the call is successful an empty '204' response is returned:

```
HTTP/1.1 204 No Content
Location: https://www.example.com/Users/2819c223-7f76-453a-919d-413861904646
```

If the call is unsuccessful a SCIM error response is returned. The 'detail' field contained in the response is then sent back to the client. The IBM Security Access Manager password update operation is not reverted. The 'Accept-Language' header from the request should be used by the REST service to determine the locale which is to be used by the REST service when generating the response.

An example error response can be:

```
HTTP/1.1 400 Bad Request
Content-Type: application/scim+json

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
   "scimType":"tooLong"
   "detail": "The password must be at least 8 characters in length.",
   "status":"400"
}
```

Credential processing

This chapter discusses features that affect processing of the WebSEAL user credential.

Topic Index:

Extended attributes for credentials

This section contains the following topics:

Mechanisms for adding registry attributes to a credential

You can configure an external service to add attributes to a user credential.

The WebSEAL 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 such as the user name and the list of groups to which the user belongs.

WebSEAL supports several different mechanisms (services) that allow administrators and application developers to extend the authentication process. When WebSEAL conducts the authentication process, it checks to see if any external services have been implemented and configured. When they have, WebSEAL calls those services. The services can do their own processing to build a list of extended attributes about the user identity. These extended attributes are added to the user credential.

The following service is supported:

Registry attribute entitlement service

> This entitlement service is built-in to Security Access Manager by default. This service is an implementation of a class of Security Access Manager entitlement services known as *credential attribute entitlement services*. The *registry attribute entitlement 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 registry attribute entitlement service is a generic entitlement service that can be used by many resource managers. This service takes the place of a previous method that required administrators to add "tag/value" entries to the **`[ldap-ext-creds-tag]`** stanza in the `pd.conf` configuration file. For configuration information, see "Configure a registry attribute entitlement service."
>
> **Note:** Note that Security Access Manager provides additional built-in entitlement services that can be used to add additional information. These additional services, however, obtain the additional information from sources other than user registry entries. For example, the *extended attribute entitlement service* obtains information from ACLs and POPs in the protected resource object space. For more information about entitlement services, see the IBM Knowledge Center.

Configure a registry attribute entitlement service

Complete the instructions in the following sections:

Determine the attributes to add to the credential:

You must determine which attributes you want added to the user credential.

About this task

You must define each user attribute that you want to add to the user credential in a Security Access Manager configuration file. Typically, this configuration is done in the WebSEAL configuration file.

Procedure

- 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 also need the user DN and group DN.

Specify the attributes to add to the credential:

The attributes to add to the credential are configured in several stanzas.

About this task

Add this information to the WebSEAL configuration file.

Review the following example entry.

```
[TAM_CRED_ATTRS_SVC]
eperson = azn_cred_registry_id
group = cn=enterprise, o=tivoli

[TAM_CRED_ATTRS_SVC:eperson]
tagvalue_credattrs_lastname = sn
tagvalue_credattrs_employeetype = employeetype
tagvalue_credattrs_address = homepostaladdress
tagvalue_credattrs_email = mail

[TAM_CRED_ATTRS_SVC:group]
tagvalue_credattrs_businesscategory = businesscategory
```

The stanza name **[TAM_CRED_ATTRS_SVC]** is the Service ID. Inside this stanza are sources of attributes to be retrieved. The source *names*, such as eperson 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.

Procedure

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 might be

```
cn=joeuser, o=tivoli
```

For this user, the LDAP user registry entries might be:

```
sn=Smith
employeetype=bankteller
homepostaladdress="3004 Mission St Santa Cruz CA 95060"
mail=joeuser@bigco.com
```

For the group cn=enterprise,o=tivoli, the LDAP group registry entry might be:

```
businesscategory=finance
```

Using these example configuration entries, the attribute list returned has 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@example.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
```

Junction handling of extended credential attributes

The user-defined credential information created in the previous section can be placed in an HTTP header of the request that is sent across a junction to a back-end server.

You must configure the junction to extract extended attribute data from the credential and insert the data into the HTTP header of the request. This functionality is achieved by setting a junction extended attribute, called **HTTP-Tag-Value**, on the junction object in the WebSEAL protected object space.

You use the **pdadmin object modify set attribute** command to set extended attributes on a junction object in the WebSEAL protected object space.

```
pdadmin> object modify object_name set attribute attr_name attr_value
```

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

An extended attribute (*attr_name*) enables the junction to perform a specific type of functionality. The **HTTP-Tag-Value** extended attribute instructs the junction to extract a particular value from a user's credential and send the value to the back-end server in an HTTP header. The value of the **HTTP-Tag-Value** extended attribute uses the following format:

```
credential_extended_attribute_name = http_header_name
```

The *credential_extended_attribute_name* entry is the same as the attribute specified in the WebSEAL configuration file but without the "tagvalue_" prefix. The entry is not case-sensitive. The *http_header_name* entry specifies the name of the HTTP header used to deliver the data across the junction.

For example (entered as one line):

```
pdadmin> object modify /WebSEAL/WS1/junctionA set attribute
HTTP-Tag-Value credattrs_lastname=surname
```

When WebSEAL processes a user request to a back-end application server, it looks for any **HTTP-Tag-Value** attributes configured on the junction object.

In this example, the configured junction looks at the credential of the user making the request, extracts the value of the *tagvalue_credattrs_lastname* credential extended attribute, and places it in an HTTP header as:

```
surname:Smith
```

In summary:

Table 41. HTTP header example description

Description	Header text
Value of **HTTP-Tag-Value** attribute set on the junction object:	`credattrs_lastname=surname`
Attribute name and value as they appear in the user credential (since `tagvalue_credattrs_lastname=sn`):	`tagvalue_credattrs_lastname:Smith`
HTTP header name and value:	`surname:Smith`

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_surname=Smith
```

Multiple user attribute data can be passed to the junctioned server in HTTP headers by using multiple **pdadmin object modify set attribute** commands to specify multiple **HTTP-Tag-Value** junction attributes (one attribute is specified per command).

HTTP-Tag-Value extended attributes must be attached directly to the junction:

When an evaluation is performed to determine what credential attributes should be passed for a particular object below a junction point, the evaluation is performed not for the child object, but for the junction object. For example, if you create **HTTP-Tag-Value** extended attributes for a junction object named

```
/WebSEAL/myinstance/jct1
```

and you also create **HTTP-Tag-Value** extended attributes for a child object of the junction named

```
/WebSEAL/myinstance/jct1/child
```

then when a client accesses /WebSEAL/myinstance/jct1/child only the attributes attached to /WebSEAL/myinstance/jct1 will be used; the child attributes will be ignored. Therefore, the **HTTP-Tag-Value** extended attributes must be attached directly to the junction.

It is important to understand that this is due not to inheritance but to the fact that WebSEAL determines the junction object associated with a child and finds the **HTTP-Tag-Value** attributes for the junction object itself. In fact, while inheritance of extended attributes was introduced in version 6.0, the processing of the **HTTP-Tag-Value** attributes does not use this inheritance. So, for example, if you create a junction at

```
/WebSEAL/myinstance/jct1
```

with no **HTTP-Tag-Value** attributes attached to it, and instead attach the attributes to the parent at

```
/WebSEAL/myinstance/
```

then those attributes will not be used for jct1 or any of its children. You must attach the **HTTP-Tag-Value** extended attributes directly to /WebSEAL/myinstance/jct1.

tagvalue_always extended attribute:

If an authentication mechanism supports the addition of attributes to the credential, you can use the **tagvalue_always** attribute to define which attributes are always inserted into the HTTP stream. Examples of such authentication mechanisms include EAI and OAuth authentication.

Add the extended attribute **tagvalue_always** to the credential to define the attributes that are always included in the HTTP stream. The format of this attribute is:

```
<attribute name>[:<header name>],...
```

Note: The attribute name and header name cannot contain a comma (,) or colon (:) character. These characters are reserved by the system.

The attribute name field corresponds to the name of the attribute that is to be inserted into the HTTP stream. The optional header name field contains the name of the HTTP header that holds the attribute value. The attribute name is used as the name of the HTTP header if no header name field is specified. Multiple attributes can be specified, with each attribute delineated by a comma. Attributes with multiple values are inserted into a single header with each attribute value separated by a comma.

Credential refresh

This section contains the following topics:

Credential refresh concepts

This section contains the following topics:

Credential refresh overview:

You can configure the credential refresh feature in WebSEAL.

When a user authenticates to WebSEAL, the authentication process accesses the Security Access Manager user registry and builds a credential for the user. The credential contains information about the user that is needed by Security Access Manager to decide whether to grant the user access to the requested resource. An example of credential information is a list of groups to which the user belongs.

During a user session, changes in user information can take place. For example, the user might be added to a new group. When this occurs, there might be a need to update or *refresh* the contents of the user credential, to reflect the new user information. WebSEAL provides a mechanism to enable a credential refresh without requiring the user to log out and then authenticate again.

You can control how the credential refresh feature occurs. WebSEAL provides configuration settings that enable you to specify credential attributes to refresh (update) and credential attributes to preserve (retain). This ability enables you to have precise control over how user credentials are manipulated during a user session.

Use of the credential refresh configuration settings can be important when the authentication process on your WebSEAL server includes call outs to mechanisms that provide additional or extended information about a user. These mechanisms include:

- Credential attribute entitlement service.

  This service is built into Security Access Manager by default.

For more information on the credential attribute services listed above, see "Mechanisms for adding registry attributes to a credential" on page 249.

When credential refresh occurs, the default credential attribute entitlement services is run.

The credential refresh configuration settings enable you to preserve attributes obtained during the initial use of an entitlement service. For example, if an attribute contained a timestamp for the start of the user session, you might want to preserve the timestamp even though the credential was refreshed.

The credential refresh configuration settings also enable you to preserve attributes obtained from a credential extended attribute authentication module. Because custom authentication modules are not run again during the rebuilding of the credential, you use the configuration file settings to specify attributes to be added to the new credential.

Credential refresh rules:

Credential refresh involves the generation of a *new credential* for user identity, followed by an evaluation of the contents of the new credential against the contents of the *old credential* that was obtained during initial user authentication. The contents of the two credentials are combined into a *merged credential* according to the following rules:

1. When an attribute occurs in the new credential but not the old credential, it is added to the merged credential.
2. The following attributes are added to the merged credential based only on their value in the old credential. These attributes are used by the authorization API. They are not changed by values in the new credential.

```
AZN_CRED_AUTHNMECH_INFO
AZN_CRED_BROWSER_INFO
AZN_CRED_IP_ADDRESS
AZN_CRED_PRINCIPAL_NAME
AZN_CRED_AUTH_METHOD
AZN_CRED_USER_INFO
AZN_CRED_QOP_INFO
```

3. For each attribute in the old credential for which there is a corresponding attribute in the new credential, the following rules apply:
   - When there is an entry in the configuration file that matches it, the attribute in the merged credential is preserved or refreshed according to the value of the entry in the configuration file.
   - When there is *not* an entry in the configuration file that matches it, the attribute in the merged credential is assigned the value from the new credential.
4. For each attribute in the old credential for which there is *not* a corresponding attribute in the new credential, the following rules apply:
   - When there is a configuration file entry for the attribute specifying `refresh`, the attribute is *not* added to the merged credential.
   - When there is a configuration file entry for the attribute specifying `preserve`, the attribute is added to the merged credential.
   - When the configuration file does not contain an entry for the attribute, the attribute is not added to the merged credential.

Refresh of cached credential information:

Some user registries maintain cached information. Cached data is kept for a specific amount of time, and is then discarded. After the cached data has expired, it is not reloaded into the cache until the next time the user registry is accessed. Therefore, when changes are made to user registry data, the data is not immediately cached in memory. Likewise, when using a replicated LDAP user registry, the updates to the replicated registries do not occur immediately.

The default lifetime of data in the WebSEAL user cache is 30 seconds. This lifetime begins when the data first enters the cache, such as when a user first authenticates, or when the cached data has expired and WebSEAL contacts the registry to update the data. WebSEAL contacts the registry to update the data during a credential refresh event. The cached information is valid for 30 seconds after it is first obtained from the registry. After 30 seconds, any credential refresh operations *go directly to the user registry*. The access to the user registry also causes the user data to be reloaded into the cache.

The following example shows the algorithm for updating the user cache:

1. The user authenticates at time *auth_time*.
2. The user is added to a group at time *auth_time + 120 seconds*
3. The user's credential is refreshed at time *auth_time + 130 seconds*

   Because the user cache data expired at time *auth_time + 30 seconds*, the new group membership will be added to the user's credential.
4. User is then added to another group at time *auth_time + 135 seconds*
5. User's credential is refreshed at time *auth_time + 140 seconds*

When the user credential is refreshed at *auth_time + 140 seconds*, it does not pick up the new group membership. This is because the user credential is built off cached user data when the cached user data is considered valid (has not expired). Because the user cache data was updated at time *auth_time + 130 seconds*, it is not scheduled to be updated until *auth_time + 160 seconds*. Therefore, the administrator must wait until time *auth_time + 160 seconds* to run the refresh command. At that time, the user credential will pick up the new group memberships.

Configuration file syntax and usage:

The credential refresh behavior is controlled by entries in the **[credential-refresh-attributes]** stanza in the WebSEAL configuration file. The format is:

```
attribute_name_pattern = {preserve|refresh}
```

The attribute name pattern is used to select a given set of attributes. Wildcard matching is supported.

A particular attribute can possibly be matched by many different wildcard patterns. Therefore, the order of elements in the configuration file is important. The firs pattern that matches a given attribute is the only pattern that applies to that attribute.

Attribute names in *attribute_name_pattern* should not be case-sensitive because attribute names in credentials are not case-sensitive.

Example – Preserve all of the tag value attributes added by an extended attribute external authentication C API module:

```
[credential-refresh-attributes]
tagvalue_* = preserve
```

Example – Update the `tagvalue_last_refresh_time` attribute with the value from the new credential, but preserve all other attributes that begin with `tagvalue_`:

```
[credential-refresh-attributes]
tagvalue_last_refresh_time = refresh
tagvalue_* = preserve
```

Note that the ordering of attributes in the file is important. In the following example, tagvalue_last_refresh_time will not be refreshed because it is first matched by the tagvalue_* entry, which is set to preserve:

```
[credential-refresh-attributes]
tagvalue_* = preserve
tagvalue_last_refresh_time = refresh
```

Avoid preserving attributes that begin with the letters `AZN_`. Such attributes are typically used internally by the authorization API during authorization decisions. See information about obtaining attribute lists from credentials in the IBM Knowledge Center.

Default settings for preserve and refresh:

The default settings in the WebSEAL configuration file are:

```
[credential-refresh-attributes]
authentication_level = preserve
tagvalue_* = preserve
```

These settings result in the following behavior:

- The user authentication level is preserved when credentials are refreshed. During a user session, the user authentication level can change when authentication strength policy (step-authentication) is applied. In most cases, you want to preserve the modified authentication level during a credential refresh.

If you do not want to preserve the authentication level, change the configuration file entry:

```
authentication_level = refresh
```

- The tagvalue_* entry preserves all credential attributes whose name begins with the characters `tagvalue_`.

  Attributes with the prefix `tagvalue_` are typically supplied by external authentication C API services that want to add user information to the credential. The prefix is needed to ensure that the credentials are included when WebSEAL inserts credential data into an HTTP header to send across a junction.

Limitations:

- It is not possible to avoid calling the credentials attribute entitlement service during credential refresh. When you have an attribute that should be set only once (during initial authentication) use an extended attribute external authentication C API module to set the attribute.

Configure credential refresh

To configure credential refresh, complete the following steps:

1. Specifying attributes to preserve or refresh:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file.
   - Add entries for attributes to preserve. For example:

     ```
     [credential-refresh-attributes]
     my_cred_attribute1 = preserve
     my_cred_attribute2 = preserve
     ```

   - Add entries to refresh:

     ```
     [credential-refresh-attributes]
     my_cred_attribute3 = refresh
     my_cred_attribute4 = refresh
     ```

   - When appropriate, use the order of the entries to handle both specific entries and groups of entries. For example, to preserve the attribute `special_cred_attr1`, but refresh all other attributes with the naming construct of `special_cred_attr*`, add the following entries:

     ```
     [credential-refresh-attributes]
     special_cred_attr1 = preserve
     special_cred_attr* = refresh
     ```

2. Enabling user session IDs:

About this task

Ensure that user session IDs are enabled for the WebSEAL instance. The credential refresh administration command does not work when user session IDs are not enabled.

```
[session]
user-session-ids = yes
```

3. Enabling placement of server name into junction header:

You can configure WebSEAL to add the server name in the junction header.

Use the **[header-names]** *<header-data>* stanza entry to configure WebSEAL to add a header with the URI-encoded authorization API administration server name to requests for junctioned applications. If you do not configure this entry, WebSEAL does not add any headers to the request.

The *<header-data>* entry has the following format:

```
[header-names]
<header-data> = <header-name>
```

where:

<header-data>

The type of data that WebSEAL adds to the *<header-name>* header of the request.

Note: Use the value `server-name` to add the Security Access Manager authorization server name for the WebSEAL server.

<header-name>

The name of the header that holds the data.

The following value is set in the default WebSEAL configuration file.

```
[header-names]
server-name = iv_server_name
```

This setting adds a header that is called **iv_server_name** to pass the name of the server to junctioned applications. For this example, if the WebSEAL instance is `default-webseald-diamond.subnet1.ibm.com`, WebSEAL passes the following header to the junction:

```
iv_server_name:default-webseald-diamond.subnet1.ibm.com
```

Typically, the default value `iv_server_name` is used. However, you can replace it with any valid string. Valid strings are limited to the following characters: A-Z, a-z, 0–9, hyphen (-), or underscore (_).

1. Ensure that the *<header-data>* stanza entry is set with a *<header-data>* value of `server-name` in the configuration file for the WebSEAL instance. For example:

   ```
   [header-names]
   server-name = iv_server_name
   ```

2. Restart the WebSEAL server.

Credential refresh usage

This section contains the following topics:

Refreshing credentials for a specified user:

You can send a command to the WebSEAL server, instructing it to perform a credential refresh operation for all of the sessions of the specified user on the WebSEAL server.

Note: The **refresh all_sessions** command is not supported in a distributed session cache environment.

The syntax is (entered as one line):

```
pdadmin> server task instance_name-webseald-host_name
 refresh all_sessions user_name
```

Enter the above command as one continuous command line.

To obtain the server name in the correct format, use the **pdadmin server list** command. Then enter the **pdadmin** command to refresh all sessions. For example, when logged in to **pdadmin** as the administrative user sec_master:

```
pdadmin sec_master> server list
default-webseald-diamond.subnet1.ibm.com
default-webseald-cmd
pdadmin sec_master> server task default-webseald-diamond.subnet1.ibm.com
refresh all_sessions brian
DPWWA2043IThe user's credential was updated.
```

Note that the pdadmin server task command must each be entered as one continuous command line.

A warning message is returned if the user is not logged in to the WebSEAL server.

Usage notes:

- Configure credential refresh for WebSEAL before using this **pdadmin** command. See "Configure credential refresh" on page 257.
- You must issue a separate pdadmin command for each user whose credentials are to be refreshed. You cannot refresh credentials for more than one user at a time.
- The user invoking this command must have server admin (the **s** ACL bit) permission on the /WebSEAL/*hostname_instance_name* server object. This permission prevents unauthorized users from performing credential refresh operations.

  Note that the name of the *hostname_instance_name* server object is different from the server name. To determine the exact name of the server object, use **pdadmin object list**. For example, when logged in to **pdadmin** as the administrative user sec_master:

  ```
  pdadmin sec_master> object list /WebSEAL
  /WebSEAL/cmd-default
  /WebSEAL/diamond.subnet1.ibm.com-default
  ```

Troubleshooting for credential refresh:

Problem:

When a new group entry is added to a user's information in a user registry, a credential refresh command does not obtain the new entry.

Solution:

Some user registries maintain cached information. The cache is updated periodically. The cache update must take place before the credential refresh can succeed. Likewise, when using a replicated LDAP user registry, the updates to the replicated registries do not occur immediately. Wait 30 seconds and try credential refresh again. For more information, see "Refresh of cached credential information" on page 255.

External authentication interface

This chapter discusses the external authentication interface (sometimes referred to as EAI).

Topic Index:

Related concepts:

"Configuration of the certificate authentication mechanism" on page 132
You can use the External Authentication Interface (EAI) protocol to configure a junctioned web application to handle certificate authentication on behalf of WebSEAL.

"Kerberos authentication through an External Authentication Interface (EAI)" on page 143
The appliance internally supports Kerberos authentication for use with Windows clients to achieve Windows desktop single sign-on. Alternatively, you can configure a junctioned web server to handle Kerberos authentication on behalf of the appliance.

External authentication interface overview

Security Access Manager provides an external authentication interface that enables you to extend the authentication process for WebSEAL. The external authentication interface allows an independent remote service to handle the authentication process for WebSEAL. The identity information returned by the external authentication interface service is used to generate user credentials.

Security Access Manager can accept identity information from EAI for the following types of users:

- Users that exist in the Security Access Manager internal user registry.
- Users that do not exist in the Security Access Manager internal user registry, but only exist in a registry external to Security Access Manager.

This extended authentication functionality is similar to the existing custom authentication module capability provided by the Web security external authentication C API. The difference, however, is that the external authentication interface returns user identity information in HTTP response headers rather than through the authentication module interface.

When using the external authentication interface, the authentication operation is performed external to WebSEAL by a custom application located on a remote, junctioned server. The design, methodology, and code for the custom authentication application is entirely the responsibility of the application developer. This developer reference document does not provide any instructions for the construction of this custom authentication operation. However, the requirement of this application is to return identity information resulting from the custom authentication process in specially named HTTP response headers.

External authentication interface process flow

The following diagram and detailed steps illustrate the process flow for external authentication interface authentication. The components of this example process flow scenario include:

- WebSEAL.

- Junctioned server with an external authentication application that uses the external authentication interface.

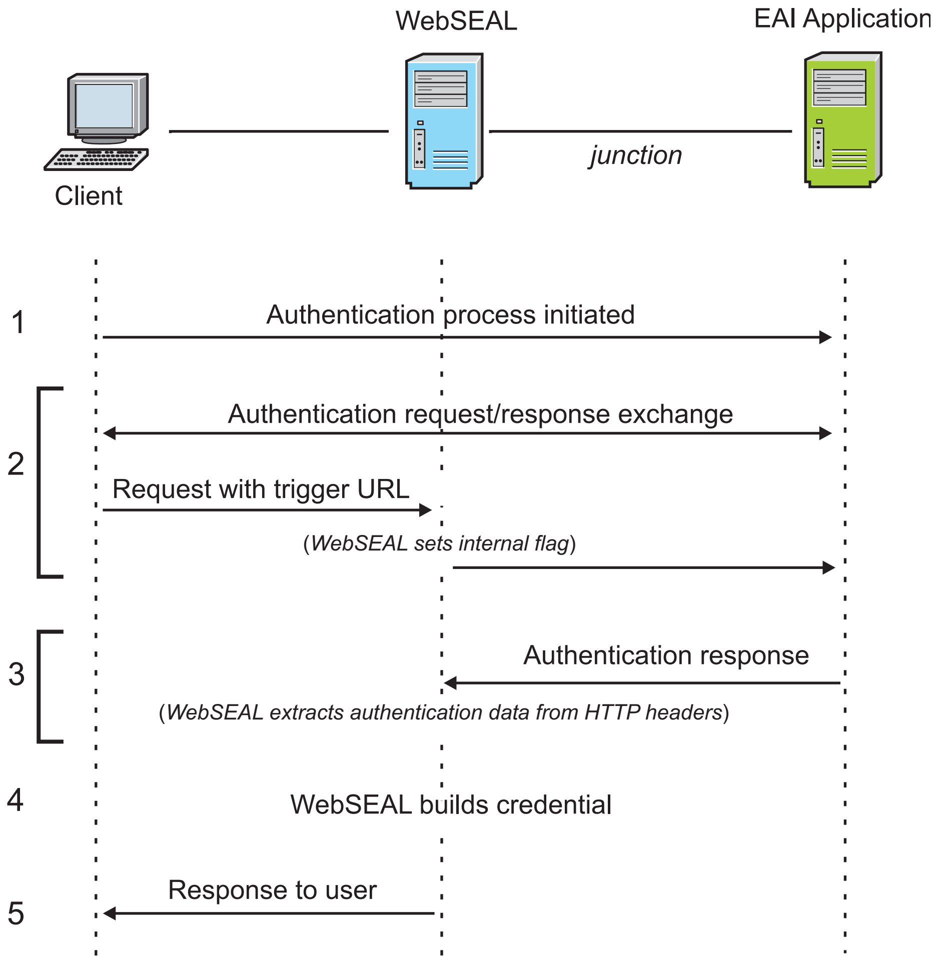

WebSEAL
EAI Application
Client
junction
1
Authentication process initiated
2
Authentication request/response exchange
Request with trigger URL
(WebSEAL sets internal flag)
3
Authentication response
(WebSEAL extracts authentication data from HTTP headers)
4
WebSEAL builds credential
5
Response to user

Figure 18. External authentication interface process flow

1. Authentication process is initiated.

   There are many possibilities for initiating the authentication process. A typical example:

   a. An unauthenticated user requests a protected resource.

   b. WebSEAL intercepts the request and returns a redirect to a customized `login.html` response page.

      The `login.html` page is customized to contain a submit link to the external authentication application.

   c. The user provides login information (user name and password) on the form and clicks the submit link to send the data to the external authentication application.

   Other examples of initiating the authentication process can include:

   - Manually typing an appropriate link to the external authentication application.
   - Signon requirement in a CDSSO environment.
   - Cached page.

- In a Federation Runtime scenario, the user is redirected to the external authentication application from a service provider, with the goal of having an identity provided for that user.

Note: A hidden configuration option enables you to give priority to an EAI header to redirect a successful login to a URL. To enable this feature, add the following option and value to the **[eai]** stanza:

```
eai-redir-url-priority = yes
```

2. Authentication request and response exchange.

   The process of authentication might require a number of exchanges between the external authentication application and the client. Exchanges are streamed through (not intercepted) by WebSEAL.

   The final authenticating request to the external authentication application must be directed to a distinct URL. This URL could, for example, include a query string that indicates the login task, such as `state=perform-login`. This final URL is specified, in part or whole, in the WebSEAL configuration file as the trigger URL. WebSEAL examines each request for this trigger URL.

   If the trigger URL is detected, WebSEAL examines the corresponding response for authentication data located in HTTP headers (specified in the WebSEAL configuration file).

   WebSEAL supports EAI logout. An HTTP header returned from the EAI enables it to instruct WebSEAL to log out a session. The header emulates the pdadmin `server task` command line input, and therefore is analogous to the hidden WebSEAL pdadmin command of the same name. The syntax for the header is `am-eai-server-task: terminate session` *<user-sess-id>*, where `terminate session` is a non-translatable keyword pair, and *<user-sess-id>* is a user session ID of the same format and contents as that used to perform the `terminate session` command using pdadmin.

   An EAI application can also return a header to log out all sessions for a specified user. The syntax for this header is `am-eai-server-task: terminate all_sessions` *<user-name>*, where `terminate all_session` is a non-translatable keyword pair, and *<user-name>* is the ID of the user to terminate all sessions for.

   Example exchange 1:

   - The user clicks a submit link on a custom login page. This link is the trigger URL.
   - The recognition of the trigger URL in the request causes WebSEAL to look for authentication data in the corresponding response.
   - The external authentication application authenticates the user and, in its response, populates the special HTTP headers with authentication data.

   Example exchange 2:

   - The external authentication application requires several exchanges with the user to receive the required login information.
   - Each request to the external authentication application uses the trigger URL. Therefore, for each response, WebSEAL looks for authentication data.
   - WebSEAL examines each corresponding response for authentication data returned from the external authentication interface in HTTP headers.
   - When no authentication takes place, these headers are empty in each response. WebSEAL continues streaming the requests and responses without taking any action.

- After several exchanges, the external authentication application receives all the login information it needs. The external authentication application authenticates the user and, in its final response, populates the special HTTP headers with authentication data.

Example exchange 3:

- The external authentication application requires several exchanges with the user to receive the required login information.
- Each request to the external authentication application uses a URL that does not match the trigger URL. Therefore, for each corresponding response, WebSEAL does not look for authentication data
- WebSEAL streams the requests and responses without taking any action.
- The final request to the external authentication application uses the trigger URL.
- The recognition of the trigger URL in this final request causes WebSEAL to look for authentication data in the corresponding response.
- The external authentication authenticates the user and, in its final response, populates the special HTTP headers with authentication data.

3. Authentication response.

   WebSEAL examines the corresponding response and finds the authentication data in the HTTP headers.
4. WebSEAL uses the authentication data to build a credential for the user.
5. WebSEAL sends a response to the user using the following precedence:
   a. If automatic redirection is enabled, the user is redirected to the location specified in the WebSEAL configuration file.

      See "WebSEAL-specified (automatic) redirection" on page 272.
   b. If the response from the external authentication application contains the streaming flag, WebSEAL streams the original response to the client.

      See "External authentication interface - authentication flags" on page 269.
   c. If the initial request was cached, the request is reprocessed for the user.

      See "Request caching with external authentication interface" on page 272.
   d. If the response from the external authentication application contains a redirection URL header, the user is redirected to the location specified by that URL.

      See "External authentication interface-specified redirection" on page 272.
   e. Otherwise, WebSEAL responds with the standard `login_success.html` page.

      See "Static server response pages" on page 58.

External authentication interface configuration

This section describes how to configure WebSEAL to use the external authentication interface.

Enabling the external authentication interface

About this task

The **eai-auth** stanza entry, located in the **[eai]** stanza of the WebSEAL configuration file, enables and disables the external authentication interface functionality. The external authentication interface can be implemented over HTTP, HTTPS, or both.

External authentication interface authentication is disabled by default.

To configure the external authentication interface:

Procedure

1. Stop the WebSEAL server.
2. Edit the WebSEAL configuration file. In the **[eai]** stanza, specify the protocols to support in your network environment. The protocols are shown in the following table.

Table 42. Configuring the external authentication interface

Protocol to Support	Configuration File Entry
HTTP	`eai-auth = http`
HTTPS	`eai-auth = https`
Both HTTP and HTTPS	`eai-auth = both`
Disable external authentication interface (default)	`eai-auth = none`

   For example, to support both protocols:

```
[eai]
eai-auth = both
```

   .

3. Restart the WebSEAL server.

Results

When `eai-auth = none` (disabled), all other configured external authentication interface-related stanza entries have no effect.

Initiating the authentication process

About this task

Typically, external authentication interface authentication can be initiated from redirection commands or custom links placed in external application pages. In an external authentication interface scenario, WebSEAL does not provide any built-in methods for initiating the authentication process. WebSEAL does not provide any special prompts or login pages.

Procedure

- You can modify WebSEAL's existing `login.html` form to include a custom link to the external authentication application. Modification of the `login.html` form is necessary to support reauthentication and authentication strength (step-up).

  See "Login page and macro support with external authentication interface" on page 274.
- You can also implement local response redirection to handle server responses to client requests. See "Local response redirection" on page 79

Configuration of the external authentication interface trigger URL

The external authentication interface authentication process supports multiple request-response exchanges. For efficiency and the security of the WebSEAL server, these exchanges are typically streamed through WebSEAL. WebSEAL intercepts this exchange only when there is an occurrence of a special *trigger URL* in a request.

A trigger URL is a server-relative or absolute URL in the WebSEAL configuration file. The trigger URL usually requests authentication from the external authentication application. For example, the trigger URL might be the URL to the external authentication application in a special link on a customized login page.

When WebSEAL detects the trigger URL in a request, it intercepts the corresponding response and examines it for authentication data in special HTTP headers.

Trigger URL strings

- Can use standard wildcard patterns. Pattern matching is appropriate only for ASCII-based strings and it is not case-sensitive.
- Must be in ASCII format if they use pattern-matching. The matching URLs in the requests must be in ASCII format.
- Must be as specific as possible in the configured URL to limit the number of times that WebSEAL intercepts the request-response exchange.

Specify trigger URL strings in the **trigger** stanza entry in the **[eai-trigger-urls]** stanza of the WebSEAL configuration file.

Virtual host junctions

- Match a trigger if their protocol, virtual host name, and port match the virtual host definition. The virtual host name matching is not case-sensitive.
- Do not use regular WebSEAL junction triggers, such as the ones that do not match a virtual host definition. Regular WebSEAL junctions do not use virtual host junction triggers.

Table 43. Examples of authentication requests to an external authentication application:

Junction type	URL	Corresponding trigger URL

standard	`http://webseal.example.com/eai-jct/login.asp?url=/return_authn_data.asp`	`[eai-trigger-urls]`
		`trigger = /eai-jct/login.asp*authn*`

virtual host	`http://vhj.webseal.example.com/login.asp?url=/return_authn_data.asp`	`[eai-trigger-urls]`
		`trigger = http://vhj.webseal.example.com/login.asp*authn*`

HTTP header names for authentication data

You must specify the names of the HTTP headers that contain the authentication data returned from the external authentication application.

There are four categories of HTTP headers that hold authentication data:

- **Privilege Attribute Certificate (PAC) format**

  The PAC is an ASN.1 data structure used to express identity information. Authentication data returned to WebSEAL in PAC format can be directly converted to a credential.
- **WebSEAL user identity structure**

  The WebSEAL user identity structure is the same structure generated by WebSEAL's default built-in authentication modules. When the user identity format type is used, the information is processed by the **eaiauthn** authentication module and a credential is built by the Security Access Manager authorization API.
- **Distributed session cache session identifier**

The session identifier is for a distributed session that is managed by the distributed session cache. See "Sharing sessions across multiple DNS domains" on page 330.

- **WebSEAL external user identify structure**

  Security Access Manager can accept identity information from the EAI for external users; that is, users that only exist in a registry external to Security Access Manager. The `eai-xattrs-header` entry also applies to external users. See "External authentication interface overview" on page 260. For more information about the `[eai]` stanza, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.
- **Common**

  The common header category holds additional information and can be used with either the PAC or user identity formats.

Complete details about these special headers can be found in the "External authentication interface HTTP header reference" on page 270.

Use the **`[eai]`** stanza of the WebSEAL configuration file to specify the names of the HTTP headers that contain the authentication data returned from the external authentication interface server. The header names can be customized. The custom external authentication interface authentication module must be written to use the header names as configured.

The following examples show the default header names used in the WebSEAL configuration file:

PAC headers:

```
[eai]
eai-pac-header = am-eai-pac
eai-pac-svc-header = am-eai-pac-svc
```

User identity headers:

```
[eai]
eai-user-id-header = am-eai-user-id
eai-auth-level-header = am-eai-auth-level
eai-xattrs-header = am-eai-xattrs
```

External user identity headers:

```
[eai]
eai-ext-user-id-header = am-eai-ext-user-id
eai-ext-user-groups-header = am-eai-ext-user-groups
```

Distributed session cache session identifier:

```
[eai]
eai-session-id-header = am-eai-session-id
```

Common headers:

```
[eai]
eai-flags-header = am-eai-flags
eai-redir-url-header = am-eai-redir-url
```

For more information about using the **eai-flags-header** common header, see "External authentication interface - authentication flags" on page 269

For more information about using the **eai-redir-url-header** common header, see "External authentication interface-specified redirection" on page 272.

Extracting authentication data from special HTTP headers

About this task

WebSEAL examines a response for special headers when a trigger URL is detected in the corresponding request.

The special HTTP headers contain authentication data provided by the custom external authentication application. The presence of either the PAC header or the user identity header causes WebSEAL to extract the authentication data from the headers and build a credential for the user. The session identifier header causes WebSEAL to retrieve the specified session from the distributed session cache.

WebSEAL follows a specific sequence for processing the special HTTP authentication headers:

Procedure

1. If the session identifier header is present, it takes precedence over the other authentication headers.
2. If both headers are present, the PAC data takes precedence and any user identity data is ignored.
3. If neither header is present, the response is streamed back to the client. This behavior also allows the external authentication application to perform authentication error handling.
4. If either the PAC or user identity header is present, but the header value is NULL or corrupted, an error is returned. Such an error can occur if an external authentication interface server is incorrectly configured.

How to generate the credential

WebSEAL can build a credential directly from PAC header data. The authorization API builds the credential for user identity header data.

Other authentication data can be supplied by the WebSEAL system itself when building a credential from user identity authentication data. WebSEAL has additional information about the client system that is required to construct the credential. This information is supplied when authentication data from the external authentication interface is used to generate a credential.

Some of these values can be overridden by the **eaiauthn** module using extended attributes to the header data.

Table 44. Supplemental credential data provided by WebSEAL

Field	Source	Can external authentication interface override value?
Client IP Address	Derived from the initial client request.	yes
Browser Information	Derived from the initial client request.	yes
Registry Type	Determined from the current WebSEAL configuration.	no
Domain	Determined from the current WebSEAL configuration.	no

External authentication interface credential replacement

WebSEAL allows a previously authenticated user to request authentication again through the external authentication interface trigger URL and establish a new session. WebSEAL deletes the old session cache entry, builds a new session cache entry containing a new credential for that user (credential replacement), and provides the user with a new session key.

Operation conditions for external authentication interface credential replacement:

- If a trigger URL is used by a previously authenticated user to make a request, that request is allowed to pass through to the external authentication application.

  Note: In earlier versions of the external authentication interface, a previously authenticated user was forced to log out and log in again when making a request using a trigger URL.
- If the external authentication interface response to the user request contains authentication data, and the user's session cache entry is flagged for authentication strength policy (step-up) or reauthentication, then WebSEAL enforces the step-up or reauthentication process. The existing session cache (and credential) for the user is not replaced.
- If the external authentication interface response to the user request contains authentication data, and the user's cache entry is not flagged as step-up or reauthentication, then:
  - The existing session cache entry is deleted and replaced with a new entry containing a new credential for the user.
  - If the user uses session cookies to maintain session state, a new session key is created and returned to the user.
  - If the user uses SSL session IDs or HTTP headers to maintain session state, the existing session key is reused.
  - If a failover cookie is used, a new failover cookie is created and returned to the user.
  - If user session IDs are used, the user session ID mapping to the WebSEAL session ID is updated.
  - If an LTPA cookie is used, a new LTPA cookie is created and returned to the user.

The external authentication interface credential replace function is important to support, for example, the account-linking features that the Liberty federate function provides. A Federation Runtime environment requires the ability to reauthenticate a previously authenticated user to achieve the Liberty federate function (Liberty Alliance Project). A federate operation allows a local account at a service provider to be linked with an account at an identity provider.

To achieve this, a user must first sign into the user's service provider and consent to linking the user's account with the identity provider. Once the federate operation has occurred, the browser focus returns to the service provider where the user's credential is updated with the new credential generated by the identity provider.

Validating the user identity

EAI applications can re-authenticate a user by returning new authentication information for a previously authenticated session. By default, WebSEAL does not validate this new authentication information. However, you can configure WebSEAL to verify that the user identity does not change during subsequent EAI authentications.

WebSEAL uses the principal name (**azn_cred_principal_name** attribute) to validate the user identity. The principal name that is contained in the newly constructed credential is compared with the principal name contained in the existing credential. If the two names differ, the validation process fails and WebSEAL returns an authentication error to the user.

To validate user identities during subsequent EAI authentication operations, set the **eai-verify-user-identity** stanza entry to `yes`. This entry is located in the **[eai]** stanza of the WebSEAL configuration file:

```
[eai]
eai-verify-user-identity = yes
```

How to write an external authentication application

The design, methodology, and code for the external authentication application is entirely the responsibility of the application developer. This developer reference document does not provide any instructions for the construction of this authentication operation.

However, the following conditions for the operation of external authentication interface should be considered when developing the custom application:

- The external authentication interface server is junctioned to WebSEAL.
- Identity information resulting from the custom authentication process is returned to WebSEAL in specially named HTTP response headers (as configured in the WebSEAL configuration file).
- Multi-step authentications are allowed.
- The external authentication application must be available to unauthenticated users.
- WebSEAL checks its user registry for credential information. Therefore, the external authentication application must either share the same registry with WebSEAL, or the external authentication application must return user information that matches an entry in the WebSEAL user registry.

External authentication interface - authentication flags:

When an EAI application performs a successful authentication, it constructs and returns the response to a trigger URL. WebSEAL detects this authentication information in the trigger URL response. You can provide authentication flags with this response to help control the authentication processing by WebSEAL.

These authentication flags are contained in the HTTP header. Use the **eai-flags-header** stanza entry in the **[eai]** stanza of the WebSEAL configuration file to specify name of the flags header.

WebSEAL supports the following flags:

stream

By default, WebSEAL replaces the EAI-generated response with a

WebSEAL-generated response for the authentication operation. You can override this default behavior and configure WebSEAL to stream the EAI-generated response back to the client. That is, after a successful EAI authentication, WebSEAL can strip the EAI-specific headers from the response and stream it back to the client.

To achieve this EAI response streaming, the flags header must contain the **stream** flag.

Example EAI flags header:

```
am-eai-flags: stream
```

The **eai-flags-header** configuration entry specifies the name of the HTTP header that contains the flags. For example:

```
[eai]
eai-flags-header = am-eai-flags
```

External authentication interface HTTP header reference

Table 45. PAC headers

Description	Stanza Entry	Default Header Name	Required	Notes

PAC	`[eai]`
	`eai-pac-header`
		`am-eai-pac`	yes	Authentication data in PAC format. Direct conversion to credential.
				This header takes precedence over the user identity header.
				Place this header before others in the response headers.

PAC Service ID	`[eai]`
	`eai-pac-svc-header`
		`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.

Table 46. User identity headers

Description	Stanza Entry	Default Header Name	Required	Notes

User Identity	`[eai]`
	`eai-user-id-header`
		`am-eai-user-id`	yes	The ID of the user to generate the credential for.
				This header should precede all others in the HTTP response.

Authentication Level	`[eai]`
	`eai-auth-level-header`
		`am-eai-auth-level`	no	The authentication strength level for the generated credential.
				If no value is specified, a default value of 1 is used.

Table 46. User identity headers (continued)

Description	Stanza Entry	Default Header Name	Required	Notes
Extended Attribute List	[eai] eai-xattrs-header	am-eai-xattrs	no	A comma delimited list of HTTP header names that should be added to the credential as extended attributes. If attributes of the same name are specified by a custom authentication module build with the external authentication C API, the attributes from the custom module take precedence over the HTTP header attributes.
External user identity	[eai] eai-ext-user-id-header	am-eai-ext-user-id	no	Specifies the name of the header that contains the ID of the external (not in the Security Access Manager user registry) user to use when creating a credential.
External group identity	[eai] eai-ext-user-groups-header	am-eai-ext-user-groups	no	Specifies the name of the header that contains the group or groups an external user is to be considered a member of when generating a credential. This entry is only used when the `eai-ext-user-id-header` stanza entry's value is provided.

Table 47. Session identifier headers

Description	Stanza Entry	Default Header Name	Required	Notes
Session Identifier	[eai] eai-session-id-header	am-eai-session-id	yes	The identify of the distributed session managed by the Session Management Server.

Table 48. Common headers

Description	Stanza Entry	Default Header Name	Required	Notes
Redirect URL	[eai] eai-redir-url-header	am-eai-redir-url	no	Only used if WebSEAL does not have a cached request or when automatic redirection is not enabled. Specifies the URI that the client is redirected to upon successful authentication. If no URI is specified, the "login-success" page is returned.
Flags header	[eai] eai-flags-header	am-eai-flags	no	The only supported flag is **stream**. Example: `am-eai-flags: stream`

Use of external authentication interface with existing WebSEAL features

This section contains the following topics:

Request caching with external authentication interface

Server-side request caching occurs for external authentication interface authentication when WebSEAL returns a login prompt as a consequence of:

- An unauthenticated user requesting a protected resource.
- An authenticated user requesting a resource protected by reauthentication.
- An authenticated user requesting a resource protected by authentication strength policy (step-up).

When one of these events occurs, WebSEAL caches the initial request. WebSEAL retains the cached request during any authentication interaction with the external authentication application. WebSEAL reprocesses the cached request only when an authentication has succeeded.

No modifications are necessary to support standard request caching for authentication using the external authentication interface.

See "Server-side request caching" on page 235.

Post-authentication redirection with external authentication interface

WebSEAL-specified (automatic) redirection:

If automatic redirection during user login is enabled through WebSEAL, clients are redirected to the specified resource upon successful external authentication interface authentication.

```
[enable-redirects]
redirect = ext-auth-interface
```

See "Automatic redirection after authentication" on page 231.

External authentication interface-specified redirection:

The external authentication application can be written to send a special HTTP header in the authentication response that specifies a redirection URL. Upon successful authentication, the client is redirected to this URL.

This optional header is configured in the same manner as other special external authentication interface headers (see "HTTP header names for authentication data" on page 265).

For example:

```
[eai]
eai-redir-url-header = am-eai-redir-url
```

Session handling with external authentication interface

The existing options for maintaining sessions, handling session cookies, and configuring session cache parameters apply for external authentication interface authentication.

Authentication strength level with external authentication interface

Authentication strength policy (step-up authentication) is supported for external authentication interface authentication.

```
[authentication-levels]
level = ext-auth-interface
```

See "Authentication strength policy (step-up)" on page 186.

You can associate an authentication strength level with an authentication performed by an external authentication interface module. An optional HTTP header can be returned by the external authentication interface module to specify this authentication level.

This header is configured in the same manner as other special external authentication interface headers (see "HTTP header names for authentication data" on page 265).

For example:

```
[eai]
eai-auth-level-header = am-eai-auth-level
```

The authentication strength level value becomes an attribute of the identity structure and the resulting credential. The authentication strength level attribute allows you to implement step-up authentication functionality by operating multiple external authentication interface authentication modules on a single external authentication interface server. Each module can process a different authentication method.

If the authentication strength level does not exist or contains an empty value, the default mechanisms for assigning an authentication level are used.

You must modify the standard WebSEAL login pages appropriately if you enable step-up authentication with external authentication interface authentication. See "Login page and macro support with external authentication interface" on page 274.

Reauthentication with external authentication interface

Reauthentication is supported for external authentication interface authentication.

Reauthentication requires that the method used by the client to reauthenticate is the same as that used by the client to initially authenticate. When WebSEAL receives the authentication response from the custom external authentication application, a check is performed (as with other reauthentication processing) to ensure:

- The authentication method used is the same as that used to create the initial credential
- The user name matches

- Any external authentication interface-specified authentication level is verified to match the existing level

You must modify the standard WebSEAL login pages appropriately if you enable reauthentication with external authentication interface authentication. See "Login page and macro support with external authentication interface."

Login page and macro support with external authentication interface

The WebSEAL login pages can be modified to cause a redirection to the external authentication interface server to perform the authentication, or to contain a link (or button) that a user can click to initiate the authentication exchange with the external authentication interface server. This modified login page is required if you enable reauthentication or step-up to external authentication interface.

An external authentication interface-specific macro (%EAIAUTHN%) is used to selectively add or mask sections from the `certlogin.html` and `stepuplogin.html` login forms. When the authentication method (indicated by the macro name) is valid, the section in the form governed by the macro is displayed. When the authentication method is not valid, the macro is replaced by a start comment delimiter (<!--). All subsequent information in the form is commented out until a comment closing delimiter (-->) is reached.

To facilitate the passing of the required authentication level for step-up as an argument in a query string, WebSEAL passes another macro (%AUTHNLEVEL%) to the `stepuplogin.html` login form.

Neither of these macros are present in the default login forms. The macros must be manually added.

You can also implement local response redirection to handle server responses to client requests.

Setting a client-specific session cache entry lifetime value

About this task

The **timeout** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, globally sets the maximum lifetime timeout value for all client session information stored in the WebSEAL session cache. You can override this global lifetime value with a per-client lifetime value that is provided as a header in the authentication response from an external authentication interface service. This value is extracted by WebSEAL and stored as an extended attribute in the user's credential.

WebSEAL receives the client-specific timeout information as the value of a header in the authentication response from the external authentication interface. WebSEAL uses the value of that header to set the lifetime timeout of the new session cache entry for that client. This value overrides the value of the **timeout** stanza entry.

The value must represent an absolute time expressed as the number of seconds since 00:00:00 UTC, January 1, 1970. The output of the UNIX **time ()** function, for example, represents the correct format of this absolute time value.

The following steps summarize the necessary configuration for setting a client-specific cache entry lifetime timeout value:

Procedure

1. Configure the custom external authentication interface program to provide, in its authentication response, an HTTP header containing the session cache lifetime timeout value appropriate for that client. The required name of this header is:

   ```
   am_eai_xattr_session_lifetime
   ```

   Note: The name of this particular header is not configurable.
   For example:

   ```
   am_eai_xattr_session_lifetime:1129225478
   ```
2. Configure the custom external authentication interface program to additionally provide an HTTP header that specifies a comma-delimited list of HTTP header names that contain extended attribute values.
   You must configure WebSEAL to look for this header name (see step 4). The default name for this header is **am-eai-xattrs**. (The **am-eai-xattrs** header name is configurable.)
3. Configure the custom external authentication interface program to include the **am_eai_xattr_session_lifetime** header name as a value to the **am-eai-xattrs** header. For example:

   ```
   am-eai-xattrs: am_eai_xattr_session_lifetime
   ```
4. Use the **[eai]** stanza of the WebSEAL configuration file to specify the names of the HTTP headers that contain authentication data returned from the external authentication interface server.
   In the **[eai]** stanza, ensure that WebSEAL looks for the **am-eai-xattrs** header name:

   ```
   [eai]
   eai-xattrs-header = am-eai-xattrs
   ```

   Note: Header names used for the external authentication interface can be customized. Ensure that the custom external authentication interface module is written to use the header names as configured.

Results

If the **am_eai_xattr_inactive_timeout** header is present in a flagged response from the external authentication interface, WebSEAL adds the value to the user's credential as an extended attribute. The entry in the credential for this example appears as follows:

```
am_eai_xattr_session_lifetime:1129225478
```

After the credential is successfully built, WebSEAL creates and entry in the session cache for that client and uses the value of the extended attribute to set the inactivity timeout for that client's session cache entry.

If the **am_eai_xattr_session_lifetime** header is not supplied. WebSEAL uses the default timeout value provided by the **timeout** stanza entry.

Example:

For example, in aFederation Runtime environment, there is an optional element of a Liberty authentication response that is used by an identity provider to dictate to a service provider the duration of a user's session at the service provider.

By modifying the external authentication interface used to authenticate users, a single attribute (the value derived from the identity provider token) can be returned to WebSEAL and used to set the lifetime timeout of session cache entry for that user. The service provider should always request a new single signon interaction with the identity provider once this cache entry lifetime value has expired.

See also:

- "Customized responses for old session cookies" on page 315
- "Cache entry lifetime timeout value" on page 287
- "Setting a client-specific session cache entry inactivity timeout value"

Setting a client-specific session cache entry inactivity timeout value

About this task

The **inactive-timeout** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, globally sets the maximum lifetime of inactive entries contained within the WebSEAL session cache. You can override this global inactivity timeout value with a per-client value that is provided as a header in the authentication response from an external authentication interface service. This value is extracted by WebSEAL and stored as an extended attribute in the user's credential.

WebSEAL receives the client-specific inactivity timeout information as the value of a header in the authentication response from the external authentication interface. WebSEAL uses the value of that header to set the inactivity timeout of the new session cache entry for that client. This value overrides the value of the **inactive-timeout** stanza entry.

The value represents the maximum number of seconds that the session can be inactive before it is removed from the WebSEAL session cache.

The following steps summarize the necessary configuration for setting a client-specific cache entry inactivity timeout value:

Procedure

1. Configure the custom external authentication interface program to provide, in its authentication response, an HTTP header containing the session cache inactivity timeout value appropriate for that client. The required name of this header is

   ```
   am_eai_xattr_session_inactive_timeout
   ```

   Note: The name of this particular header is not configurable. For example:

   ```
   am_eai_xattr_session_inactive_timeout:120
   ```
2. Configure the custom external authentication interface program to additionally provide an HTTP header that specifies a comma-delimited list of HTTP header names that contain extended attribute values.

   You must configure WebSEAL to look for this header name (see step 4). The default name for this header is **am-eai-xattrs**. (The **am-eai-xattrs** header name is configurable.)

3. Configure the custom external authentication interface program to include the **am_eai_xattr_session_inactive_timeout** header name as a value to the **am-eai-xattrs** header. For example:

   ```
   am-eai-xattrs: am_eai_xattr_session_inactive_timeout
   ```

4. Use the **[eai]** stanza of the WebSEAL configuration file to specify the names of the HTTP headers that contain authentication data returned from the external authentication interface server.
   In the **[eai]** stanza, ensure that WebSEAL looks for the **am-eai-xattrs** header name:

   ```
   [eai]
   eai-xattrs-header = am-eai-xattrs
   ```

   Note: Header names used for the external authentication interface can be customized. Ensure that the custom external authentication interface module is written to use the header names as configured.

Results

If the **am_eai_xattr_session_inactive_timeout** header is present in a flagged response from the external authentication interface, WebSEAL adds the value to the user's credential as an extended attribute. The entry in the credential for this example appears as follows:

```
am_eai_xattr_session_inactive_timeout:120
```

After the credential is successfully built, WebSEAL creates an entry in the session cache for that client and uses the value of the extended attribute to set the inactivity timeout for that client's session cache entry.

If the **am_eai_xattr_session_inactive_timeout** header is not supplied, WebSEAL uses the default timeout value provided by the **inactive-timeout** stanza entry.

See also:

- "Customized responses for old session cookies" on page 315
- "Cache entry lifetime timeout value" on page 287
- "Setting a client-specific session cache entry lifetime value" on page 274

Chapter 4. Session State

Session state overview

This chapter discusses basic concepts of how WebSEAL maintains session state.

Topic Index:

Session state concepts

A client/server session is a series of related interactions between a single client and a server that take place over a period of time. With an established session, the server can identify the client associated with each request, and has the ability to remember—over numerous requests—a specific client.

Without an established session, the communication between the client and the server must be renegotiated for each subsequent request. Session state information improves performance in the following ways:

- For client authentication methods such as basic authentication, where authentication data is included with every request to the WebSEAL server, session state information eliminates the need to validate the user name and password with every request.
- For other client authentication methods that require prompting the user to log in, session state information eliminates the need to prompt the user to log in with every request to the WebSEAL server. The client can log in once and make numerous requests without performing a separate login for each request.

Supported session ID data types

WebSEAL can maintain session state with both HTTP and HTTPS clients. The SSL transport protocol is specifically designed to provide a session ID to maintain session state information. In contrast, HTTP is a "stateless" protocol and does not provide any means of distinguishing one request from another. (HTTP communication can be encapsulated over SSL to become HTTPS.)

However, WebSEAL must often handle HTTP communication from unauthenticated clients. There are also times when the SSL session ID is not an appropriate solution.

To maintain session state with clients over HTTP or HTTPS, WebSEAL can use one of several data types to provide a client-identifying session key, known as the WebSEAL session ID.

WebSEAL maintains the specific client identity and session information in a session cache. Each session cache entry is indexed by a session key (the WebSEAL session ID).

The following supported data types can provide the session key used by WebSEAL to maintain session state with a client:

- SSL session ID (defined by the SSL protocol)
- Server-specific session cookie

When WebSEAL examines a client request, it searches for the session key in the order specified in this list.

Information retrieved from a client request

Session identification is the process of examining the information associated with an HTTP request (such as the URL, HTTP headers and cookies, IP address, and SSL session ID) to retrieve a session ID that can be used to associate a particular client with the request.

WebSEAL examines a client request for the following information:

- **Session key**

  A session key is information that identifies a specific connection between the client and the WebSEAL server. The session key is stored with the client and accompanies subsequent requests by that client. It is used to re-identify the client session to the WebSEAL server and avoid the overhead of establishing a new session for each request. The session key is a locator index to the associated session data stored in the WebSEAL server session cache. The session key is also known as the WebSEAL session ID.
- **Authentication data**

  Authentication data is information from the client that identifies the client to the WebSEAL server. Examples of authentication data types include client-side certificates, passwords, and token codes.

When WebSEAL receives a client request, WebSEAL always looks for the session key and associated session data first, followed by authentication data.

Validation of the client identifier for a session

You can configure IBM Security Access Manager to validate the client identifier to ensure that different clients do not attempt to use the session.

A client identifier can be the client's IP address or the contents of a configured HTTP header. The client identifier is associated with the session when the session is first established. WebSEAL then checks the client identifier on subsequent requests to ensure that a different client is not attempting to access the session.

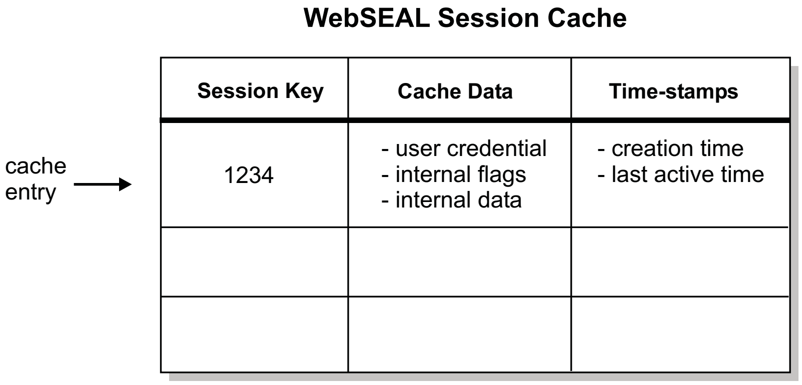
If the client is able to connect directly to the WebSEAL server, the IP address of the client can be used to identify the client. However, if the WebSEAL traffic is routed through a network terminating firewall, the contents of an HTTP header (for example, the **X-Forwarded-For** header) can be used to identify the client.

You can configure the client identifier to be validated for a session with the **client-identifier** stanza entry. This identifier is added to the credential as the **client_identifier** attribute and is validated on subsequent requests to ensure that the client does not change. See client-identifier for more information.

Note: If failover cookies are used, add the **client_identifier** credential attribute to the failover cookie by modifying the **[failover-add-attributes]** and **[failover-restore-attributes]** stanzas so that the client identifier can persist across a failover event. This step prevents an attacker from establishing a new session from a different client with the failover cookie.

WebSEAL session cache structure

The WebSEAL session cache can be represented as an internal table where WebSEAL stores information about all sessions established by authenticated users. The session key, stored with the client, is a locator index to the associated session data stored in the WebSEAL session cache.



WebSEAL Session Cache

Session Key	Cache Data	Time-stamps
1234	- user credential - internal flags - internal data	- creation time - last active time

cache entry

Figure 19. WebSEAL session cache

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

Each cache entry contains the following types of information:

- **Session key**

  The session key (the WebSEAL session ID) is a unique identifier, or key, that is sent with each request made by that user. The session key identifies the specific cache entry for that user.
- **Cache data**

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

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

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

The **user credential** is an encoded opaque data structure representing the authenticated user. The credential contents can include:

- User name
- Group memberships
- Extended attributes

  Extended attributes allow you to store customized data in the user credential.

Deployment considerations for clustered environments

Consider the following topics when you deploy multiple replica WebSEAL servers in a clustered environment for fault-tolerance or performance reasons:

Consistent configuration on all WebSEAL replica servers

To maintain a consistent user experience regardless of which WebSEAL server a client accesses, all WebSEAL replica servers must be identically configured.

For example, if a junction exists on one WebSEAL server and not on another, clients can receive errors when they access the WebSEAL server that does not have the proper junction definition. All configuration (for example, dynamic URLs, junction mapping table, authentication, and authorization) must be identical across all the WebSEAL servers in the cluster.

The **server-name** configuration option in the **[server]** stanza of the WebSEAL configuration file can be used to force all WebSEAL servers to perform authorization checks on the same protected object space. This configuration allows you to apply ACLs and POPs only once. Most other WebSEAL configuration options must be set individually for every server in the cluster.

Client-to-server session affinity at the load balancer

Whenever possible, load balancers should be configured to maintain session affinity.

Session affinity provides improved performance, improved user experience, and simplifies WebSEAL configuration.

Failover to a new master

If the cluster master becomes unavailable for an extended period, re-configure the slaves to use a different master. To configure the new master, modify the **master-name** configuration entry in the **[cluster]** stanza for each slave WebSEAL server. Ensure that the newly designated master has the most up-to-date configuration.

Failover from one WebSEAL server to another

When clients failover from one WebSEAL server to another, there must be some mechanism for the new WebSEAL server to identify the client.

Security architects must choose from several possible options for handling failover events. Each option involves different trade-offs in complexity, security, and performance.

In addition, some options for handling failover events can provide additional functionality, such as single-signon or more flexible session management tools for use by software support personnel and WebSEAL administrators.

Options for handling failover in clustered environments

WebSEAL offers several solutions to the challenge of providing secure sharing of session state across multiple servers in a clustered environment. The following sections describe the options available for handling failover events in clustered environments:

Option 1: No WebSEAL handling of failover events

If the load balancer in front of the WebSEAL cluster is able to maintain session affinity for long periods of time, failover events can be very rare.

When failover events do occur, clients are forced to log in again.

This option is relatively easy to configure, but contains the risk of a poor user experience if a WebSEAL server becomes unavailable for any reason or if the load balancer is unable to maintain session affinity.

Option 2: Authentication data included in each request

Some authentication methods such as basic authentication or client-side certificates provide authentication data with every request.

If the WebSEAL servers in the cluster are configured to use such an authentication method, then failover events result in automatic authentication of the user without prompting the user to login again.

This method of handling failover events is relatively easy to configure and provides a good user experience. However, this method does not allow the use of certain WebSEAL features such as reauthentication and **pkmslogout**.

Option 3: Failover cookies

The failover cookie is a mechanism for transparently reauthenticating the user and is not actually a mechanism for maintaining sessions. Failover cookies contain encrypted user authentication data that a WebSEAL server can use to validate a user's identity. A failover cookie maintains the following information:

- User credential information
- Session inactivity timeout value
- Session lifetime timeout value

All other session state data, however, is not captured or maintained by failover cookies.

Failover cookie configuration requires the distribution of a shared secret key to all of the WebSEAL servers in the cluster, and requires more configuration than the first two options discussed.

Failover cookies pose a greater security risk than normal session cookies. If an attacker hijacks a session cookie, the session cookie is only valid until the WebSEAL server deletes the associated session. Failover cookies are valid until the lifetime or inactivity timeout in the failover cookie is reached.

Failover cookies do allow the enforcement of session lifetime timeouts, inactivity timeouts, and **pkmslogout**. Failover cookies can also provide single-signon across multiple WebSEAL clusters in the same DNS domain.

For further information on the failover cookie mechanism, see “Failover solutions” on page 292.

Option 4: The distributed session cache

The distributed session cache is used for session storage by all WebSEAL servers in the cluster. When a client fails over, the new WebSEAL server can retrieve the user's session data from the distributed session cache and therefore avoid prompting the user to log in again.

Like failover cookies, the distributed session cache allows consistent inactivity and lifetime timeout tracking across all of the WebSEAL servers in the cluster. Also like failover cookies, the distributed session cache allows for single-signon across multiple WebSEAL clusters in the same DNS domain.

The distributed session cache reduces the security risk that is posed by the failover cookie, since only a normal session cookie is used.

The distributed session cache also provides extra features that are not available with any other method of maintaining session state across server clusters. For example, the distributed session cache allows customer support personnel and WebSEAL administrators to view all of the users who are logged in to the cluster at a given time.

The distributed session cache also supports a **max-concurrent-web-sessions** policy that limits the number of concurrent sessions that are allowed per user.

For more information about the distributed session cache, see "Advanced configuration for the distributed session cache" on page 338.

Option 5: LTPA cookie

The failover cookie is primarily a mechanism for transparently authenticating the user and is not actually a mechanism for maintaining sessions. LTPA cookies contain encrypted user authentication data that a WebSEAL server can use to validate a user's identity. An LTPA cookie maintains the following information:

- User name
- Session lifetime timeout value

All other session state data, however, is not captured or maintained by LTPA cookies. LTPA cookie configuration requires the distribution of a shared secret key to all of the servers in the cluster, and requires more configuration than the first two options discussed.

LTPA cookies pose a greater security risk than normal session cookies. If an attacker hijacks a session cookie, the session cookie is only valid until the WebSEAL server deletes the associated session. LTPA cookies are valid until the lifetime timeout in the LTPA cookie is reached.

LTPA cookies do allow the enforcement of session lifetime timeouts, and **pkmslogout**. LTPA cookies can also provide single-signon across multiple WebSEAL clusters in the same DNS domain, along with single-signon across other LTPA-enabled servers in the same DNS domain (for example, WebSphere Application Server, DataPower).

If you are using a cookie-based failover approach, you should use the failover cookie, mentioned in option 3, over the LTPA cookie option. The LTPA cookie is mostly designed to enable single-signon to third-party servers (for example WebSphere Application Server, DataPower).

For further information on the LTPA cookie mechanism, see "LTPA authentication" on page 158.

Session cache configuration

This chapter talks about configuring the SSL session cache and the WebSEAL session cache.

Topic Index:

Session cache configuration overview

A session cache allows a server to store session information from multiple clients. WebSEAL uses two types of session caches to accommodate both HTTPS and HTTP session state information between clients and WebSEAL:

- **WebSEAL session cache**

  The WebSEAL session cache stores information about all sessions established by authenticated and unauthenticated users. The session key, stored with the client, is a locator index to the associated session data stored in the WebSEAL session cache.

  The WebSEAL session cache stores, among other data, the credential information obtained for each client. Credential information is cached to eliminate repetitive queries to the user registry database during authorization checks.

- **SSL session ID cache**

  The SSL session cache stores the SSL session ID used to maintain SSL session state.

  SSL session IDs can be used as the session index for the WebSEAL session cache.

Configuration file entries for configuring the WebSEAL session cache and the SSL session ID cache are summarized in the following diagram:

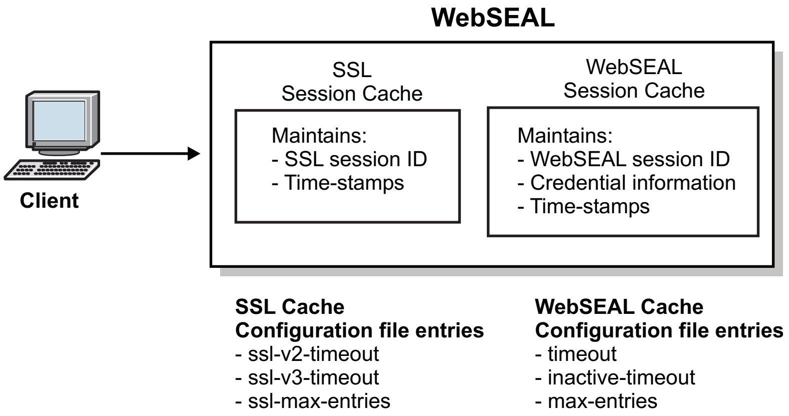


Figure 20. Session cache configuration file entries

For an overview of session state concepts, see “Session state overview” on page 279.

SSL session ID cache configuration

The following configuration tasks are available for the SSL session ID cache:

Cache entry timeout value

The stanza entries for setting the maximum lifetime timeout for an entry in the SSL session ID cache are located in the **[ssl]** stanza of the WebSEAL configuration file. There are two stanza entries: one for SSL v2 connections (**ssl-v2-timeout**) and one for SSL v3 connections (**ssl-v3-timeout**). The SSL v3 session timeout is also used for TLS v1 connections.

The default SSL v2 session timeout (in seconds) is 100 (with a possible range of 1-100):

```
[ssl]
ssl-v2-timeout = 100
```

The default SSL v3 session timeout (in seconds) is 7200 (with a possible range of 1-86400):

```
[ssl]
ssl-v3-timeout = 7200
```

Maximum concurrent SSL sessions value

The **ssl-max-entries** stanza entry, located in the **[ssl]** stanza of the WebSEAL configuration file, sets the maximum number of concurrent SSL sessions in the SSL session ID cache.

This value limits the number of SSL sessions the WebSEAL server tracks at any given time. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm. If a client whose SSL session was discarded contacts the WebSEAL server again, WebSEAL automatically negotiates a new SSL session with the client.

If SSL session IDs are being used as the session index for the WebSEAL session cache, the client's WebSEAL session ID changes because of the renegotiation. The client must reauthenticate to WebSEAL.

The default number of concurrent SSL sessions is 4096:

```
[ssl]
ssl-max-entries = 4096
```

WebSEAL session cache configuration

WebSEAL maintains two separate session caches, one for authenticated users and the other for users who are in the process of authenticating. Once a user is authenticated, their session cache entry is moved from the unauthenticated session cache to the authenticated session cache.

The following sections describe configuration and use of WebSEAL session caches:

Maximum session cache entries value

The **max-entries** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, specifies the maximum number of session cache entries in the WebSEAL unauthenticated and authenticated session caches.

This value corresponds to the number of concurrent login sessions. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm to allow new incoming logins.

The following conditions affect the specified value:

- If the specified value is less than or equal to 0, the cache size becomes unlimited.
- If the specified value is between 0 and 8192, the actual number of entries allowed is rounded up to the next multiple of 32.
- Any specified value greater than 8192 is accepted as given.

WebSEAL does not impose a maximum value. See the guidelines on maximum size of integer values in Chapter 12, “Guidelines for changing configuration files,” on page 623.

The default number of concurrent login sessions is 4096:

```
[session]
max-entries = 4096
```

The value for a particular session cache (either unauthenticated or authenticated) can be supplied by prefixing the configuration entry with the session cache name (either `unauth` or `auth`). For example:

```
unauth-max-entries = 1024
```

Cache entry lifetime timeout value

The **timeout** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, sets the maximum lifetime timeout value for all user session information stored in the WebSEAL authenticated or unauthenticated session caches.

WebSEAL caches credential information internally, so the session cache **timeout** stanza entry dictates the length of time authorization credential information remains in memory on WebSEAL.

The stanza entry is not an inactivity timeout. The value maps to a "credential lifetime" rather than a "session inactivity timeout". Its purpose is to enhance security by forcing the user to reauthenticate when the specified timeout limit is reached.

The default session cache entry lifetime timeout (in seconds) is 3600:

```
[session]
timeout = 3600
```

The value for a particular session cache (either unauthenticated or authenticated) can be supplied by prefixing the configuration entry with the session cache name (either `unauth` or `auth`). For example:

```
unauth-max-entries = 1024
```

WebSEAL does not impose a maximum value for this stanza entry.

A value of "0" disables this timeout feature (lifetime value is unlimited). The control of cache entries is then governed by the **inactive-timeout** and **max-entries** stanza entries.

When a cache is full, the entries are cleared based on a least-recently-used algorithm. See “Maximum session cache entries value” on page 286.

Note: This stanza entry is ineffective for authentication methods that include authentication data in every request to the WebSEAL server, such as basic authentication (BA), SPNEGO, and some forms of certificate authentication. Those authentication methods automatically reauthenticate the user to the WebSEAL server if the user's session has been deleted due to inactivity or lifetime timeouts. The result is repeated resetting of the inactive and lifetime timeout values.

Tip: You can configure WebSEAL to return session timeout information to the client by adding a “`<header-name> = %SESSION_EXPIRY%`” entry to the **`[rsp-header-names]`** stanza. See [rsp-header-names] stanza.

Setting a client-specific session cache entry lifetime value

About this task

The **timeout** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, globally sets the maximum lifetime timeout value for all client session information stored in the WebSEAL session cache. You can override this global lifetime value with a per-client lifetime value that is provided as a header in the authentication response from an external authentication interface service. This value is extracted by WebSEAL and stored as an extended attribute in the user's credential.

WebSEAL receives the client-specific timeout information as the value of a header in the authentication response from the external authentication interface. WebSEAL uses the value of that header to set the lifetime timeout of the new session cache entry for that client. This value overrides the value of the **timeout** stanza entry.

The value must represent an absolute time expressed as the number of seconds since 00:00:00 UTC, January 1, 1970. The output of the UNIX **time ()** function, for example, represents the correct format of this absolute time value.

The following steps summarize the necessary configuration for setting a client-specific cache entry lifetime timeout value:

Procedure

1. Configure the custom external authentication interface program to provide, in its authentication response, an HTTP header containing the session cache lifetime timeout value appropriate for that client. The required name of this header is:

   ```
   am_eai_xattr_session_lifetime
   ```

   Note: The name of this particular header is not configurable.
   For example:

   ```
   am_eai_xattr_session_lifetime:1129225478
   ```
2. Configure the custom external authentication interface program to additionally provide an HTTP header that specifies a comma-delimited list of HTTP header names that contain extended attribute values.
   You must configure WebSEAL to look for this header name (see step 4). The default name for this header is **am-eai-xattrs**. (The **am-eai-xattrs** header name is configurable.)
3. Configure the custom external authentication interface program to include the **am_eai_xattr_session_lifetime** header name as a value to the **am-eai-xattrs** header. For example:

   ```
   am-eai-xattrs: am_eai_xattr_session_lifetime
   ```
4. Use the **[eai]** stanza of the WebSEAL configuration file to specify the names of the HTTP headers that contain authentication data returned from the external authentication interface server.
   In the **[eai]** stanza, ensure that WebSEAL looks for the **am-eai-xattrs** header name:

   ```
   [eai]
   eai-xattrs-header = am-eai-xattrs
   ```

   Note: Header names used for the external authentication interface can be customized. Ensure that the custom external authentication interface module is written to use the header names as configured.

Results

If the **am_eai_xattr_inactive_timeout** header is present in a flagged response from the external authentication interface, WebSEAL adds the value to the user's credential as an extended attribute. The entry in the credential for this example appears as follows:

```
am_eai_xattr_session_lifetime:1129225478
```

After the credential is successfully built, WebSEAL creates and entry in the session cache for that client and uses the value of the extended attribute to set the inactivity timeout for that client's session cache entry.

If the **am_eai_xattr_session_lifetime** header is not supplied. WebSEAL uses the default timeout value provided by the **timeout** stanza entry.

Example:

For example, in aFederation Runtime environment, there is an optional element of a Liberty authentication response that is used by an identity provider to dictate to a service provider the duration of a user's session at the service provider.

By modifying the external authentication interface used to authenticate users, a single attribute (the value derived from the identity provider token) can be returned to WebSEAL and used to set the lifetime timeout of session cache entry for that user. The service provider should always request a new single signon interaction with the identity provider once this cache entry lifetime value has expired.

See also:
- "Customized responses for old session cookies" on page 315
- "Cache entry lifetime timeout value" on page 287
- "Setting a client-specific session cache entry inactivity timeout value" on page 276

Cache entry inactivity timeout value

The **inactive-timeout** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, sets the timeout value for user session inactivity.

For example, if a user is inactive for a period of time longer than the inactivity timeout, WebSEAL either deletes the user's session entirely or flags the session as requiring re-authentication. For information on requiring re-authentication for inactive sessions, refer "Reauthentication with external authentication interface" on page 273.

The default login session inactivity timeout (in seconds) is 600:

```
[session]
inactive-timeout = 600
```

The value for a particular session cache (either unauthenticated or authenticated) can be supplied by prefixing the configuration entry with the session cache name (either `unauth` or `auth`). For example:

```
unauth-inactive-timeout = 300
```

WebSEAL does not impose a maximum value for this stanza entry.

A value of "0" disables this inactivity timeout feature (inactivity timeout value is unlimited). The control of cache entries is then governed only by the **timeout** and **max-entries** stanza entries.

When a cache is full, the entries are cleared based on a least-recently-used algorithm. See "Maximum session cache entries value" on page 286.

Note: This stanza entry is ineffective for authentication methods that include authentication data in every request to the WebSEAL server, such as basic authentication (BA), SPNEGO, and some forms of certificate authentication. Those authentication methods automatically reauthenticate the user to the WebSEAL server if the user's session has been deleted due to inactivity or lifetime timeouts. The result is repeated resetting of the inactive and lifetime timeout values.

Preserve inactivity timeout

In some circumstances, you might not want the requests for a particular resource to affect the inactivity timeout for a session. For example, you might want to preserve the inactivity timeout when a server is polled by an Ajax script running in the background of a client browser.

You can create security policies to specify the resources that must not affect the inactivity timeout of the user session. To define this security policy, you must create a protected object policy (POP) with an extended attribute named **preserve-inactivity-time**. You can attach this POP to any object that requires the inactivity timeout to be unaffected by a request. Remember that all children of the object with the POP also inherit the POP conditions.

Use the following commands to create and apply the preserve-inactivity-time POP:

- **pdadmin pop create**
- **pdadmin pop modify**
- **pdadmin pop attach**

The following example creates a POP called **robot** with the **preserve-inactivity-time** extended attribute and attaches it to the **status.html** object:

```
pdadmin> pop create robot
pdadmin> pop modify robot set attribute preserve-inactivity-time true
pdadmin> pop attach /WebSEAL/hostA/junction/status.html robot
```

When this policy is in place, requests made to **status.html** will not impact the inactivity timeout for the user session.

Tip: You can configure WebSEAL to return session timeout information to the client by adding a "`<header-name> = %SESSION_EXPIRY%`" entry to the **`[rsp-header-names]`** stanza. See [rsp-header-names] stanza.

Concurrent session limits

You can configure WebSEAL to limit the number of concurrent requests for a single user session.

Concurrent session threads hard limit

The hard limit is the maximum number of concurrent threads that a single user session can consume. When a user session reaches its thread limit, WebSEAL stops processing any new requests for the user session and returns an error to the client.

Use the **concurrent-session-threads-hard-limit** configuration entry in the **[server]** stanza to configure the session hard limit.

If you do not specify a value for this entry, there is no limit to the number of concurrent threads that a user session can consume.

For example, the following configuration results in a maximum of 10 concurrent threads for a single user session:

```
[server]
concurrent-session-threads-hard-limit = 10
```

Concurrent session threads soft limit

You can use the **concurrent-session-threads-soft-limit** configuration entry in the **[server]** stanza to configure the session soft limit.

The soft limit is maximum number of concurrent threads that a single user session can consume before WebSEAL generates warning messages. WebSEAL continues processing requests for this session until it reaches the configured **concurrent-session-threads-hard-limit**.

For example, if the following entries are set then WebSEAL generates warnings when the number of threads for a user session exceeds the soft limit of five. WebSEAL continues to process requests until the hard limit of 10 is reached. Any further requests cause WebSEAL to return an error to the client.

```
[server]
concurrent-session-threads-soft-limit = 5
concurrent-session-threads-hard-limit = 10
```

Session cache limitation

Limitation:

When you delete a user from the registry, the credentials of that user in the WebSEAL session cache are not removed. If the user has a browser session active at the time the account is deleted, the user can continue to browse, based on the existing session cache entry.

The credentials of the user are not reevaluated, based on the current information in the user registry, until either a new login occurs or the session cache entry expires. The contents of the WebSEAL session cache are cleared when the user logs out of the browser session.

Workaround:

As the administrator, you can force an immediate halt to user activity in a domain by adding an explicit entry to the default WebSEAL ACL policy for the deleted user with the traverse (T) permission removed. You can also terminate the session manually, using either from a command line or using a Security Access Manager administration API function. See "Terminating user sessions" on page 596.

Failover solutions

Use the failover cookie solution to maintain session state in clustered environments.

Topic Index:

Failover cookies and the distributed session cache are alternate solutions to the same challenge of maintaining session state in clustered server environments. See also "Advanced configuration for the distributed session cache" on page 338.

Failover authentication concepts

WebSEAL provides an authentication method that preserves an authenticated session between a client and WebSEAL when the WebSEAL server becomes unavailable in a replicated server (fault-tolerant) environment. The method is called *failover authentication*.

This section contains the following topics:

See also "Failover authentication configuration" on page 298.

The failover environment

The failover cookie is not actually a mechanism for maintaining sessions; it is a mechanism for transparently reauthenticating the user. Failover authentication is most commonly used in a scenario where client requests are directed by a load balancing mechanism to two or more replicated WebSEAL servers.

The replicated servers have identical configuration. They contain replica copies of the WebSEAL protected object space, junction database, and (optionally) dynurl database.

The client is not aware of the replicated front-end server configuration. The load balancing mechanism is the single point of contact for the requested resource. The load balancer connects the client with an available server.

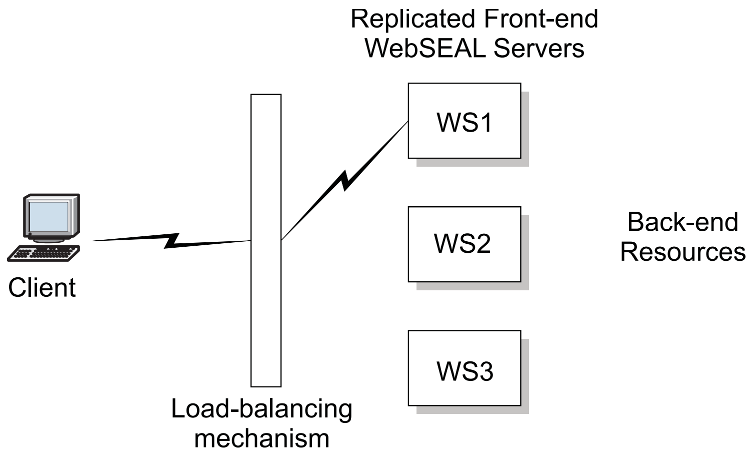


Figure 21. Failover for replicated WebSEAL servers

If the server where the client is connected suddenly becomes unavailable, the load balancer redirects the request to one of the other replicated servers. This action causes the loss of the original session-to-credential mapping. The client is new to this substitute server and is normally forced to login again.

The purpose of failover authentication is to prevent forced login when the WebSEAL server that has the original session with the client suddenly becomes unavailable. Failover authentication enables the client to connect to another WebSEAL server, and create an authentication session containing the same user session data and user credentials.

Failover authentication in a replicated server deployment provides two useful features:

- Performance improvements through load balancing
- Failover of client sessions between WebSEAL servers

References:

- For more information on the replication of WebSEAL servers, see “Replicating front-end WebSEAL servers” on page 587.
- For information on failover solutions in an environment without session affinity (non-sticky), see “Failover for non-sticky failover environments” on page 306.

Failover cookie

WebSEAL supports failover authentication of a user through a *failover cookie*. The failover cookie can be a server-specific cookie or a domain cookie.

The failover cookie contains encrypted client-specific data, such as:

- User name
- Cookie-creation time stamp
- Original authentication method
- Attribute list

By default, the attribute list contains the user's current authentication level. WebSEAL can be configured to add additional extended attributes to the attribute list. See “Failover for non-sticky failover environments” on page 306 for a failover solution that stores the client's session ID as an extended attribute.

The cookie is placed on the browser when the client first connects. If the initial WebSEAL server becomes temporarily unavailable, the cookie is presented to the substitute server.

The replicated WebSEAL servers share a common key that can decrypt the cookie information. When the substitute replica WebSEAL server receives this cookie, it decrypts the cookie, and uses the user name and authentication method to regenerate the client's credential. WebSEAL can also be configured to copy any extended attributes from the cookie to the user credential.

The client can now establish a new session with a replica WebSEAL server without being prompted to log in.

The failover cookie is not a mechanism for maintaining session state. The failover cookie is a mechanism for transparently reauthenticating a user.

You can use the **`failover-cookie-name`** entry in the **`[failover]`** stanza to configure the name of the failover cookie. By default, WebSEAL uses the name `PD-ID` for the failover cookie.

Note: Failover cookies can be used over either HTTP or HTTPS.

Failover authentication process flow

The following steps explain the sequence of events for a failover authentication event:

1. The client (browser) attempts to access a protected resource. The client request goes to a load balancer that controls access to the replicated WebSEAL servers.
2. The load balancer selects a target WebSEAL server and forwards the user request.
3. The client successfully authenticates to WebSEAL using one of the supported authentication methods.
4. WebSEAL creates a failover authentication cookie that contains client authentication information, and sends the cookie to the client browser.
5. The browser sends the cookie through the load balancer to WebSEAL with each subsequent request. The WebSEAL server processes each request.
6. If the load balancer finds that the original WebSEAL server is no longer available, the client request is directed to another replicated WebSEAL server.
7. The replicated WebSEAL server is configured to check for the existence of a failover authentication cookie every time it attempts to authenticate a user.
8. The replicated WebSEAL server uses the information in the cookie to establish a session with the client, without requiring the client to manually log in again. The client's session data and user credential are built, and the request for the protected resource is processed.
9. The change of session from one WebSEAL server to another WebSEAL server is transparent to the client. Because the WebSEAL servers contain identical resources, the client session continues uninterrupted.

Example failover configuration

In this example, a WebSEAL server is configured to support forms authentication and failover authentication.

1. The user authenticates to WebSEAL using forms authentication.
2. The WebSEAL server sends a failover authentication cookie to each client (browser).

   The cookie data specifies that the cookie was created in a forms authentication environment.
3. When the WebSEAL server becomes unavailable, the failover cookie is sent to a second WebSEAL server.
4. The second WebSEAL server receives the failover cookie, and examines it to determine the user's previous authentication method.
5. The second WebSEAL server uses data from the cookie to authenticate the user and build a user credential.

Configuration instructions in this chapter:

- "Adding the authentication strength level" on page 301

Addition of data to a failover cookie

WebSEAL automatically adds specific data from the user session to each failover authentication cookie. WebSEAL can be configured to add additional information from the client data maintained in the credential cache.

By default WebSEAL adds the following data to each cookie:

- **User name**

  This name corresponds to the name used to identify the user in the user registry

  Note: When an authenticated user has used the WebSEAL switch user function to obtain the effective identity of another user, the identity of the other user is not added to the cookie. Only the original authenticated user identity is added to the cookie.
- **Authentication method**

  The authentication method used to authenticate the user to WebSEAL.
- **Cookie creation time**

  The system time when the cookie was created.

WebSEAL also creates an attribute list containing additional data. By default, the attribute list contains one value:

- **Authentication strength level**

  An integer value that corresponds to the WebSEAL authentication strength level (also an integer value) that is assigned on the local WebSEAL server to the current authentication method. Authentication strength, also known as step-up authentication, enables a user to authenticate to a different authentication method without having to logout.

WebSEAL defines additional user data that can be added to the cookie attribute list:

- **Session lifetime timestamp**

  When a user authenticates, WebSEAL tracks the age or lifetime of the user entry in the WebSEAL session cache. The session lifetime timestamp consists of the current time, advanced by the number of seconds configured for the maximum time that a user's session data can remain in the session cache. When the current system time exceeds the timestamp value, WebSEAL invalidates the user's entry in the session cache (including the user credentials).

  WebSEAL can be configured to add the session lifetime timestamp to the cookie. When this timestamp is added to the cookie, the session lifetime timer can be preserved across failover events. WebSEAL administrators can choose whether or not to reset the client's session timer when the client session is established on a replicated server.

  Note that successful use of this feature is dependent on synchronization of clocks between replicated WebSEAL servers. If clock skew becomes great, sessions can expire at unintended times.
- **Session activity timestamp**

  The session activity timestamp is a time value placed as an attribute in the failover cookie when it is created at the server that responds to the initial request.

  This timestamp differs from the session inactivity timeout maintained for the WebSEAL session cache. The system activity timestamp for failover cookies is calculated by combining the `Current system time` with the `Maximum time`. The timestamp is updated at a frequency determined by the `Time interval`.

  Current® system time

  The current time on the WebSEAL server in the HH:MMformat.

  Example value: `13:30`.

Maximum time

The number of seconds that a user's session can remain inactive (`[session], inactive-timeout`).

Example value: `600`

Time interval

The number of seconds between updates to the failover authentication cookie (`[failover], failover-update-cookie`).

Example value: `300`

The timestamp value in the failover cookie in the preceding example is 13:40. If a future request during this session is failed-over from Server 1 to Server2, Server2 accepts the request only if the time on Server2 is less than 13:40. If the time on Server2 is greater than or equal to 13:40, Server2 rejects the request and prompts the user to login to Server2.

The setting for the interval between failover cookie updates affects performance. Administrators must choose a balance between optimal performance and absolute accuracy of the timestamp in the cookie. To keep the timestamp most accurate, failover cookies should be updated every time the user makes a request. However, frequent updating of cookie contents incurs overhead and decreases performance.

Each administrator must choose an interval that best fits the WebSEAL deployment. In some cases, an update of the failover cookie with every user request is appropriate. In other cases, the administrator might choose to never update the timestamp in the failover cookie.

- **Additional extended attributes**

  Administrators can configure WebSEAL to insert a customized set of attributes into a failover cookie. Attributes can be specified individually or in a group. To specify a group of attributes, use wildcard pattern matching in configuration file entries.

  This feature is useful in deployments that also use customized authentication modules to insert special attributes into a user credential. By specifying those attributes in the WebSEAL configuration file, the administrator can ensure that the attributes are available to add to the recreated user credential during failover authentication.

Note: The maximum size of a failover authentication cookie is 4 kilobytes (4096 bytes)

Configuration instructions in this chapter:

- "Adding the authentication strength level" on page 301
- "Addition of session lifetime timestamp" on page 302
- "Adding the session activity timestamp" on page 302
- "Addition of an interval for updating the activity timestamp" on page 303
- "Addition of extended attributes" on page 304

Extraction of data from a failover cookie

When a failover authentication event occurs, a replica WebSEAL server receives a failover authentication cookie and by default extracts the following data from each cookie:

- User name
- Authentication method

- Cookie creation time

WebSEAL first determines if the cookie is valid by subtracting the cookie creation time from the system time, and comparing this value against the WebSEAL configuration file entry for failover cookie lifetime.

If the cookie lifetime has been exceeded, the cookie is not valid, and failover authentication is not attempted. If the cookie lifetime has not been exceeded, WebSEAL uses the user name and authentication method to authenticate the user and build a user credential.

WebSEAL next checks configuration settings to determine if additional cookie data should be extracted and evaluated. Note that the WebSEAL server does not by default extract any other attributes from the failover authentication cookie. Each additional attribute to be extracted must be specified in the WebSEAL configuration file. Wildcard pattern matching can be used to obtain groups of attributes.

WebSEAL can be configured to extract the following defined attributes:

- **Authentication strength level**

  When this value is extracted, WebSEAL uses it to ensure that the user is authenticated with the authentication method necessary to maintain the specified authentication level.

  Note that WebSEAL can obtain authentication strength levels from several different places:

  – Failover cookie

  – Failover authentication library

  – Cross-domain authentication service

  – Entitlements service

  The authentication strength level extracted from the failover cookie takes precedence over levels obtained from the other places.
- **Session lifetime timestamp**

  WebSEAL can use this timestamp to determine if the user's entry in the original server's session cache would have expired. If it would have, WebSEAL discards the cookie and all its potential credential attributes. The session lifetime is not preserved, and the user is prompted to log in.
- **Session inactivity timestamp**

  WebSEAL can use this timestamp to determine if the user's entry in the original server's session cache would have been inactive for too long. If it would have, WebSEAL discards the cookie and all its potential credential attributes. The session lifetime is not preserved, and the user is prompted to log in.

  Note: Successful use of these timestamps requires synchronization of clocks between replicated WebSEAL servers. If clock skew becomes great, sessions will expire or become inactive at unintended times.
- **Additional extended attributes**

  These include user-defined customized attributes, such as those generated by cross-domain authentication services. WebSEAL adds the attributes to the user credential.

Attributes that are not specified in the WebSEAL configuration file will be ignored and not extracted. In addition, administrators can specify that certain attributes *must* be ignored during failover cookie extraction. Although *ignore* is the default behavior, this specification can be useful, for example, to ensure that user attributes are obtained from the user registry instead of from the failover cookie.

Domain-wide failover authentication

WebSEAL supports an optional configuration that enables failover authentication cookies to be marked as available for use during failover authentication to any and all other WebSEAL servers in the DNS domain. This configuration option enables failover authentication cookies to be used in deployments that do not necessarily have a load balancer and replicated WebSEAL servers.

When a client session goes through a failover authentication event to a replicated WebSEAL server, the client continues to access the same set of protected resources. When a client session goes through a failover authentication event to a WebSEAL server that is not replicated, it is possible that a different set of resources will be available to the client. In large deployments, this partitioning of resources within the DNS domain is common. This partitioning can be done for performance reasons and for administrative purposes.

Domain-wide failover authentication can be used to redirect a client to another WebSEAL server at a time when the client's requests have led it to request a resource that is not available through the local WebSEAL server. In this case, the client (browser) is redirected to another WebSEAL server. The receiving WebSEAL server can be configured to look for failover authentication cookies. The WebSEAL server attempts to authenticate the client and recognizes the failover authentication cookie. By using the cookie, the WebSEAL server does not need to prompt the client for login information, but instead can establish a session with the client and construct a valid set of user credentials.

Note: Enabling domain-wide failover authentication introduces additional security risks to the WebSEAL deployment, because the failover cookie can be sent to any server that is in the same DNS domain as the WebSEAL server. If an attacker controls any Web server in the domain or can compromise the DNS server for the domain, they can hijack failover cookies and impersonate users.

Configuration instructions in this chapter:
- “Enabling domain-wide failover cookies” on page 305

Failover authentication configuration

This section contains the following topics:

See also “Failover authentication concepts” on page 292.

Configuring failover authentication

You can configure WebSEAL for failover authentication.

About this task

To configure failover authentication, complete the following tasks:

Note:

For more information about the configuration entries that are related to these tasks, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.

Procedure

1. Stop the WebSEAL server.
2. To enable failover authentication, complete each of the following tasks:
   a. "Protocol for failover cookies"
   b. "Generating a key pair to encrypt and decrypt cookie data" on page 300
   c. "Specifying the failover cookie lifetime" on page 300
   d. "Specifying UTF-8 encoding on cookie strings" on page 301
   e. "Adding the authentication strength level" on page 301
   f. "Reissue of missing failover cookies" on page 301
3. Optionally, you can configure WebSEAL to maintain session state across failover authentication sessions. If this configuration is appropriate for your deployment, complete the following instructions:
   a. "Addition of session lifetime timestamp" on page 302
   b. "Adding the session activity timestamp" on page 302
   c. "Addition of an interval for updating the activity timestamp" on page 303
4. Optionally, you can configure WebSEAL to add extended attributes to the failover cookie:
   - "Addition of extended attributes" on page 304
5. When WebSEAL is configured to add attributes to the failover cookie, you must configure WebSEAL to extract the attributes when reading the cookie:
   - "Attributes for extraction" on page 304
6. Optionally, you can enable failover authentication cookies for use on any WebSEAL server in the domain. If this configuration is appropriate for your deployment, see:
   - "Enabling domain-wide failover cookies" on page 305
7. To maintain compatibility with failover authentication cookies generated by WebSEAL servers from versions before version 8.0, complete the instructions in "Enabling compatibility for failover cookies" on page 305.
8. To maintain compatibility with failover authentication cookies generated by WebSEAL servers from versions before version 6.0, complete the following instructions:
   a. "Specifying UTF-8 encoding on cookie strings" on page 301
   b. "Validation of a lifetime timestamp" on page 306
   c. "Validation of an activity timestamp" on page 306
9. After completing all the instructions applicable to your deployment, restart the WebSEAL server.

Protocol for failover cookies

Failover authentication cookies are disabled by default. To enable failover cookies, edit the WebSEAL configuration file.

In the **[failover]** stanza, specify a value that instructs WebSEAL how to handle requests with failover cookies. The following table shows the valid values.

Table 49. Supported protocols for failover cookies

Stanza Entry	Description
`failover-auth = http`	Failover cookies enabled over HTTP protocol.
`failover-auth = https`	Failover cookies enabled over HTTPS (SSL) protocol.

Table 49. Supported protocols for failover cookies (continued)

Stanza Entry	Description
`failover-auth = both`	Failover cookies enabled over both HTTP and HTTPS (SSL) protocol.

Note: Enabling failover authentication to either HTTP or HTTPS causes cookies to be written to clients connecting over *all* protocols. The value specified in the **failover-auth** stanza entry dictates the protocol over which cookies will be accepted for authentication during a failover authentication event.

Generating a key pair to encrypt and decrypt cookie data

About this task

Use the LMI to generate a key pair that can secure the cookie data. WebSEAL provides this utility. You can generate a symmetric key pair that can encrypt and decrypt the data in a failover cookie.

Note:

- Do not reuse key pairs (used to encrypt and decrypt cookie data) generated for a specific load-balanced environment (configured for failover) in any other load-balanced environments. Always generate unique key pairs for each load-balanced environment configured for failover authentication.
- If you do not configure WebSEAL to encrypt failover authentication cookies, and you have enabled failover authentication, WebSEAL generates an error and refuses to start. Failover authentication cookies must be encrypted.

Procedure

1. Use the LMI to generate the key file, such as `ws.key`. Use the SSO Keys management page to create the key file. To access this page, go to **Secure - Reverse Proxy Settings** > **Global Keys** > **SSO Keys**.
2. Edit the WebSEAL configuration file. In the **`[failover]`** stanza, specify the key file.

   ```
   [failover]
   failover-cookies-keyfile = keyfile_name
   ```
3. Manually copy the key file to each of the remaining replicated servers.
4. On each replicated server, edit the WebSEAL configuration file to supply the correct path name to **`failover-cookies-keyfile`** in the **`[failover]`** stanza.

Specifying the failover cookie lifetime

About this task

Specify the failover cookie lifetime value in the WebSEAL configuration file.

Procedure

Edit the WebSEAL configuration file. Specify the valid lifetime (in minutes) for the failover cookie. The default lifetime is 60 minutes. For example:

```
[failover]
failover-cookie-lifetime = 60
```

NOTE: WebSEAL does not use the **expires** attribute that is contained in a standard HTTP failover cookie. Instead it uses an expiration value that is contained within the token itself. As a result, the lifetime of the cookie expires when the session ends, such as exiting the browser, or when the token expires.

Specifying UTF-8 encoding on cookie strings

About this task

Use UTF-8 when user names or credential attributes in the cookie are not encoded in the same code page as the one that the WebSEAL server is using. By default, WebSEAL servers use UTF-8 encoding. When all WebSEAL servers in the WebSEAL deployment use UTF-8 encoding, leave this value at the default setting of "yes".

Procedure

Edit the WebSEAL configuration file. Specify whether or not WebSEAL should use UTF-8 encoding on strings within the failover cookies.

```
[failover]
use-utf8 = yes
```

The default value is "yes".

Adding the authentication strength level

About this task

To specify authentication strength level in the failover authentication cookie, add the authentication level to the WebSEAL configuration file.

Procedure

Use the **AUTHENTICATON_LEVEL** stanza entry as follows:

```
[failover-add-attributes]
AUTHENTICATION_LEVEL = add
```

The actual value for **AUTHENTICATION_LEVEL** is an integer that WebSEAL tracks internally. You do not need to specify the integer in this stanza.

Reissue of missing failover cookies

In certain proxied environments, it is possible for a client with a valid session to lose a failover cookie. Such a client can continue to maintain a session with the initial WebSEAL system. However, without the failover cookie, the client cannot failover to a new system.

You can use the **reissue-missing-failover-cookie** stanza entry in the **[failover]** stanza of the WebSEAL configuration file to help ensure that a client always has a failover cookie for the duration of the session when failover authentication is enabled. Valid values are “yes” (enable) and “no” (disable).

The failover cookie reissue mechanism is disabled by default. For example:

```
[failover]
reissue-missing-failover-cookie = no
```

When `reissue-missing-failover-cookie = yes`, WebSEAL saves any failover cookie generated for a client in the WebSEAL session cache entry for that client. If previous cookie contents are already stored in the cache entry, they are removed and replaced with the new cookie data.

If the client makes a subsequent request to that WebSEAL server and does not supply the failover cookie in the request, WebSEAL reissues the cached original failover cookie in the response to the client, based on the following conditions:

- The failover cookie reissue mechanism is enabled:

  ```
  reissue-missing-failover-cookie = yes
  ```
- The client has a valid session.
- Failover authentication is enabled for this client type.
- A failover cookie for this client has been stored in the session cache entry for that client.
- No other mechanism has generated a new failover cookie for this request.

Addition of session lifetime timestamp

WebSEAL calculates the session lifetime timestamp by combining the following values:

- Current system time.
- Maximum lifetime in seconds that an entry is allowed to exist in the WebSEAL credential cache.

  This maximum lifetime in seconds is specified in the WebSEAL configuration file **[session]** stanza:

  ```
  [session]
  timeout = 3600
  ```

To add this value to the failover authentication cookie, manually edit the WebSEAL configuration file and add the following entry:

```
[failover-add-attributes]
session-lifetime-timestamp = add
```

When a failover incident occurs, the session lifetime value is used to determine the time remaining for the life of the user's session (the session lifetime value is not reset at failover). If the session lifetime has expired, the user must login.

Note that this attribute cannot be set by wildcard matching. The exact entry **session-lifetime-timestamp** must be entered.

Adding the session activity timestamp

About this task

WebSEAL calculates the session activity timestamp by adding together the following three values:

- Current system time.

  For example: 13:30
- Maximum time (seconds) that a user's session can remain inactive.

  The maximum lifetime for inactive cache entries is set in the **[session]** stanza in the WebSEAL configuration file. For example

  ```
  [session]
  inactive-timeout = 600
  ```

The default value is 600 seconds.

- Time interval (seconds) between updates to the failover authentication cookie.

  This value is set in the **[failover]** stanza in the WebSEAL configuration file. For example:

  ```
  [failover]
  failover-update-cookie = 300
  ```

  The default value is -1 seconds. For more information, see "Addition of an interval for updating the activity timestamp."

The timestamp value in this example is 13:50.

Procedure

Edit the WebSEAL configuration file and add the following entry:

```
[failover-add-attributes]
session-activity-timestamp = add
```

Note: The attribute cannot be set by wildcard matching. The exact entry **session-activity-timestamp** must be entered.

Note: When you set **failover-update-cookie** to a number greater than zero, ensure that you also set `session-activity-timestamp = add`. If you do not set `session-activity-timestamp = add`, WebSEAL decodes the failover cookie on each user access. This repetitive action could adversely affect performance.

Addition of 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 that specifies the interval (in seconds) used to update 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 294.

Note: When you set **failover-update-cookie** to a number greater than zero, ensure that you also set `session-activity-timestamp = add`. If you do not set `session-activity-timestamp = add`, WebSEAL will decode the failover cookie on each user access. This repetitive action could adversely affect performance. See "Adding the session activity timestamp" on page 302.

Addition of extended attributes

WebSEAL can optionally be configured to place a copy of specified extended attributes from a user credential into a failover authentication cookie. No extended attributes are configured by default.

To add extended attributes, add entries to the **[failover-add-attributes]** stanza in the WebSEAL configuration file. The syntax is:

```
[failover-add-attributes]
attribute_pattern = add
```

The *attribute_pattern* can be either a specific attribute name, or a not case sensitive 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 the **[failover-add-attributes]** stanza 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.

Attributes for extraction

WebSEAL can optionally be configured to extract attributes from a failover authentication cookie and place them into a user credential. No attributes are configured for extraction by default.

Attributes to be extracted are declared in the **[failover-restore-attributes]** stanza in the WebSEAL configuration file. The syntax is:

```
[failover-restore-attributes]
attribute_pattern = {preserve|refresh}
```

The value `preserve` tells WebSEAL to extract the attribute and add it to the credential.

The value `refresh` tells WebSEAL to ignore the attribute, and not extract it from the cookie.

The *attribute_pattern* can be either a specific attribute name, or a case-insensitive wildcard expression that matches more than one attribute name. For example, to extract all attributes with the prefix **tagvalue_**, add the following entry:

```
[failover-restore-attributes]
tagvalue_* = preserve
```

Attributes that do not match any patterns specified with the `preserve` value are not extracted from the failover authentication cookie.

The order of the stanza entries is important. Rules that appear earlier in **[failover-restore-attributes]** take priority over those placed later in the stanza.

The following attributes cannot be matched by a wildcard pattern, but must be explicitly defined for extraction:

- Authentication level

```
[failover-restore-attributes]
AUTHENTICATION_LEVEL = preserve
```

- Session lifetime timestamp

```
[failover-restore-attributes]
session-lifetime-timestamp = preserve
```

- Session activity timestamp

```
[failover-restore-attributes]
session-activity-timestamp = preserve
```

Enabling domain-wide failover cookies

About this task

You can allow a failover authentication cookie to be used by any WebSEAL server within the same domain as the WebSEAL server that creates the cookie. This feature is controlled by a stanza entry in the **[failover]** stanza.

By default, domain-wide failover cookie functionality is disabled:

```
[failover]
enable-failover-cookie-for-domain = no
```

Procedure

To enable the domain-wide failover cookie functionality, set **enable-failover-cookie-for-domain** to "yes":

```
[failover]
enable-failover-cookie-for-domain = yes
```

For information on the effects of enabling this stanza entry, see “Domain-wide failover authentication” on page 298.

Enabling compatibility for failover cookies

Security Access Manager, version 8.0, changed the default cipher that WebSEAL uses to encode failover cookies. Failover cookies that are generated with this default cipher are not compatible with failover cookies that are generated by earlier versions of WebSEAL.

About this task

If you integrate failover authentication cookies with WebSEAL servers that include versions of Security Access Manager earlier than version 8.0, you can use the **pre-800-compatible-tokens** stanza entry in the **[server]** stanza to enable compatibility.

Compatibility with previous versions of WebSEAL is not enabled by default:

```
[server]
pre-800-compatible-tokens = no
```

Procedure

To enable compatibility, set **pre-800-compatible-tokens** to yes:

```
[server]
pre-800-compatible-tokens = yes
```

Validation of a lifetime timestamp

WebSEAL servers can optionally be configured to *require* that each failover authentication cookie contain a session lifetime timestamp. The session lifetime timestamp is not required by default. The default configuration file entry is:

```
[failover]
failover-require-lifetime-timestamp-validation = no
```

This stanza entry is used primarily for compatibility with prior versions of WebSEAL. Failover authentication cookies created by WebSEAL servers prior to version 5.1 do not contain this timestamp. For compatibility with failover cookies created by WebSEAL servers prior to version 5.1, set this entry to "no".

- When this value is "no", and the session lifetime timestamp is missing from the failover cookie, the receiving server will view the cookie as *valid*.
- When this value is "yes", and the session lifetime timestamp is missing from the failover cookie, the receiving server will view the cookie as *not valid*.
- When this value is either "no" or "yes", and the session lifetime timestamp is present in the failover cookie, the receiving server evaluates the timestamp. If the timestamp is not valid, the authentication fails. If the timestamp is valid, the authentication process proceeds.

Note: The session lifetime timestamp is configured separately from the session activity timestamp.

Validation of an activity timestamp

WebSEAL servers can optionally be configured to *require* that each failover authentication cookie contain a session activity timestamp. The session activity timestamp is not required by default. The default configuration file entry is:

```
[failover]
failover-require-activity-timestamp-validation = no
```

This stanza entry is used primarily for compatibility with prior versions of WebSEAL.

- When this value is "no", and the session activity timestamp is missing from the failover cookie, the receiving server will view the cookie as *valid*.
- When this value is "yes", and the session activity timestamp is missing from the failover cookie, the receiving server will view the cookie as *not valid*.
- When this value is either "no" or "yes", and the session activity timestamp is present in the failover cookie, the receiving server evaluates the timestamp. If the timestamp is not valid, the authentication fails. If the timestamp is valid, the authentication process proceeds.

Note: The session activity timestamp is configured separately from the session lifetime timestamp.

Failover for non-sticky failover environments

This section contains the following topics:

Non-sticky failover concepts

WebSEAL failover authentication is an appropriate solution for a fault-tolerant environment where client requests are directed by a load balancing mechanism to two or more replicated WebSEAL servers (see "Failover authentication concepts" on page 292 on page 292). Each replica server contains the same content and configuration. If the server where the client is connected suddenly becomes unavailable, the load balancer redirects the requests to one of the other replicated servers.

WebSEAL can handle such failover events by issuing an encrypted failover cookie during the initial authentication of the user. The replica servers share a common encryption key that is used to encrypt and decrypt this failover cookie. This cookie, when presented to another WebSEAL replica, contains sufficient information (along with proof of the replica's own identity) to authenticate the user, build a user credential, and create a new session on the new replica. The user never receives an additional login prompt when a failover occurs.

In a fault-tolerant environment, the load balancer manages the physical distribution of the request load. This discussion uses the terms "sticky" and "non-sticky" to describe the ability of the load balancing mechanism to maintain (or not maintain) a connection between the client and a specific server.

- **Sticky** load distribution maintains a connection between the client and a specific server. This condition is also described as stateful, or observing server affinity.
- **Non-sticky** distribution does not maintain a connection between the client and a specific server. This condition is also described as stateless, or not maintaining server affinity.

In a sticky environment scenario, the load-balancer keeps the client communicating with one replica for the duration of the session. In the rare case of that replica failing, all subsequent requests go to another replica, which is determined by the load balancer.

In non-sticky environments, the load-balancer does not hold the client to one replica. During the course of a user session, multiple requests can be directed to any of the available replica servers at any time.

In either environment, the servers do not share session information and it is WebSEAL's responsibility to maintain session state between the client and the servers. WebSEAL maintains session state through use of a session cookie that is placed on the client browser.

When a client request is switched to another replica server, the failover cookie prevents an additional login. However, the new replica has no knowledge of any session state on any other replica and builds a new session cookie that replaces the previous session cookie. In a non-sticky load balancing environment, the task of frequently issuing new cookies and establishing new sessions significantly degrades both the responsiveness of the user's session and the performance of WebSEAL.

This performance problem can be reduced by ensuring that all replicas reuse the original session cookie issued to the client. WebSEAL can be configured to store the user's original session ID as an extended attribute in the failover cookie.

Each replica involved in handling the client's requests builds a session cache entry using the original session ID. During subsequent returns to a replica, the server ID is verified in a secure manner by comparing the ID in the failover cookie attribute to the ID in the session cookie. If the ID match is successful, the server uses the existing session cache entry does not issue a new session cookie to the browser.

Note: Whenever possible, configure load balancers to maintain session affinity. Session affinity provides improved performance, improved user experience, and makes WebSEAL configuration simpler.

Configuring the non-sticky failover solution

About this task

The following configuration steps enable WebSEAL to reuse a client's original session ID to improve failover authentication response and performance in a non-sticky load-balancing environment. WebSEAL reuses the original session ID by storing the ID as an extended attribute to the failover cookie.

To enable the functionality of including the user's original session ID in the failover cookie, set the **failover-include-session-id** stanza entry in the **[failover]** stanza of the WebSEAL configuration file to "yes":

```
[failover]
failover-include-session-id = yes
```

When you enable the non-sticky failover solution (`failover-include-session-id = yes`), you must configure the following four stanza entries correctly. WebSEAL reports a startup error and fails to start if any of these settings are incorrect:

Procedure

1. The non-sticky failover solution requires the use of the WebSEAL session cookie, rather than the SSL session ID, to maintain session state over HTTPS. Verify that the **ssl-id-sessions** stanza entry in the **[session]** stanza of the WebSEAL configuration file is set to "no" (default):
   ```
   [session]
   ssl-id-sessions = no
   ```
2. To encode the user's original session ID as an extended attribute in the failover cookie, set the **tagvalue_failover_amweb_session_id** stanza entry in the **[failover-add-attributes]** stanza of the WebSEAL configuration file to "add":
   ```
   [failover-add-attributes]
   tagvalue_failover_amweb_session_id = add
   ```
3. When the user session is switched to another replica for the first time, WebSEAL (on that replica) must build a credential and session cache entry for the user, using the information contained in the failover cookie. To ensure that the session ID (encoded in the failover cookie) is added to the user credential, set the **tagvalue_failover_amweb_session_id** stanza entry in the **[failover-restore-attributes]** stanza of the WebSEAL configuration file to "preserve":
   ```
   [failover-restore-attributes]
   tagvalue_failover_amweb_session_id = preserve
   ```
4. The credential refresh feature allows you to update the contents of a user credential on demand by issuing a **pdadmin** command (see "Credential refresh" on page 253). To preserve the session ID attribute used in the non-sticky failover performance solution during a credential refresh, you must set the **tagvalue_failover_amweb_session_id** stanza entry in the **[credential-refresh-attributes]** stanza of the WebSEAL configuration file to "preserve":
   ```
   [credential-refresh-attributes]
   tagvalue_failover_amweb_session_id = preserve
   ```

Use of failover cookies with existing WebSEAL features

The following information discusses the impact of the non-sticky failover solution to other WebSEAL features.

- **Switch-user**

  Failover authentication is not supported for the switch user feature. Therefore, the non-sticky failover solution is also not supported.
- **Authentication methods**

  The non-sticky failover solution does not affect other supported WebSEAL authentication methods.
- **Reauthentication**

  The non-sticky failover solution does not affect reauthentication because reauthentication does not change the user's session ID.
- **Authentication strength (step-up)**

  The non-sticky failover solution does not affect authentication strength policy (step-up) because authentication strength does not change the user's session ID.
- **Credential refresh**

  The **tagvalue_failover_amweb_session_id** stanza entry in the **[credential-refresh-attributes]** stanza of the WebSEAL configuration file allows you to preserve the session ID information in a user's credential during a credential refresh operation. However, the credential refresh command cannot be used to control credential refresh across multiple replica servers. If you perform a credential refresh operation in a server cluster environment, you must issue the credential refresh command to each replica member of the replica set.

Change password operation in a failover environment

The password change operation during authentication can be adversely affected in a non-sticky load balancing environment. For example, a user receives the expired password form and completes the password change information required on the form. When sending the completed form, the load balancer connects to a different replica server. Because this new server is not aware of the previous contact with the original server, it prompts the user to log in. The user provides the old password and is again presented with the expired password form.

The **change-password-auth** stanza entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file allows you to prevent additional login requests during change password operations. Setting `change-password-auth = yes` allows the new replica server to use the existing authentication information in the change password request (user name, original password, and new password) to authenticate the user and change the user's password.

To enable this controlled change password operation in a failover environment, set:

```
[acnt-mgt]
change-password-auth = yes
```

For compatibility with versions of WebSEAL prior to version 6.0, the default setting is "no".

Session state in non-clustered environments

This chapter discusses basic concepts and procedures for managing session state between clients and a single WebSEAL server (non-clustered server environment).

For environments where more than one WebSEAL server provides protection to resources, see “Advanced configuration for the distributed session cache” on page 338.

To terminate individual user sessions or all user sessions, see “User session management for back-end servers” on page 593.

Topic Index:

Maintain session state in non-clustered environments

This section contains the following topics:

Control on session state information over SSL

The **ssl-id-sessions** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, allows you to control whether the SSL session ID or another session key data type is used to maintain the login session for clients accessing over HTTPS.

If the stanza entry value is set to "yes", the SSL session ID is used for all authentication methods. For example:

```
[session]
ssl-id-sessions = yes
```

If the stanza entry value is set to "no" (default), session cookies are used for most authentication methods. For example:

```
[session]
ssl-id-sessions = no
```

A configuration setting of "no" for this stanza entry results in the following conditions for clients accessing over HTTPS:

- The SSL session ID is never used to maintain session state.
- The HTTP header is used as session ID data for clients authenticating with HTTP headers.
- The IP address is used as session ID data for clients authenticating with IP addresses.
- Cookies is used to maintain sessions with clients authenticating with all other methods.

See “Valid session key data types” on page 311.

Use of the same session key over different transports

You can configure WebSEAL to use the same session, or not, when a client authenticates over one type of transport (HTTP, for example), establishes a session, and then connects over another type of transport (HTTPS, for example).

The **use-same-session** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file, provides the following two choices:

- **No**

  When a client authenticates over one transport, establishes a session, and then connects over another transport, the client must authenticate again. A separate session is created using a second session key. The two sessions are maintained independently in the WebSEAL session cache. The appropriate session key used for future connections is determined by the transport that the client uses.

  ```
  [session]
  use-same-session = no
  ```

- **Yes**

  When a client authenticates over one transport, establishes a session, and then connects over another transport, the client uses the same session and corresponding session key that was created for the first transport. The client is not required to authenticate a second time.

  ```
  [session]
  use-same-session = yes
  ```

A "yes" configuration setting for this stanza entry results in the following conditions:

- The HTTP header is used to maintain sessions for clients accessing with HTTP headers over all transport types.
- The IP address is used to maintain sessions for clients accessing with IP addresses.
- Session cookies are used to maintain sessions for all other authentication methods.
- The **ssl-id-sessions** configuration is ignored; the resulting behavior is the same as if **ssl-id-sessions** were set to "no".

  This logic is important because HTTP clients do not have an SSL session ID available as session data.
- Because the cookies are available to both HTTP and HTTPS clients, they are not flagged with the "secure" cookie attribute.

See "Valid session key data types."

Valid session key data types

You can configure the session key data type that WebSEAL uses for each authentication method.

The session key data type used with an authentication method is determined by specific combinations of the following configuration items:

- Setting for the **ssl-id-sessions** stanza entry.
- Setting for the **use-same-session** stanza entry.
- Header names defined in the **[session-http-headers]** stanza.

Table legend:

- **C** (session cookie); **H** (HTTP header); **IP** (IP address); **SSL** (SSL session ID)
- **S-I-S** (ssl-id-sessions); **U-S-S** (use-same-session); **S-H-H** ([session-http-headers])
- If a stanza entry appears in the table header, the value of the entry is "yes". Otherwise, the value is "no".

Table 50. Valid Session Key Data Types for HTTPS Clients

	-	**S-I-S**	**-**	**-**	**S-I-S**	**S-I-S**	**-**	**S-I-S**
Authn	**-**	**-**	**S-H-H**	**-**	**S-H-H**	**-**	**S-H-H**	**S-H-H**
Method	**-**	**-**	**-**	**U-S-S**	**-**	**U-S-S**	**U-S-S**	**U-S-S**
BA	C	SSL	H	C	SSL	C	H	H
CDSSO	C	SSL	H	C	SSL	C	H	H
Certificate	C	SSL	H	C	SSL	C	H	H
External authn interface	C	SSL	H	C	SSL	C	H	H
Failover cookie	C	SSL	H	C	SSL	C	H	H
LTPA cookie	C	SSL	H	C	SSL	C	H	H
Forms	C	SSL	H	C	SSL	C	H	H

Table 51. Valid Session Key Data Types for HTTP Clients

	-	**-**	**-**	**-**
	-	**S-H-H**	**-**	**S-H-H**
Authn Method	**-**	**-**	**U-S-S**	**U-S-S**
BA	C	H	C	H
CDSSO	C	H	C	H
Certificate	C	H	C	H
External authentication interface	C	H	C	H
Failover cookie	C	H	C	H
LTPA cookie	C	H	C	H
Forms	C	H	C	H

Effective session timeout value

When **ssl-id-sessions** is set to "yes", several different values can determine the actual timeout for the session.

The session cache entry lifetime timeout is set in the **timeout** entry in the **[session]** stanza.

The session inactivity timeout is set by the **inactive-timeout** entry in the same stanza.

SSL timeouts are set in the **[ssl]** stanza, where both **ssl-v2–timeout** and **ssl-v3–timeout** stanza entries are declared.

When `ssl-id-sessions = yes`, the actual effective session timeout is set to the *lowest* of the values set for each of the following timeout settings:

```
[session]
timeout
inactive-timeout
```

```
[ssl]
ssl-v2-timeout
ssl-v3-timeout
```

Netscape 4.7x limitation for use-same-session

Problem:

The **use-same-session** feature fails on Netscape Navigator Version 4.7x when requests made to WebSEAL include the port number in the URL, such as: http://webseal:80

Explanation:

When WebSEAL is configured for the default HTTP/HTTPS ports, and the port number is not included in the URL, the request succeeds. Requests fail when WebSEAL is configured on non-default ports and the `use-same-session = yes` configuration option is enabled.

Netscape 4.7x does not consider host names with non-standard port numbers to be in the same domain as those with different port numbers. For example, when you access:

`https://`*`hostname`*`:443`

WebSEAL sets a cookie. When you later access:

`http://`*`hostname`*`:80`

Netscape does not send the cookie because domain:80 is not the same as domain:443.

Workaround:

Upgrade to Netscape Navigator version 6.2, or higher.

Session cookies

This section contains the following topics:

Session cookies concepts

One method of maintaining session state between a client and a server is to use a cookie to hold this session information. The server packages the session key 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 key) 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.

The session cookie is a server-specific cookie that cannot be passed to any machine other than the one that generated the cookie. The session cookie allows the browser to re-identify itself to the single, unique server to which the client had previously authenticated. When using session cookies, WebSEAL does not need to prompt the client for another login.

The session key stored in the session cookie contains only a random number identifier (“key”) that is used to index the server's session cache. There is no other information exposed in the session cookie.

Conditions for using session cookies

The following basic conditions apply to session cookies:

- The session cookie contains session information only; it does not contain identity information.
- The session cookie is located only in the browser memory (it is not written to the browser cookie jar on the disk).
- The session cookie has a limited lifetime.
- The session cookie is a server-specific cookie; the browser can send this cookie in a request only to the same host where the cookie was created.
- Client browsers can be configured to either accept or reject cookies. If a client browser rejects a session cookie and then successfully logs in, WebSEAL must, for each additional request by the client, establish a new session by reauthenticating the user. With basic authentication (BA) however, WebSEAL uses BA header information to reauthenticate the user and the user never experiences a prompt to re-login. However, the overhead of reauthentication and session creation can reduce server performance.

Customization of the session cookie name

You can customize the names of the WebSEAL session cookies. The WebSEAL configuration file provides different default names for session cookies used over TCP and SSL connections:

```
[session]
tcp-session-cookie-name = PD-H-SESSION-ID
ssl-session-cookie-name = PD-S-SESSION-ID
```

Conditions for modifying the default cookie names:

- Cookie names must be alphanumeric.
- Cookie names must be unique.
- To use the same cookie for both TCP and SSL communication, configure `use-same-session=yes`.
- These stanza entries affect both host and domain type cookies.

Sending session cookies with each request

About this task

When you use cookies to maintain session state, the cookie is sent to the browser only once, following a successful login. However, some browsers enforce a limit on the number of in-memory cookies they can store concurrently.

In some environments, applications can place a large number of in-memory cookies per domain on client systems. In this case, any configured WebSEAL session cookie, failover cookie or LTPA cookie can be easily replaced by another cookie.

When you configure WebSEAL to use session cookies, you can additionally set the **resend-webseal-cookies** stanza entry, located in the **[session]** stanza of the WebSEAL configuration file. This stanza entry instructs WebSEAL to re-send the session cookie to the browser for all responses to requests that originally contained a session cookie. This action helps to ensure that the session cookie remains in the browser memory.

The **resend-webseal-cookies** stanza entry has a default setting of "no":

```
[session]
resend-webseal-cookies = no
```

Procedure

Enable WebSEAL to examine each request for a session cookie and include the cookie in the corresponding response by configuring the stanza entry to "yes".

```
[session]
resend-webseal-cookies = yes
```

Note: The **resend-webseal-cookies** stanza entry produces the same results for failover cookies, e-community cookies and LTPA cookies.

Customized responses for old session cookies

This section contains the following topics:

Session removal and old session cookie concepts

When a user uses the **/pkmslogout** command to log out of a session, the entry for that user in the WebSEAL session cache is automatically removed.

If the session cookie for that session remains on the browser of the user, it becomes an old, or stale cookie. A stale cookie no longer maps to an existing entry in the WebSEAL session cache. When the user makes a subsequent request for a protected object, WebSEAL requires authentication and returns a login form. The response to the new request under these conditions must be *expected* by the user. If the user session was removed from the WebSEAL session cache for unknown reasons, the original session cookie remaining on the browser of the user becomes a stale cookie. The stale cookie does not map to an existing entry in the WebSEAL session cache. Session timeout, session displacement, or session termination are some of the reasons which might cause the session removal from WebSEAL, and might be unknown to the user.

When the user requests for a protected object, WebSEAL requires authentication, and returns a login form. This response to the new request under these conditions might be *unexpected* to the user.

You can customize the login response to contain additional information that helps to explain the reason for an unexpected login prompt. Follow these steps to provide a customized response:

1. Trigger a custom login response whenever WebSEAL receives a stale session cookie that does not map to any existing entry in the session cache.

   See “Triggering a custom login response” on page 316.
2. Configure WebSEAL to attempt to remove the session cookie from browsers during standard logouts by using the **/pkmslogout** command.

   See “Removing cookies from browsers during normal logout” on page 316.

Triggering a custom login response:

The standard WebSEAL login forms contain a macro called `OLDSESSION`. The `OLDSESSION` macro can be blank by default, or have a value of 1. When WebSEAL receives a stale session cookie that does not map to any existing entry in the session cache, the `OLDSESSION` macro value is set to 1.

You can cause the appropriate WebSEAL login form, such as `login.html`, to do the following actions:

- Read the value of the `OLDSESSION` macro.
- Trigger a custom response to the user when the value of the macro equals 1.

The custom response informs the user that it is possible that their session was terminated because of inactivity.

Removing cookies from browsers during normal logout:

By default, WebSEAL does not remove cookies from client systems during session termination, regardless of the cause of the termination. The termination might either be initiated by a client logout or by the server. This means that the value of the `OLDSESSION` macro is always set to 1 when a user makes a subsequent request after the session termination. The scenario makes it impossible to trigger a custom login response.

Procedure

Configure WebSEAL to attempt to remove cookies from client systems when the user logs out by sending a cookie in the corresponding response.
Typically, the only time WebSEAL receives stale cookies is when the terminated sessions were not initiated by the user. The scenario requires a custom login response.

Important: While typically successful, there are times when configuring WebSEAL to remove a cookie might not remove cookies from client systems. For example, following a request for **`/pkmslogout`** and a successful session termination at WebSEAL, a network issue might interfere with the logout response transmission. In the scenario, the stale session cookie is left on the client system.

For this reason, security-sensitive decisions must not be based on the presence of the cookie, or the `OLDSESSION` macro value.

Note: An intentional logout does not leave a user with a stale cookie because it is normal browser operation to remove session cookies when a browser closes. A user logs out intentionally by closing the browser application, and the `OLDSESSION` macro is not set during the next request by that user.

However, the user session cache entry remains on the WebSEAL server, or Session Management Server. It continues to count against the `max-concurrent-web-sessions` policy setting until the cache entry expires because of lifetime or inactivity timeout.

Enabling customized responses for old session cookies

You can configure WebSEAL to enable customized responses for old session cookies.

Procedure

1. Configure WebSEAL to remove session cookies from the browsers of users who logs out in a standard manner.

   The `logout-remove-cookie` stanza entry in the `[session]` WebSEAL configuration file stanza controls the removal of session cookies from the browsers of users who does a standard log out.

   The user enters the **/pkmslogout** command in the command line to log out in a standard manner.

   A value of `yes` sets WebSEAL to attempt to remove the cookies from the browsers of users who logs out in a standard manner. For example:`[session] logout-remove-cookie = yes`.
2. Customize the appropriate WebSEAL login form, such as `login.html` to do the following actions:
   - To read the value of the `OLDSESSION` macro.
   - To generate a custom response to the user when the macro value is set to 1.

   You can use any of the following tools to check the `OLDSESSION` macro in the login form:
   - Javascript
   - HTML Meta tags
   - Local response redirection

     See "Local response redirection" on page 79 for details.

Results

Compatibility with WebSEAL versions before version 6.0:

The `logout-remove-cookie = no` default setting sets WebSEAL not to remove cookies from the browsers of users who logs out in a standard manner. For example:

```
[session]
logout-remove-cookie = no
```

The default `no` value exists for compatibility with WebSEAL versions before version 6.0.

Maintain session state with HTTP headers

This section contains the following topics:

HTTP header session key concepts

WebSEAL provides support for maintaining session state using HTTP headers as session keys, independent of the authentication method used.

For example, to allow simultaneous mobile device and internet user support, a Federation Runtime environment requires that WebSEAL use a pre-supplied HTTP header to maintain session state for wireless device clients.

In this scenario, mobile device users connect to a WebSEAL-protected intranet through an authenticated multiplexing proxy agent (MPA) gateway. The WAP gateway serves as a Liberty-enabled proxy (LEP). An LEP is a networking standard created by the Liberty Alliance Project (LAP).

Session state with clients is maintained and managed through Mobile Station Integrated Services Digital Network (MSISDN) HTTP headers. HTTP headers used as session keys are only accepted by WebSEAL when requests are proxied through an authenticated multiplexing proxy agent (MPA).

Configuring HTTP headers to maintain session state

About this task

To configure HTTP headers to maintain session state, specify the header names in the **[session-http-headers]** stanza of the WebSEAL configuration file.

Each header is listed on a per-transport basis. The same header can be listed for both transports. Valid transports include "http" and "https". Use the following syntax:

```
[session-http-headers]
header-name = http|https
```

For example:

```
[session-http-headers]
entrust-client = http
entrust-client = https
```

Conditions for HTTP header session key configuration:

- To allow HTTP headers to be used for maintaining session state, you must set:

  ```
  [session]
  ssl-id-sessions = no
  ```

- If `ssl-id-sessions = yes`, the **[session-http-headers]** stanza is ignored. An exception occurs if MPA support is enabled:

  ```
  [mpa]
  mpa = yes
  ```

- WebSEAL must be configured to accept only HTTP headers in requests proxied through an authenticated multiplexing proxy agent (MPA). See “Setup for requiring requests from an MPA” on page 319.
- List all headers that are to be used for maintaining sessions.
- Limit the header list to no more than 20 entries per transport.
- Do not include the colon (:) character in the header names.
- HTTP headers can be enabled and disabled on a per-transport basis.

Process flow for establishing session state with HTTP headers:

- Session cookies always take precedence over HTTP headers for maintaining session state.

  Upon receiving a request, WebSEAL first looks for a session cookie before continuing to look for configured HTTP headers.

  If an incoming request contains a WebSEAL session cookie, WebSEAL does not look for any configured HTTP headers.
- If a request (containing no session cookie) has an HTTP header matching an entry in the **[session-http-headers]** stanza, that HTTP header is used to maintain session state for that client.
- More than one header can be entered into the **[session-http-headers]** stanza. WebSEAL stops searching requests when the first matching HTTP header is found, regardless of whether or not the header is a key to an existing cache entry.

For example, two headers are configured in the order header A, then header B. A session is established using header B. Header A is for some reason added to a later request from the same client. WebSEAL searches the **[session-http-headers]** stanza and finds a match with header A. Because the existing entry for that client in the session cache is based on header B, WebSEAL does not find an existing session cache entry and prompts the user to authenticate.

- If no entries exist in the **[session-http-headers]** stanza, WebSEAL uses session cookies to maintain session state.
- If `ssl-id-sessions = no` and none of the configured HTTP headers are found in an incoming request, WebSEAL uses session cookies to maintain sessions.

Setup for requiring requests from an MPA

The use of HTTP headers to maintain session state involves the risk that the session keys can be stolen and user sessions spoofed. In an environment involving clients with mobile devices, MSISDN telephone numbers are used as the values of the session keys. Unlike the large size and random nature of session cookie key values, telephone numbers have a smaller, more predictable, and therefore less secure format.

In a secure WebSEAL environment, HTTP header session keys are only valid when requests are proxied through an authenticated multiplexing proxy agent (MPA). The **require-mpa** stanza entry in the **[session]** stanza of the WebSEAL configuration file allows you to control this requirement.

A "yes" setting instructs WebSEAL to only accept HTTP headers from requests that are proxied through an authenticated multiplexing proxy agent (MPA). WebSEAL must authenticate the gateway itself before accepting proxied client connections. For example:

```
[session]
require-mpa = yes
```

A "no" setting allows WebSEAL to accept HTTP headers under any condition. For example:

```
[session]
require-mpa = no
```

A WebSEAL implementation with an MPA must adhere to the following conditions:

- To avoid conflicts, the MPA cannot use the same session key type as a client accessing WebSEAL through the MPA.

  For example, if the MPA maintains sessions using session cookies, a client session must be maintained sessions by a different mechanism.
- To avoid conflicts, the MPA cannot use the same authentication method as a client accessing WebSEAL through the MPA.

  For example, if the MPA uses forms authentication, the client must authenticate using some other mechanism, such as the external authentication interface. In a typical scenario, the MPA uses basic authentication or certificate authentication, and the client uses the external authentication interface.

Note: If WebSEAL is upgraded to 6.1 or later from a previous release that supports HTTP header authentication, the default value for **require-mpa** changes from *no* to *yes*. If HTTP header authentication is being used, the upgrade causes authentication to fail until **require-mpa** is set to *no*.

Share sessions with Microsoft Office applications

You can configure WebSEAL to instruct browsers to share session information with Microsoft Office applications. Sharing session information avoids the need for the Microsoft Office applications to re-authenticate the user.

Overview of session sharing with Microsoft Office applications

You can configure WebSEAL to use cookies to maintain client sessions. For security reasons, WebSEAL uses non-persistent cookies. Since Internet Explorer and Microsoft Office are only capable of sharing persistent cookies, the Microsoft Office applications cannot share the WebSEAL user session by default.

You can configure WebSEAL to create a short-lived persistent session cookie. This cookie stores an index into a temporary session cache that WebSEAL uses to locate the corresponding session in the standard session cache.

You can configure this temporary cache entry for a single use or multiple uses by WebSEAL. The cache is not shared between WebSEAL instances. Microsoft Office applications can use the persistent cookie to locate the corresponding user session from Internet Explorer.

A request for the `/pkmstempsession` URI triggers the creation of this temporary session cookie. You can include a target redirect URL in the `/pkmstempsession` request. WebSEAL redirects the client to this URL when the processing of the `/pkmstempsession` request is complete. If no redirect URL is provided, WebSEAL returns a default results page to the client.

```
http://<server>/pkmstempsession?url=<requested_resource>
```

where:

<server>

The fully qualified host name of the WebSEAL server.

<requested_resource>

The location of the target resource.

For example, a Microsoft Office document: `/server/test.doc`.

Note: The request resource URL can optionally contain query string arguments. These arguments remain unchanged in the resulting WebSEAL redirect request.

The short-lived persistent cookie is created by sending a request to the `/pkmstempsession` URI. This cookie creation must occur before the client switches context from WebSEAL to Microsoft Office. See “Configure shared sessions with Microsoft Office applications” on page 323 for configuration details of two common use cases in a Microsoft Office environment.

Configure the temporary session cache

You can configure the temporary session cache that WebSEAL uses for its session sharing functionality. In particular, you can control the lifetime of cache entries, specify the name of the temporary session cookie, and configure the default results page.

The following WebSEAL configuration settings are related to the temporary session cache.

- “Configuring the lifetime of entries in the temporary session cache” on page 321

- "Controlling whether the temporary session cache cookies are single use"
- "Configuring the name of the temporary session cookie" on page 322
- "Configuring the temporary cache response page" on page 322

Configuring the lifetime of entries in the temporary session cache:

You can configure the maximum lifetime of entries in the temporary session cache that WebSEAL uses for session sharing with Microsoft Office applications.

About this task

The temporary session cache stores a short-lived session. WebSEAL uses this cache to create an intermediate session that maps between the short-lived session and the standard session. The index into the temporary session cache is returned to the client as a persistent cookie.

Use the **temp-session-max-lifetime** entry in the **[session]** stanza of the WebSEAL configuration file to set the maximum lifetime (in seconds) of entries in the temporary session cache.

Procedure

Set the **temp-session-max-lifetime** value to 0 to disable the use of the temporary session cache.

Example

The following example configures a maximum lifetime duration of 10 seconds:

```
[session]
temp-session-max-lifetime = 10
```

Controlling whether the temporary session cache cookies are single use:

You can control whether the client can access an entry in the temporary session cache more than once. If you are sharing sessions with Microsoft Office applications, configure WebSEAL to accept multiple accesses to the temporary session cookie.

About this task

The temporary session cache stores a short-lived session. WebSEAL uses this cache to create an intermediate session that maps between the short-lived session and the standard session. The index into the temporary session cache is returned to the client as a persistent cookie.

Procedure

1. Locate the **[session]** stanza in the WebSEAL configuration file.
2. Set the **temp-session-one-time-use** entry to one of the following values:

   true Specifies that the client can use the temporary session cookie one time.

   false Specifies that the client can access and use the temporary session entry multiple times until **temp-session-max-lifetime** expires. Use this setting if you want to share sessions with Microsoft Office applications, such as Microsoft SharePoint.

Example

In the following example, WebSEAL creates a single-use cookie in the temporary session cache:

```
[session]
temp-session-one-time-use = true
```

Configuring the name of the temporary session cookie:

You can configure the name of the temporary session cookie that WebSEAL creates to share a session with Microsoft Office applications.

About this task

WebSEAL creates the short-lived session cookie in response to a request for the **/pkmstempsession** management page. This cookie stores an index into the temporary session cache that WebSEAL uses later to locate the corresponding session in the standard session cache.

Procedure

Use the **`temp-session-cookie-name`** entry in the **`[session]`** stanza of the WebSEAL configuration file to specify the name of the temporary session cookie.

Note: This configuration item is valid only when the temporary session cache is enabled. To enable the cache, you must set a non-zero value for the **`temp-session-cookie-name`** entry in the **`[session]`** stanza.

Example

The following example configures a cookie name of `PD-TEMP-SESSION-ID`:

```
[session]
temp-session-cookie-name = PD-TEMP-SESSION-ID
```

Configuring the temporary cache response page:

About this task

Use the **temp-cache-response** entry in the **[acnt-mgt]** stanza of the WebSEAL configuration file to set the default management page. WebSEAL returns this page after processing a `pkmstempsession` request if no redirect URL is supplied in the `pkmstempsession` request.

By default, WebSEAL returns `temp_cache_response.html`.

```
[acnt-mgt]
temp-cache-response = temp_cache_response.html
```

Example

The following example illustrates a typical request:

```
http://<server>/pkmstempsession?url=<requested_resource>
```

If the **url** argument is not included in the request then WebSEAL returns the page specified by the **temp-cache-response** configuration entry.

Configure shared sessions with Microsoft Office applications

You can configure WebSEAL to use the temporary session cache so that both Internet Explorer and Microsoft Office products can reference the same user session.

A request to the `/pkmstempsession` management page creates a temporary session cookie. Both Internet Explorer and Microsoft Office products can access this session cookie.

You can use Ajax requests to automatically request this management page and trigger the creation of a persistent session cookie. Through this process, you can achieve session sharing with applications that are accessed through Internet Explorer.

Microsoft SharePoint 2013 and SharePoint 2016 server:

You can modify the JavaScript on the Microsoft SharePoint 2013 and SharePoint 2016 server so that Internet Explorer and Microsoft Office can share the same session when accessing a SharePoint resource.

About this task

To achieve session sharing with Microsoft SharePoint server, the SharePoint administrator must determine when to create the persistent session cookie for sharing sessions. The goal is to request the `pkmstempsession` management page immediately before the context switch for the requested resource.

SharePoint does not send any notification to WebSEAL before this context switch. However, you can create a custom JavaScript file on the SharePoint server, to automatically send an Ajax request to WebSEAL before accessing the requested resource. This Ajax HTTP request can collect the session cookie for the temporary session cache. You must configure SharePoint to use this custom JavaScript file instead of `core.js`.

Note: Do not directly update the `core.js` file on the SharePoint Server. Updating this file directly is not a supported SharePoint modification. Instead, create a custom JavaScript file and custom master page to override the required functions of `core.js`. Use the custom master page to configure your site collections to use these updated JavaScript functions.

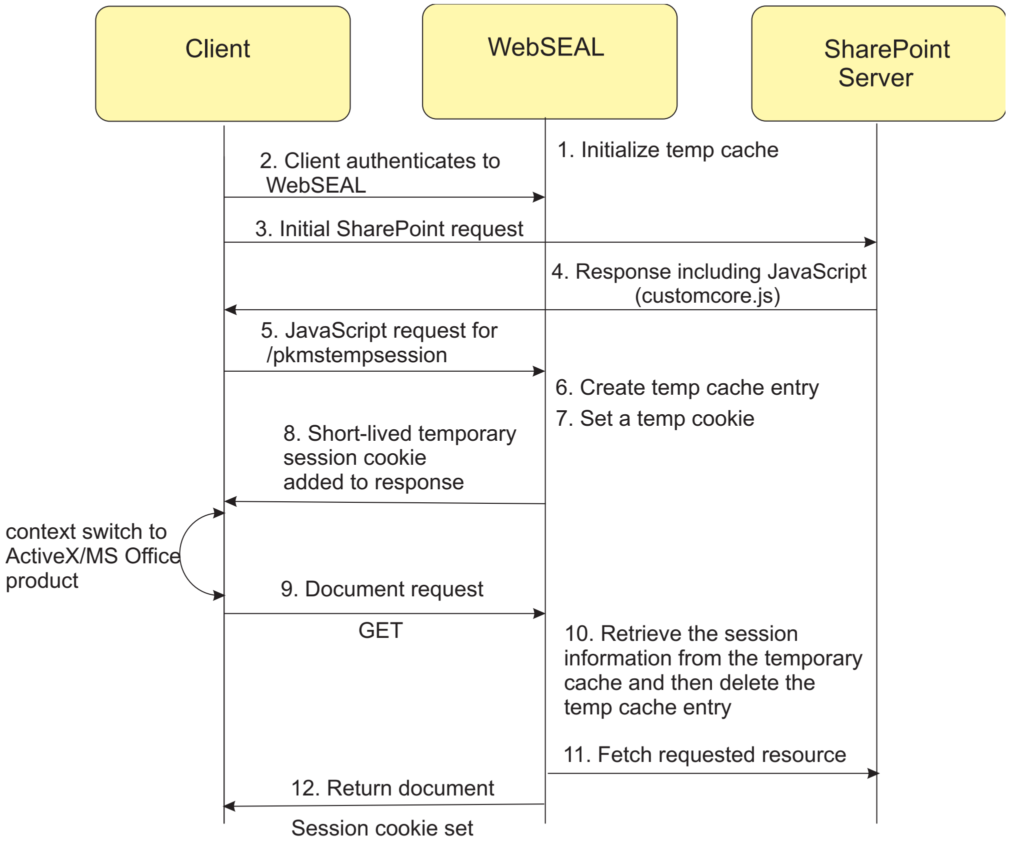

Client
WebSEAL
SharePoint Server
1. Initialize temp cache
2. Client authenticates to WebSEAL
3. Initial SharePoint request
4. Response including JavaScript (customcore.js)
5. JavaScript request for /pkmstempsession
6. Create temp cache entry
7. Set a temp cookie
8. Short-lived temporary session cookie added to response
context switch to ActiveX/MS Office product
9. Document request
GET
10. Retrieve the session information from the temporary cache and then delete the temp cache entry
11. Fetch requested resource
12. Return document
Session cookie set

Figure 22. Sharing WebSEAL sessions with Microsoft SharePoint server

Figure 22 illustrates the temporary session cache sequence of operations.

1. When WebSEAL starts, it initializes the temporary cache.
2. The client authenticates to the WebSEAL server.
3. The client sends a request for a SharePoint resource.
4. SharePoint sends a response back to the client that includes JavaScript (customcore.js).
5. Before sending the SharePoint resource request, the updated JavaScript sends an Ajax HTTP request for `/pkmstempsession`.
6. WebSEAL adds an entry for the session in the temporary cache.
7. WebSEAL sets a short-lived cookie with the session information.
8. WebSEAL adds the short-lived cookie to the response that is returned to the client browser.
9. The client switches context to MS Office/ActiveX and the application uses the GET method to send the SharePoint document request.
10. WebSEAL retrieves the session information from the temporary cache and then deletes the entry so that the cookie cannot be used again.
11. WebSEAL fetches the requested resource from the SharePoint server.
12. WebSEAL returns the document to the client with the session cookie set.

Procedure

1. Copy the `core.js` file. Paste the copy into the folder that contains the `core.js` file and rename it as `customcore.js`.

In a default SharePoint 2013 and SharePoint 2016 installation, you can find the `core.js` file at:

```
C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\
TEMPLATE\LAYOUTS\core.js
```

2. Open the custom JavaScript file (`customcore.js`) for editing.
3. You must modify the JavaScript to make an out of band Ajax HTTP request to the **/pkmstempsession** page. Add the following JavaScript to any link that opens the Office Document by using the ActiveX Controls. In particular, add the new JavaScript at the beginning of the following functions in the `customcore.js` file on the SharePoint Server:
   - `function DispEx(...)`
   - `function createNewDocumentWithProgIDCore(...)`

   Note: Remove all of the other JavaScript functions from `customcore.js` so that it contains only the functions that are being overridden. Deleting the functions that are not needed improves processing efficiency.

   Example JavaScript

```
var cookieRequest = false;
try {
 cookieRequest = new XMLHttpRequest();
}
catch (trymicrosoft) {
 try {
  cookieRequest = new ActiveXObject("Msxml2.XMLHTTP");
 } catch (othermicrosoft) {
  try {
   cookieRequest = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (failed) {
   cookieRequest = false;
  }
 }
}
if (cookieRequest) {
 var url = "/pkmstempsession";
 cookieRequest.open("GET", url, false);
 cookieRequest.send(null);
 if (cookieRequest.status != 200){
  alert("ERROR: Single-Signon Cookie Request Failed!,Application may not load Document");
 }
}
```

4. If you are using the `default.master` page as the master page in your SharePoint site, make a copy of the `default.master` page. Rename the copy as `custom.master`.

   In a default SharePoint 2013 and SharePoint 2016 installation, you can find the `default.master` file at:

```
C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\
TEMPLATE\GLOBAL\default.master
```

5. In the `custom.master` page, add a line to use the `customcore.js` file. Insert this new entry immediately after the existing entry that references `core.js` as follows:

```
<SharePoint:ScriptLink language="javascript" name="core.js" Defer="true"
   runat="server"/>
<SharePoint:ScriptLink language="javascript" name="customcore.js" Defer="true"
   runat="server"/>
```

   Note: You must keep a reference to the original `core.js` file in this custom master page. You must set the `Defer="true"` property to override the original `core.js` file.

6. Save the custom.master page and upload it to the master pages gallery of the site.
7. Set the custom.master page as the default master page for the site:
   a. Log in to your site in SharePoint.
   b. Go to **Site Action** > **Site Settings** > **Modify All Site Settings** > **Look and Feel** > **Master Page**.
   c. Select **Specify a master page to be used by this site and all sites that inherit from it**.
   d. Select **custom.master** from the drop-down list.

Results

The **Defer="true"** setting specifies the load order of the SharePoint functions. Including two entries with **Defer="true"** effectively merges the JavaScript files together. The entry for the customcore.js file is included in the master page source after the entry for the core.js file because SharePoint runs the deferred scripts in order. SharePoint runs core.js first, followed by the overriding customcore.js script.

Note: Be aware of the following considerations if you use custom JavaScript functions to override core.js:

- To minimize the amount of script that is transmitted during processing, keep only the methods that you want to customize in the customcore.js file.
- If you upgrade the SharePoint product or install a patch, review your custom scripts against the updated core.js to ensure that the override scripts are still applicable.

Chapter 5. Distributed session cache

Distributed session cache overview

The following concepts relate to the distributed session cache, which maintains session state in clustered server environments.

Failover cookies and the distributed session cache are alternate solutions to the same challenge of maintaining session state in clustered server environments. See also "Failover solutions" on page 292.

The distributed session cache replaces the session management server solution that was available in earlier releases of IBM Security Access Manager.

The failover environment

The distributed session cache is most commonly used in a scenario where client requests are directed by a load balancing mechanism to two or more replicated WebSEAL servers.

The replicated servers are identical. They contain replica copies of the WebSEAL protected object space, junction database, and (optionally) dynurl database.

The client is not aware of the replicated front-end server configuration. The load balancing mechanism is the single point of contact for the requested resource. The load balancer connects the client with an available server.

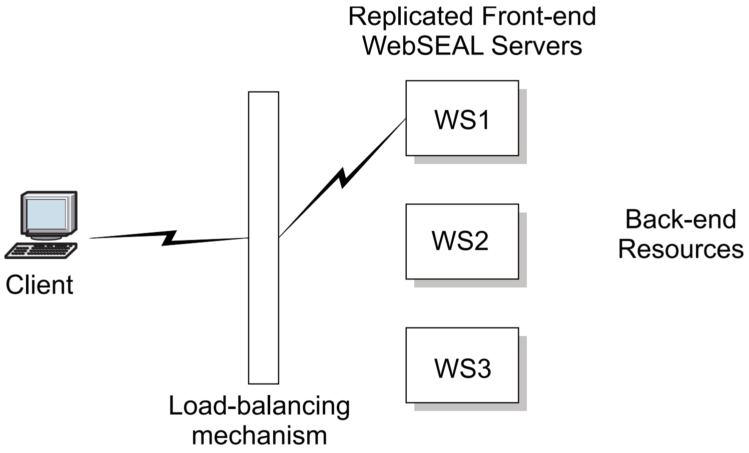


Figure 23. Failover for replicated WebSEAL servers

If the server where the client is connected suddenly becomes unavailable, the load balancer redirects the request to one of the other replicated servers. This action causes the loss of the original session-to-credential mapping. The client is new to this substitute server and is normally forced to login again.

The purpose of the distributed session cache is to prevent forced login when the WebSEAL server that has the original session with the client suddenly becomes unavailable. The distributed session cache enables the client to connect to another WebSEAL server, and create an authentication session containing the same user session data and user credentials.

Note: Whenever possible, configure load balancers to maintain session affinity. Session affinity provides improved performance, improved user experience, and makes WebSEAL configuration simpler.

The distributed session cache

The distributed session cache is an independent service that acts as a centralized session repository for a clustered WebSEAL server environment. Servers in the cluster can use the distributed session cache to provide failover for user sessions.

The primary master of the cluster acts as the distributed session cache server. You can use a stand-alone cluster with only a single node for the distributed session cache. However, you can specify up to three supplementary masters to ensure high availability of the distributed session cache. These supplementary masters maintain slave copies of the distributed session cache for failover purposes.

For more information about how to configure a cluster of IBM Security Access Manager appliances, see the "Cluster Support" topics in the Administering Web Reverse Proxy topics in the Knowledge Center.

The distributed session cache server can manage sessions from any of the following sources:

- Appliance-based WebSEAL instances that are members of the same cluster as the distributed session cache server.
- Appliance-based WebSEAL instances that are on appliances that are not in the same cluster as the distributed session cache server. However, it is best to include the appliance in the same cluster as the distributed session cache server where possible.
- Software-based WebSEAL, version 7.0, instances.

The distributed session cache provides the following benefits:

- Manages sessions across clustered Web security servers.
- Resolves session inactivity and session lifetime timeout consistency issues in a replicated Web security server environment.
- Provides secure failover and single sign-on among replicated Web security servers.
- Provides controls over the maximum number of allowed concurrent sessions per user.
- Provides single sign-on capabilities and single sign-off among other websites in the same DNS domain.
- Provides performance and high availability protection to the server environment in the event of hardware or software failure.
- Allows administrators to view and modify sessions on the WebSEAL server.

Failover for the distributed session cache

The distributed session cache server is configured in a clustered environment. You can configure up to four master nodes in the cluster to ensure high availability for the distributed session cache.

The primary master of the cluster provides the distributed session cache service to the cluster. If the distributed session cache on the primary master fails, an error is printed in the WebSEAL log.

For example:

```
2013-12-10-12:44:25.525+10:00I----- 0x38CF0B24 webseald ERROR wwa soap
AMWSOAPHandle.cpp 261 0x7f8cac0ac720 --
DPWWA2852E   An error occurred when attempting to communicate
with the SOAP server URL http://10.150.21.1:2028/DSess/services/DSess:
HTTPTransportException:Client failed to open Failed to open connection to server:
--hostname='10.150.21.1' -- service='2028' --
Error Message='Connection refused'  Error Code='111' --  (error code: 12/0xc).
```

If there are no supplementary masters specified, the distributed session cache service is unavailable until the primary master is restored. However, if there is a secondary master configured, the distributed session cache fails over to the secondary master. You can also configure tertiary and quaternary masters to provide extra failover for the distributed session cache. For more information, search for "Failover in a cluster" in the Administering Web Reverse Proxy topics in the Knowledge Center.

Replica sets

A cluster of Security Access Manager servers that use the distributed session cache to share sessions is called a *replica set*.

Servers in the replica set can use the distributed session cache to provide failover of user sessions. A client session created by one member of a replica set can be used unmodified by another.

Certain policies, including maximum concurrent session policy and policies that affect credential change, can apply consistently across a replica set. From the user and administrator points of view, sessions exist as a single entity across a replica set.

Distributed session cache process flow

In a distributed session cache environment, the client browser sends requests to the WebSEAL server cluster, which interacts with the distributed session cache for session management.

The following diagram shows the basic process flow for session management in an environment where WebSEAL is configured to use the distributed session cache. The example contains the following conditions:

- WebSEAL 1 and WebSEAL 2 are configured with replica virtual hosts (vhostA).
- The replica virtual hosts belong to a replica set.

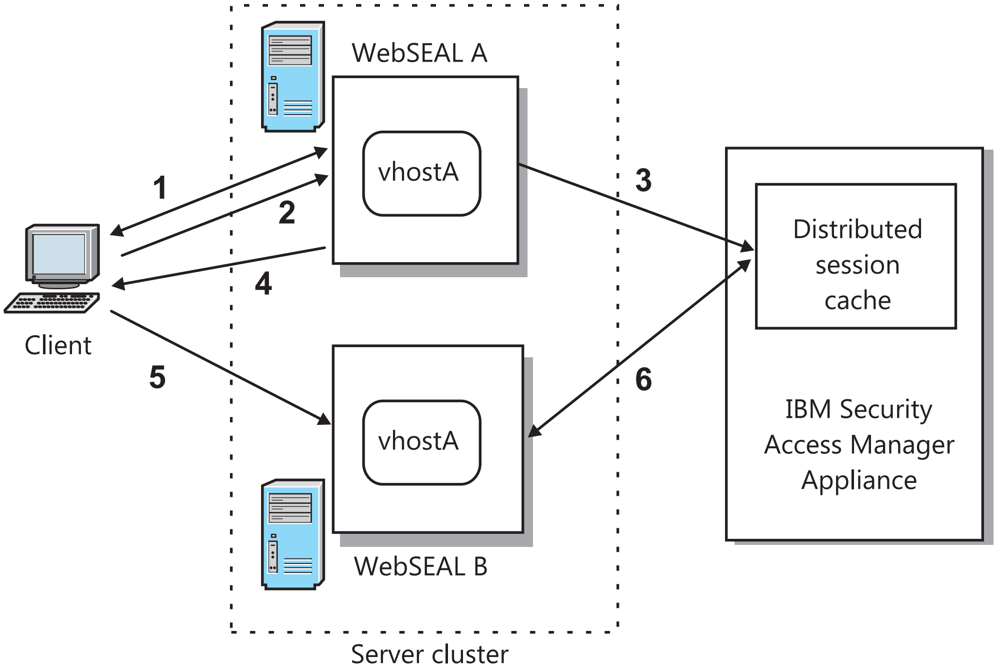

WebSEAL A
vhostA
1
2
3
4
5
6
Client
Distributed session cache
IBM Security Access Manager Appliance
vhostA
WebSEAL B
Server cluster

Figure 24. Distributed session cache process flow

1. A user makes a request for a protected object located in the Web space of **vhostA**. WebSEAL A intercepts the request and creates a local cache entry for the user. WebSEAL A prompts the user to log in.
2. The user provides authentication data to WebSEAL. WebSEAL updates the local session cache entry with the client's credential.

   Maintaining a local session cache improves the performance of that specific WebSEAL server during future requests for resources.
3. WebSEAL A notifies the distributed session cache of the new session and the associated credential information. The distributed session cache maintains this information in its own database.
4. WebSEAL A sends a session cookie to the user's browser.
5. An additional request for a resource on **vhostA** by the same user, using the same session cookie, fails over to another server in the replica set (WebSEAL B).
6. Using the session cookie, WebSEAL B consults the distributed session cache to determine whether the user has already authenticated. The distributed session cache replies with the user's cached credential.

   WebSEAL B uses the credential to trust the user and allows the request for the resource to proceed. The user is not prompted to login again.

Sharing sessions across multiple DNS domains

When operating in a multi-domain environment, you must use a different mechanism to communicate the session identifier to the WebSEAL servers in the different DNS domains.

About this task

The index into the distributed session cache is stored and transmitted in a domain cookie. Clients do not present this cookie to servers that reside outside of the source DNS domain.

Extensions to the external authentication interface (EAI) allow an EAI application to supply WebSEAL with a session identifier instead of authentication data. For more details, see “External authentication interface HTTP header reference” on page 270. This session identifier corresponds to the session index into the distributed session cache. The mechanism by which the EAI application receives the session identifier depends on the implementation of the EAI application. For example, you can configure:

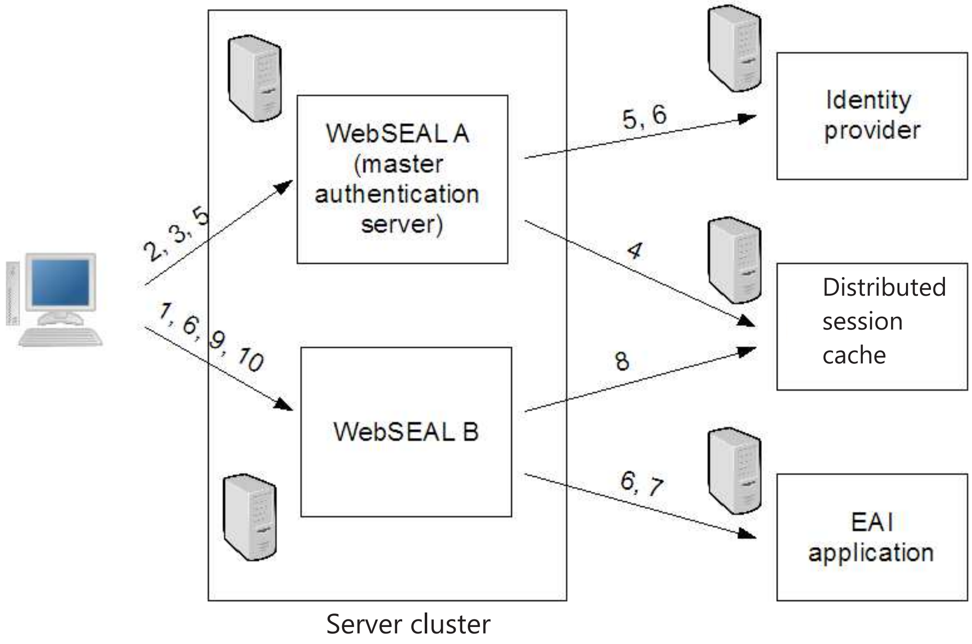
- The Federation Runtime to supply the identifier as a part of a SAML assertion.
- An identity provider to add the session identifier into a SAML assertion. The service provider then retrieves the session identifier from the SAML assertion and passes the information back to WebSEAL.

For this mechanism to work, designate a single WebSEAL server or multiple load-balanced Web servers in the same DNS domain as the primary authentication server. This WebSEAL server establishes the authenticated session. It also hosts the identity provider, which communicates the session identifier back to the EAI application.

To make the session identifier available to the identity provider application, see “User session management for back-end servers” on page 593.

The following diagram shows the basic process flow for session management in a multi-domain environment where WebSEAL is configured to use the distributed session cache. The example is based on the following conditions:

- WebSEAL A is configured as the primary authentication server.
- WebSEAL A & B reside in different DNS domains.
- No session is established in either WebSEAL server.



Procedure

1. A user makes a request for a protected object located in the Web space of WebSEAL B, which:

   - Intercepts the request.
   - Creates a local cache entry for the user.
   - Sends a redirect back to the identity provider application that is behind the primary authentication server. You can either set this redirect as a meta-header within the login form or specify it using the local-response-redirect functionality.
2. The identity provider is a protected application that requires authentication. WebSEAL A:
   - Intercepts the request.
   - Creates a local cache entry for the user that contains the requested URL for the identity provider.
   - Prompts the user to log in.
3. The user provides authentication data to WebSEAL, which performs a local authentication and updates the local session cache entry with the credential of the client.
4. WebSEAL A notifies the distributed session cache of the new session and the associated credential information. The distributed session cache maintains this information in its database.
5. WebSEAL A sends:
   - A session cookie to the browser of the user.
   - A redirect to the cached URL for the identity provider.
6. The identity provider:
   - Extracts the session identifier from the HTTP request headers.
   - Sends a redirect to the EAI application junctioned behind WebSEAL B. The mechanism for passing the session identifier to the EAI application is implementation-specific.
7. The EAI application:
   - Locates the session identifier supplied by the identity provider.
   - Passes the identifier back to WebSEAL B in the appropriate HTTP header.
8. WebSEAL B retrieves the corresponding session from the distributed session cache.
9. WebSEAL B sends:
   - A session cookie to the browser of the user.
   - A redirect to the original cached URL.
10. WebSEAL B:
    - Uses the session cookie set in step 9 to locate the appropriate local session and credential.
    - Allows the request for the resource to proceed.
    - Does not prompt the user to log in again.

Advantages of using the distributed session cache to maintain session state

The distributed session cache solution has many advantages over the failover cookies solution for maintaining session state.

Table 52. Comparison of the distributed session cache solution

Topic	**Failover cookies**	**Distributed session cache**
Security	Slightly less secure than the distributed session cache as identity information is stored in an encrypted cookie.	Provides defense in depth with the distributed session cache located behind the DMZ.
Failover between WebSEAL instances	Higher CPU usage is required for WebSEAL to decrypt the failover cookie. A new session is established, which means that you do not share session semantics such as timeout information with other WebSEAL instances.	The distributed session cache shares sessions rather than using a single sign-on mechanism. Session semantics such as timeout information are shared between the various WebSEAL instances.
Session policy	• No concurrent session policy enforcement. • No central administration of sessions.	• Concurrent session policy enforcement. • Central management of sessions, including session termination by using the **dscadmin** tool. • You can list the number of failed login attempts on an account since a successful login.
Maintenance	Requires periodic renewal of failover cookie keys in line with corporate policy. This process is manual.	The distributed session cache does not require encryption keys.
Cookies	The failover cookie is larger than the distributed session cache session cookie. The failover cookie can be configured to include many attributes, which can significantly increase its size.	The distributed session cache uses a basic session cookie, which is relatively small. The cookies in a distributed session cache environment are typically less than 100 bytes.
Logout	In browser scenarios, you cannot successfully log out with failover cookies turned on.	You can logout from browser session when you use the distributed session cache.

Migrating from a Session Management Server (SMS) environment

The distributed session cache replaces the session management server (SMS) solution that was available in earlier releases of IBM Security Access Manager. You can configure IBM Security Access Manager, version 7.0 servers, which previously used the SMS, to use a distributed session cache server in your environment.

The IBM Security Access Manager, version 8.0, distributed session cache can interoperate with IBM Security Access Manager, version 7.0 servers. You can introduce anIBM Security Access Manager version 8.0 appliance to provide a distributed session cache that manages sessions for WebSEAL version 7.0 and version 8.0 servers.

The following component versions are supported:

Policy Server
IBM Security Access Manager versions 7.0 and 8.0.

WebSEAL
IBM Security Access Manager versions 7.0 and 8.0. The WebSEAL servers can be appliance-based or software-based.

Distributed Session Cache Server
IBM Security Access Manager version 8.0 appliance.

To configure an IBM Security Access Manager, version 7.0, WebSEAL server to use the distributed session cache, complete the steps that are described in "Configuration for WebSEAL instances that are external to the cluster to use the distributed session cache" on page 335.

Note: You can use the local management interface to configure IBM Security Access Manager, version 8.0 appliances in the cluster to use the distributed session cache. For more information, see "Configuring a WebSEAL instance on a cluster member to use the distributed session cache."

Quickstart guide for WebSEAL to use the distributed session cache

The process for configuring WebSEAL to use the distributed session cache depends on whether WebSEAL is running on an appliance in the same cluster as the distributed session cache server.

WebSEAL instance that is internal to the distributed session cache server cluster
If WebSEAL is on an appliance in the same cluster, there is a configuration option in the local management interface (LMI) to configure WebSEAL to use the distributed session cache. For more information, see "Configuring a WebSEAL instance on a cluster member to use the distributed session cache."

WebSEAL instance that is external to the distributed session cache server cluster
For a WebSEAL instance that is not on an appliance in the distributed session cache server cluster, you can manually configure WebSEAL to use the distributed session cache. This manual configuration is required for WebSEAL instances that fall into the following two categories:

- Software-based WebSEAL instances, such as ISAM 7.0 software installations.
- Appliance-based WebSEAL instances that are not in the same cluster as the distributed session cache server.

For more information, see "Configuration for WebSEAL instances that are external to the cluster to use the distributed session cache" on page 335.

"Advanced configuration for the distributed session cache" on page 338 describes the distributed session cache manual configuration options in greater detail.

Configuring a WebSEAL instance on a cluster member to use the distributed session cache

For nodes that are in the distributed session cache server cluster, use the local management interface (LMI) to configure WebSEAL to use the distributed session cache.

About this task

For an appliance-based WebSEAL instance that is internal to the distributed session cache server cluster, the configuration occurs automatically when you enable the distributed session cache in the LMI.

Procedure

1. In the LMI, select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
2. Select the WebSEAL instance that you want to use the distributed session cache.
3. Click **Edit**.
4. Select the **Session** tab.
5. Select **Enable Distributed Session Cache**.
6. Save and deploy the configuration changes.
7. Restart the WebSEAL instance.

Results

WebSEAL is automatically configured to use the distributed session cache. WebSEAL uses SSL to communicate with the distributed session cache.

A message is printed in the WebSEAL log regarding the synchronization between the local session cache and the distributed session cache. No further manual configuration is required.

What to do next

You can use the local management interface to view and manage the servers in the replica set. Go to **Secure Web Settings** > **Manage** > **Distributed Session Cache**.

The **dscadmin** tool is also available for session management. For more information about this tool, search for "dscadmin command" in the Configuring Web Reverse Proxy topics in the Knowledge Center.

By default, all WebSEAL instances are added to the `default` replica set. If you want to group the WebSEAL instances into separate replica sets, change the name of the replica set that is associated with each instance in the WebSEAL configuration file. For more information about this configuration, see “Replica set configuration” on page 341.

You can configure the maximum concurrent sessions policy, as described in “Set the maximum concurrent sessions policy” on page 338. By default, there is no concurrent limit.

You can use the steps that are described in “Test the configuration” on page 338 to verify that the environment is working correctly.

Configuration for WebSEAL instances that are external to the cluster to use the distributed session cache

You can manually configure a WebSEAL instance that is on an appliance, which is external to the distributed session cache server cluster, to use the distributed session cache.

Prerequisites

Familiarize yourself with the concepts that relate to the distributed session cache. For more information about the distributed session cache, see "Distributed session cache overview" on page 327.

See "Advanced configuration for the distributed session cache" on page 338 for information about advanced configuration for your environment.

This configuration summary assumes the following requirements for the WebSEAL environment:

- Forms authentication to WebSEAL.

  Forms authentication is not required for the environment with the distributed session cache. However, basic authentication (the default WebSEAL authentication method) is not suitable for use with session displacement.
- Mutual authentication (SSL) between WebSEAL and the distributed session cache.
- The maximum concurrent sessions policy is enforced.

Manual configuration steps

The following manual configuration steps are required for WebSEAL instances that are running on appliances, which are external to the distributed session cache server cluster. These external appliances can be either software-based WebSEAL instances or appliance-based WebSEAL instances that are not in the same cluster as the distributed session cache server.

1. "Information gathering."
2. "WebSEAL configuration file settings" on page 337.
3. "Restart the WebSEAL server" on page 337.
4. "Create junctions for virtual hosts" on page 338.
5. "Set the maximum concurrent sessions policy" on page 338.
6. "Test the configuration" on page 338.

Information gathering

Configuring WebSEAL to use the distributed session cache requires that you gather information.

You need the following details:

- The host name and port number of the distributed session cache server.
- A key database and stash file for SSL communication with the distributed session cache. The database must contain the SSL certificate that the distributed session cache uses.

  You can find this certificate in the local management interface of the distributed session cache server. Go to the cluster configuration page under **Manage** > **System Settings** > **Cluster Configuration** and select the **Session Cache** tab. There is an **SSL Certificates** link that you can use to access the key database for the distributed session cache and manage the certificates.

  Update the following entries in the WebSEAL configuration file with the key file details so that WebSEAL can access the SSL certificate for the distributed session cache: **`[dsess-cluster], ssl-keyfile`**, **`[dsess-cluster], ssl-keyfile-label`**, and **`[dsess-cluster], ssl-keyfile-stash`**. These stanza entries and values appear in the configuration file as follows:

```
[dsess-cluster]
ssl-keyfile = default-webseald.kdb
ssl-keyfile-label = dsc_cert
ssl-keyfile-stash = default-webseald.sth
```

Note: If the **[dsess-cluster]** entries are not set in a software-based WebSEAL environment, WebSEAL uses the corresponding values in the **[ssl]** stanza. That is, the following entries in the **[ssl]** stanza are used if the **[dsess-cluster]** values are not available:

```
[ssl]
ssl-keyfile = /var/pdweb/keytab-default/default-webseald.kdb
ssl-keyfile-label = dsc_cert
ssl-keyfile-stash = /var/pdweb/keytab-default/default-webseald.sth
```

WebSEAL configuration file settings

The following lists of stanzas and stanza entries in the WebSEAL configuration file represent the complete set of options required for configuring WebSEAL to use the distributed session cache.

Where necessary, edit the WebSEAL configuration file to set these options. Some options may be set by default, based on the original WebSEAL configuration file template.

```
[ses sion]
logout-remove-cookie = yes
dsess-enabled = yes
prompt-for-displacement = yes
dsess-last-access-update-interval = 60
standard-junction-replica-set = replica_set_for_standard_junctions

[replica-sets]
replica-set = replica_set_for_standard_junctions
replica-set = replica_set_for_a_virtual_host_junction
... repeat for all of the replica sets in which
this WebSEAL server participates ...

[dsess]
dsess-sess-id-pool-size = 125
dsess-cluster-name = DSC cluster name

[dsess-cluster]
server = [0-9],<URL>
response-by = 60
handle-pool-size = 10
handle-idle-timeout = 240
timeout = 30
```

Restart the WebSEAL server

After updating the WebSEAL configuration for the distributed session cache, you must restart the WebSEAL server.

Procedure

- Restart the WebSEAL server.
- Check the WebSEAL server log file for any errors or warnings that occur during the startup process. Some warning messages are normal and do not indicate a problem. For example:

  ```
  WebSEAL received notification that the distributed session cache
  for replica-set "replica-set" was cleared.
  All local reference to sessions are being discarded to
  synchronize the local session cache with the distributed session cache.
  ```

  If other warnings or errors occur that are cause for concern, investigate and resolve the problems.

Create junctions for virtual hosts

If you are using virtual hosts, create virtual host junctions.

Procedure

Use the **-z** option to specify the name of the replica set for each virtual host junction.

Set the maximum concurrent sessions policy

Set the maximum concurrent sessions policy for your environment.

About this task

The following example command sets the maximum concurrent sessions policy:

```
pdadmin> policy set max-concurrent-web-sessions displace
```

Test the configuration

When the configuration for the distributed session cache environment is complete, you can test the configuration.

About this task

Check whether any problems occur during the following steps. Examine any errors that are displayed in the browser or in the WebSEAL server log file to help correct the issue.

Procedure

1. Test that logging in works properly:
   - Access the following URL:

     `https://webseal`

     You must be prompted to log in.
   - Log in as a test user.

     You must be allowed access to the website.
2. Verify that the maximum concurrent session policy is enforced:
   - From another server, access the following URL:

     `https://webseal`

     You must be prompted to login.
   - Log in as your test user with the correct password.

     You must be prompted to displace your old session.
   - Click the **Terminate existing login** link.

     You must be allowed access to the Web site.
   - Return to your old browser and access the following URL again:

     `https://webseal`

     If your browser cached the front page of the Web site, you may need to reload (refresh) the page with the Shift key held down.

     You must be prompted to login, because your original session was displaced.

Advanced configuration for the distributed session cache

There are many advanced configuration options for integrating WebSEAL with a distributed session cache solution.

These advanced configuration tasks are primarily for configuring WebSEAL instances that are external to the distributed session cache server cluster. However, these tasks can also be useful to fine-tune the configuration on appliances that are internal to the distributed session cache server cluster.

Topic Index:

Distributed session cache configuration for WebSEAL

There are numerous configuration entries in the **[session]**, **[dsess]**, and **[dsess-cluster]** stanzas for configuring WebSEAL to use the distributed session cache.

This section contains the following topics:

Enabling and disabling the distributed session cache for WebSEAL

Use the **dsess-enabled** stanza entry in the **[session]** stanza of the WebSEAL configuration file to enable and disable use of the distributed session cache by WebSEAL.

About this task

When the distributed session cache is enabled for a clustered WebSEAL server environment, session cookies are used to maintain distributed session state information. The **[session] ssl-id-sessions** stanza entry does not apply when the distributed session cache is in use.

Note: The phrase "dsess" refers to "distributed session".

Procedure

- To enable WebSEAL to use the distributed session cache to maintain user sessions, enter a value of yes. For example:

```
[session]
dsess-enabled = yes
```

- To disable WebSEAL from using the distributed session cache to maintain user sessions, enter a value of no (default). For example:

```
[session]
dsess-enabled = no
```

Specifying the distributed session cache cluster and location

To configure WebSEAL to use the distributed session cache, you must specify the distributed session cache server cluster and location in the **[dsess]** and **[dsess-cluster]** stanzas.

Before you begin

Gather the following information about the cluster that contains the distributed session cache server:

- The IP address of the primary master.
- The IP addresses of the supplementary masters, including the secondary, tertiary, and quaternary masters (if applicable).
- The port for the distributed session cache.

Use the LMI of the primary master to view the **General** and **Session Cache** tabs of Cluster Configuration page to obtain these details. For more information about the cluster configuration fields, search for "Managing cluster configuration" in the Administering Web Reverse Proxy topics in the Knowledge Center.

About this task

Configuration entries for using the distributed session cache are located in the **[dsess]** and **[dsess-cluster]** stanzas of the WebSEAL configuration file.

Procedure

- To specify the location of the distributed session cache, define a cluster name in the **dsess-cluster-name** entry of the **[dsess]** stanza. For example:

  ```
  [dsess]
  dsess-cluster-name = dsess
  ```

  Note: You can assign any name to this cluster. This value is not set as part of the cluster configuration process.
- Next, define the details for the cluster in a corresponding **[dsess-cluster:*<cluster-name>*]** stanza. Use the **server** entry to specify the location of the distributed session cache server in the following format: `https://<IP_Address>:<Port>/DSess/services/DSess`.

  Where:

  <IP_Address>
  The IP address of the distributed session cache server. For example, `10.150.21.80`.

  <Port>
  The port for the distributed session cache. For example, `2126`.

  Note: The default parameters and values to define a cluster of distributed session cache servers are provided in the **[dsess-cluster]** stanza.

  For architectures where more than one distributed session cache is installed in a failover configuration, create multiple instances of this configuration entry. You can specify multiple server entries for failover purposes. The complete set of these server entries defines the membership of a distributed session cache cluster.
- You must specify a priority level for each distributed session cache server by including a number, 1-9, before the URL. This digit represents the priority of the server in the cluster (9 being the highest, 0 being lowest). You must assign the highest priority to the primary master, the next highest priority to the secondary master, and the following priorities to the tertiary and quaternary masters if present.

  For example:

  ```
  [dsess-cluster:dsess]
  server = 9,https://<primary_master_IP_address>:<port>/DSess/services/DSess
  server = 8,https://<secondary_master_IP_address>:<port>/DSess/services/DSess
  server = 7,https://<tertiary_master_IP_address>:<port>/DSess/services/DSess
  server = 6,https://<quaternary_master_IP_address>:<port>/DSess/services/DSess
  ```

  When the **server** entry specifies the HTTPS protocol in the URL, you must configure WebSEAL for SSL communication with the distributed session cache. See “SSL configuration for WebSEAL and the distributed session cache” on page 345.

Retrieving the maximum concurrent sessions policy value

You can use the maximum concurrent sessions policy (**pdadmin policy set max-concurrent-web-sessions**) to control the number of sessions each user can have at one time in a distributed session cache environment.

About this task

By default, this policy is enabled:

```
[session]
enforce-max-sessions-policy = yes
```

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users registered in this secure domain. The policy is stored in the Security Access Manager user registry. To be enforced by the authentication process in a distributed session cache environment, the policy must be retrieved from the registry and stored as an extended attribute in each user's credential.

Replica set configuration

A replica set consists of clustered servers that share sessions. A client session created by one member of a replica set can be used unmodified by another.

You must specify the name of each replica set used in your server environment. By convention, set the replica set names to match the DNS name (fully-qualified host name) used by a website. For example, if the website DNS name is `www.example.com`, set the replica set name for the website to `www.example.com`.

You must specify each replica set name in the configuration file of each WebSEAL instance that participates in those replica sets. Additionally, you must assign each junctioned or virtual host to the appropriate replica set. There are different procedures for assigning standard junctions and virtual hosts to a replica set. See:

Configuring WebSEAL to participate in multiple replica sets

Each replica set that WebSEAL participates in must be listed in the **[replica-sets]** stanza of the WebSEAL configuration file.

Example

If WebSEAL participates in replica sets named `vhostA.example.com`, `vhostB.example.com`, and `www.example.com`, the stanza is configured as follows:

```
[replica-sets]
replica-set = vhostA.example.com
replica-set = vhostB.example.com
replica-set = www.example.com
```

Assigning standard junctions to a replica set

You must configure the standard junction replica set for the distributed session cache.

About this task

By design, all standard junctions for a WebSEAL instance are assigned to one replica set, as specified by the **`standard-junction-replica-set`** stanza entry in the **`[session]`** stanza for the WebSEAL configuration file.

To use the distributed session cache, the **standard-junction-replica-set** stanza entry value must also be listed in the **[replica-sets]** stanza. If the **standard-junction-replica-set** value is not present in the **[replica-sets]** stanza, WebSEAL does not start.

```
[session]
standard-junction-replica-set = replica-set-name

[replica-sets]
replica-set = replica-set-name
```

Example

```
[session]
standard-junction-replica-set = www.example.com

[replica-sets]
replica-set = www.example.com
```

Virtual hosts assigned to a replica set

In contrast to standard junctions, virtual hosts can be individually assigned to different replica sets by using the **-z** option of the **pdadmin server task virtualhost create** command.

When the distributed session cache is in operation, the **-z** option specifies the replica set that sessions on the virtual host junction are managed under. If WebSEAL is configured to use the distributed session cache, then the replica set for a virtual host junction can be specified with the **-z** option. Different virtual host junctions must be assigned to different replica sets. If the **-z** option is not used, then WebSEAL uses the virtual host name of the junction as the name of the replica set.

Additionally, the name of the replica set used by this virtual host must be defined by the **replica-set** stanza entry in the **[replica-sets]** stanza of the configuration file for the WebSEAL instance:

```
[replica-sets]
replica-set = replica-set-name
```

Example conditions:

- Virtual host type: TCP
- Host name for the machine where the virtual host resides: abc.example.com
- Virtual host name: vh1.example.com
- Virtual host belongs to this replica set: vh1.example.com
- Virtual host junction label: vhost-vh1-http

Example virtual host create command (entered as one line):

```
pdadmin> server task default-webseald-webseal.example.com virtualhost create
-t tcp -h abc.example.com -v vh1.example.com -z vh1.example.com vhost-vh1-http
```

Example WebSEAL configuration for replica sets:

```
[replica-sets]
replica-set = vh1.example.com
```

Adjustment of the last access time update frequency for the distributed session cache

The **dsess-last-access-update-interval** stanza entry in the **[session]** stanza of the WebSEAL configuration file specifies the frequency at which WebSEAL updates the session last access time at the distributed session cache.

If you are adjusting session inactivity timeouts or configuring reauthentication based on session inactivity policy (reauth-for-inactive = yes), and you are using the distributed session cache, you might need to adjust this value.

Smaller values offer more accurate inactivity timeout tracking, at the expense of sending updates to the distributed session cache more frequently. Values of less than 1 second are not permitted. The default value is 60 seconds. For example:

```
[session]
dsess-last-access-update-interval = 60
```

As an example, consider the following configuration:

```
[session]
inactive-timeout = 600
dsess-last-access-update-interval = 60
```

With these configuration values, a user's session may be flagged as "inactive" at the distributed session cache anywhere between 540 seconds and 600 seconds after the user's last access to the WebSEAL server.

Small values for the **dsess-last-access-update-interval** parameter are not recommended and can seriously impact WebSEAL server performance.

See also "Reauthentication with external authentication interface" on page 273.

See also "Cache entry inactivity timeout value" on page 289

Communication timeout configuration for the distributed session cache

There are two configuration entries to control the communication timeout values for the distributed session cache environment.

This section contains the following topics:

Configuring the response timeout for the distributed session cache

You can configure the maximum amount of time that WebSEAL waits for a response from the distributed session cache.

About this task

Use the **timeout** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file to specify the amount of time (in seconds) WebSEAL can wait for a response from the distributed session cache.

The default value is 30 seconds. For example:

```
[dsess-cluster]
timeout = 30
```

If the timeout limit is reached with no response from the distributed session cache, WebSEAL assumes the distributed session cache is unavailable. When this occurs, the following actions are taken:

- A separate WebSEAL server thread begins attempting to contact the distributed session cache every 60 seconds to see if the distributed session cache has recovered or a backup has come online.
- All attempts to create or access a session on the WebSEAL server receive an HTTP "503 Service Unavailable" error page from the WebSEAL server (`38b9a4b1.html`). You can customize this page by creating an error page for error status "0x38b9a4b1" as described in “HTML server response page modification” on page 65.

When the distributed session cache recovers, WebSEAL attempts to determine whether the outage was due to a temporary network outage or if the distributed session cache server was restarted. If the distributed session cache server was restarted, the local WebSEAL session cache is cleared. All sessions on the WebSEAL server are deleted. This is done so that sessions across all of the WebSEAL servers in the cluster remain synchronized.

Configuring connection timeout for broadcast events

You can control the maximum amount of time that WebSEAL keeps its connection open and waits for a broadcast event from the distributed session cache cluster.

About this task

Some clustered server architectures may implement a firewall between the WebSEAL cluster members and the appliance that runs the distributed session cache. Firewalls often restrict the flow of communication to one direction. WebSEAL communicates through the firewall to send session information to the distributed session cache.

To additionally receive broadcast events from the distributed session cache, WebSEAL must open another connection through the firewall. The firewall timeout policy can shut down this connection while WebSEAL is waiting for broadcast events from the distributed session cache.

Procedure

Use the **response-by** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file to specify the length of time (in seconds) that WebSEAL keeps a connection open to the distributed session cache for receiving broadcast events for the distributed session cache cluster. When the timeout value is reached, WebSEAL recreates a new connection.

```
[dsess-cluster]
response-by = 60
```

To ensure the most optimal conditions for keeping this connection open, set the **response-by** stanza entry value to be less than the internal firewall timeout value.

Performance configuration for the distributed session cache

You can configure the maximum number of pre-allocated session IDs and the handle pool size to fine-tune the performance of the distributed session cache.

This section contains the following topics:

Maximum pre-allocated session IDs

The **dsess-sess-id-pool-size** stanza entry in the **[dsess]** stanza of the WebSEAL configuration file specifies the maximum number of session IDs that are pre-allocated in the session ID pool.

This stanza entry is used by the server cluster. This setting is required when the distributed session cache is enabled.

```
[dsess]
dsess-sess-id-pool-size = 125
```

Configuration of the handle pool size

The **handle-pool-size** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file specifies the maximum number of idle Simple Access Object Protocol (SOAP) handles that the distributed session client maintains at any given time.

These handles are used for anticipated exchanges with the distributed session cache. The default value is 10.

For example:

```
[dsess-cluster]
handle-pool-size = 10
```

The default value is adequate for most environments.

The value can be increased if communication between WebSEAL and the distributed session cache is slow because all handles are in use. If you decide to modify this value, be aware that each handle reserves a file descriptor on the WebSEAL server. A large value can prevent other WebSEAL functionality that requires file descriptors from working properly.

The maximum value for the **handle-pool-size** depends on both the platform on which the WebSEAL server is running and various WebSEAL configuration options. In general, do not increase the **handle-pool-size** beyond 25 handles.

SSL configuration for WebSEAL and the distributed session cache

When the **[dsess-cluster] server** stanza entry specifies the HTTPS protocol in the URL, you must configure WebSEAL for SSL communication with the distributed session cache. WebSEAL can authenticate to the distributed session cache with client certificates.

Configuring WebSEAL for SSL communication with the distributed session cache requires that you provide WebSEAL the following information:

- The CA certificate used to sign the distributed session cache SSL server certificate.
- The DN contained in the distributed session cache SSL server certificate.

You can also configure additional GSKit attributes to use when initializing the SSL connection with the distributed session cache.

Note: This SSL configuration is only required for WebSEAL instances that are external to the distributed session cache server cluster. For appliances that are in the same cluster as the distributed session cache, no manual SSL configuration is required. The SSL configuration is automatically set up by the **Enable Distributed Session Cache** option.

This section contains the following topics:

Configuring the WebSEAL key database

WebSEAL stores client-side certificates and CA root certificates, used for SSL communication with the distributed session cache, in a key database file.

About this task

The purpose of each certificate is as follows:

- The CA root certificate is used to validate the server certificate returned by the distributed session cache.
- The client-side certificate is used by WebSEAL to communicate with the distributed session cache server.

Procedure

- To specify the key database file, use the **ssl-keyfile** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file. For example:

  ```
  [dsess-cluster]
  ssl-keyfile = key-file-name
  ```

  Unless Security Access Manager SSL certificates are being used for communication between WebSEAL and the distributed session cache, use a separate key file from the other WebSEAL key files as the value for **ssl-keyfile**.
- To specify the key database stash file (containing password information for access to the database file), use the **ssl-keyfile-stash** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file. For example:

  ```
  [dsess-cluster]
  ssl-keyfile-stash = key-file-name
  ```
- To specify the label name for the client-side certificate, use the **ssl-keyfile-label** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file. For example:

  ```
  [dsess-cluster]
  ssl-keyfile-label = label-name
  ```

Specifying the SSL certificate distinguished name (DN)

The CA root certificate that is stored in a WebSEAL key database file validates that a certificate received from the distributed session cache is authentic. By additionally checking the DN value in the certificate, you can ensure that the server certificate received by WebSEAL from the distributed session cache is the expected certificate.

About this task

To specify the accepted certificate DN values, use the **ssl-valid-server-dn** stanza entry in the **[dsess-cluster]** stanza of the WebSEAL configuration file.

Example

```
[dsess-cluster]
ssl-valid-server-dn = DN-value
```

Obtaining the server certificate DN value:

The **ssl-valid-server-dn** in the **[dsess-cluster]** stanza of the WebSEAL configuration file requires the value of the DN found in a valid server certificate sent by the distributed session cache during its communication with WebSEAL.

About this task

You can obtain the DN value from the distributed session cache administrator directly.

Alternatively, you can indirectly determine the value by performing the following procedure:

Procedure

1. Enable the distributed session cache for WebSEAL:
   ```
   [session] dsess-enabled = yes
   ```
2. Ensure that the distributed session cache is configured for SSL. The URL to the distributed session cache requires the HTTPS protocol:
   ```
   [dsess-cluster] server = https://server/DSess/services/DSess
   ```
3. Follow the procedures for configuring the **ssl-keyfile**, **ssl-keyfile-stash**, and **ssl-keyfile-label** stanza entries in the **[dsess-cluster]** stanza of the WebSEAL configuration file. See "Configuring the WebSEAL key database" on page 346.
4. Enter a test value for the **ssl-valid-server-dn** stanza entry. For example:
   ```
   [dsess-cluster] ssl-valid-server-dn = test
   ```
5. Restart the WebSEAL server.
6. WebSEAL returns the following error message:
   ```
   The DN contained within the server certificate, <DN>, is not a configured DN.
   ```
   The DN listed in the message is the DN of the certificate presented by the distributed session cache.

   Use this value to correctly specify the value for the **ssl-valid-server-dn** stanza entry.
7. To verify you are communicating with the right SSL server, confirm, with the distributed session cache administrator, the value for the DN returned in the error message.

   Once you are sure you have the right value for the DN of the distributed session cache server certificate, use that DN for the value of the **ssl-valid-server-dn** stanza entry.

GSKit configuration for distributed session cache connections

There are a number of GSKit attributes that you can use to control how GSKit creates SSL connections. You can configure WebSEAL to use particular GSKit attributes when it initializes SSL connections.

The **gsk-attr-name** configuration entry in the **[dsess-cluster]** stanza controls the GSKit attributes that WebSEAL uses when initializing a connection with the distributed session cache. You can specify this configuration entry multiple times. Include each desired GSKit attribute as a new entry.

```
[dsess-cluster]
gsk-attr-name = {enum | string | number}:id:value
```

Note: Similar configuration entries exist in the **[ssl]** stanza for connections with clients and junctioned web servers.

For further details about these configuration entries, see the Web Reverse Proxy Stanza Reference topics in the IBM Knowledge Center.

Maximum concurrent sessions policy

This section contains the following topics:

Setting the maximum concurrent sessions policy

You can control the number of sessions each user can have at one time in a distributed session environment managed by the distributed session cache. The **pdadmin policy set max-concurrent-web-sessions** command specifies this maximum number of concurrent sessions.

About this task

As the administrator, you can apply this policy to a specific user or apply the policy globally to all users registered in this secure domain. See "Per user and global settings" on page 350.

Use the **enforce-max-sessions-policy** stanza entry in the **[session]** stanza of the WebSEAL configuration file to control whether or not a specific WebSEAL instance enforces the **max-concurrent-web-sessions** policy. See "Enforcing the maximum concurrent sessions policy" on page 351.

Command syntax for pdadmin policy (each entered as one line):

```
policy set max-concurrent-web-sessions {unset|number|displace|unlimited}
[-user username]

policy get max-concurrent-web-sessions [-user username]
```

Argument descriptions for pdadmin policy set:

- `unset`

  Disables the **max-concurrent-web-sessions** policy. With this setting, the policy contains no value. The effective policy for the user is the same as the `unlimited` setting.

  The `unset` setting is the default policy.

  For example (global setting):

  ```
  pdadmin> policy set max-concurrent-web-sessions unset
  ```

- ***number***

  Specifies the number of concurrent sessions allowed per user. The user is prevented from establishing more sessions beyond this number.

  For example (global setting):

  ```
  pdadmin> policy set max-concurrent-web-sessions 2
  ```

  A error response page (`38b9a41f.html` "Additional Login Denied") is returned to the user when a login attempt is made that exceeds this value.

- `unlimited`

  Allows an unlimited number of concurrent sessions per user.

  For example (global setting):

  ```
  pdadmin> policy set max-concurrent-web-sessions unlimited
  ```

- `displace`

Limits users to one active session at one time by forcing a value of 1 session for **max-concurrent-web-sessions** policy.

For example (global setting):

```
pdadmin> policy set max-concurrent-web-sessions displace
```

The response to additional login attempts is governed by the **prompt-for-displacement** in the **[session]** stanza of the WebSEAL configuration file.

See "Interactive displacement" and "Non-interactive displacement" on page 350.

Interactive displacement:

The **prompt-for-displacement** stanza entry in the **[session]** stanza of the WebSEAL configuration file determines whether or not a user is prompted for appropriate action when the **max-concurrent-web-sessions displace** policy has been exceeded. This section discusses the interactive option (`prompt-for-displacement = yes`), where the user is prompted for appropriate action.

Example configuration:

- Policy setting (global example):

  ```
  pdadmin> policy set max-concurrent-web-sessions displace
  ```

- Prompt setting:

  ```
  [session]
  prompt-for-displacement = yes
  ```

When a second login is attempted, the user receives the `too_many_sessions.html` response page. You can customize the contents of this page. The default message on this page states:

```
You are already logged in from another client. Do you want to terminate
your existing login or cancel this new login request?

Terminate existing login
Cancel this new login
```

Action descriptions:

- **Terminate existing login**

  The terminate action calls the WebSEAL **/pkmsdisplace** function. This function terminates the existing (original) login, creates a new session for the user, logs the user in transparently, and redirects the user to the requested URL.

  Note: The `pkmsdisplace` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

  The original session cookie remaining on the user's original browser becomes an old, or "stale" cookie that no longer maps to an existing entry in the WebSEAL session cache. If the user attempts to access another protected resource from the original (older) login session, WebSEAL requires authentication and responds with the standard login form.

  The OLDSESSION macro contained in this form is set to the value of "1", indicating that the request contains an old ("stale") cookie that no longer matches any entry in the WebSEAL session cache. You can use the value of the OLDSESSION macro as a trigger mechanism for a customized response to the user. This custom response could more accurately explain to the user why the session is not valid anymore.

For further information on this feature, see “Customized responses for old session cookies” on page 315.

- **Cancel this new login**

  The cancel action calls the WebSEAL **/pkmslogout** function. This function closes the current login attempt and returns the standard WebSEAL logout page to the user. The original (older) login session can continue accessing resources.

Prerequisite: Maximum concurrent sessions policy must be enabled through an additional configuration. See “Enforcing the maximum concurrent sessions policy” on page 351.

Non-interactive displacement:

The **prompt-for-displacement** stanza entry in the **[session]** stanza of the WebSEAL configuration file determines whether or not a user is prompted for appropriate action when the **max-concurrent-web-sessions displace** policy has been exceeded. This section discusses the non-interactive option (`prompt-for-displacement = no`), where the user is not prompted for appropriate action.

Example configuration:

- Policy setting (global example):

  ```
  pdadmin> policy set max-concurrent-web-sessions displace
  ```

- Prompt setting:

  ```
  [session]
  prompt-for-displacement = no
  ```

When a second login is attempted, the original (older) login session is automatically terminated with no prompt. A new session is created for the user and the user is logged in to this new session transparently. The original (older) session is no longer valid.

The original session cookie remaining on the user's original browser becomes an old, or "stale" cookie that no longer maps to an existing entry in the WebSEAL session cache. If the user attempts to access another protected resource from the original (older) login session, WebSEAL requires authentication and responds with the standard login form.

The OLDSESSION macro contained in this form is set to the value of "1", indicating that the request contains an old ("stale") cookie that no longer matches any existing entry in the WebSEAL session cache. You can use the value of the OLDSESSION macro as a trigger mechanism for a customized response to the user. This custom response could more accurately explain to the user why the session is not valid anymore.

For further information on this feature, see “Customized responses for old session cookies” on page 315.

Per 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 a policy (with the **unset** argument). With this setting, the policy contains no value.

Examples:

A global maximum concurrent web sessions policy of 1 session per user is created. As an exception to this policy, user **brian** is given a maximum concurrent web sessions policy of 4 sessions.

```
pdadmin> policy set max-concurrent-web-sessions 1
pdadmin> policy set max-concurrent-web-sessions 4 -user brian

pdadmin> policy get max-concurrent-web-sessions
Maximum concurrent web sessions: 1
pdadmin> policy get max-concurrent-web-sessions -user brian
Maximum concurrent web sessions: 4
```

***

The specific maximum concurrent web sessions policy for user **brian** is unset. User **brian** now has no maximum concurrent web sessions policy. However, user Brian is effectively governed by the current global maximum concurrent web sessions policy of 1 session.

```
pdadmin> policy set max-concurrent-web-sessions unset -user brian

pdadmin> policy get max-concurrent-web-sessions -user brian
Maximum concurrent web sessions: unset
```

***

The global maximum concurrent web sessions policy is unset. All users, including user **brian**, now have no maximum concurrent web sessions policy. However, the effective policy for all users is the same as the **unlimited** setting.

```
pdadmin> policy set max-concurrent-web-sessions unset

pdadmin> policy get max-concurrent-web-sessions
Maximum concurrent web sessions: unset
```

Enforcing the maximum concurrent sessions policy

Use the **enforce-max-sessions-policy** stanza entry in the **[session]** stanza of the WebSEAL configuration file to control whether or not a specific WebSEAL instance enforces the **max-concurrent-web-sessions** policy.

Procedure

- To set this WebSEAL instance to enforce the **max-concurrent-web-sessions** policy, enter a value of yes (default). For example:

  ```
  [session]
  enforce-max-sessions-policy = yes
  ```

- To set this WebSEAL instance to not enforce the **max-concurrent-web-sessions** policy, enter a value of no. For example:

  ```
  [session]
  enforce-max-sessions-policy = no
  ```

  Note: This stanza entry is effective only when you have configured the distributed session cache to manage sessions for your environment.

  ```
  [session]
  dsess-enabled=yes
  ```

  By default, all systems in the distributed session environment enforce this policy:

  ```
  [session]
  enforce-max-sessions-policy = yes
  ```

- You can modify the **enforce-max-sessions-policy** stanza entry for specific WebSEAL instances in the same environment to disable enforcement of the **max-concurrent-web-sessions** policy:

  ```
  [session]
  enforce-max-sessions-policy = no
  ```

  Users accessing those WebSEAL servers with `enforce-max-sessions-policy = no` can have unlimited login sessions.

  For information on setting the maximum concurrent sessions policy, see "Setting the maximum concurrent sessions policy" on page 348.

  Note: Maximum concurrent sessions policy is enforced on a per replica set basis.

Example

Use the **pdadmin policy set** command to globally specify a maximum concurrent session policy of 1:

```
pdadmin> policy set max-concurrent-web-sessions 1
```

Switch user and maximum concurrent sessions policy

When an administrator uses switch user to impersonate another user, the session at the distributed session cache is considered to belong to the switch user administrator. The maximum concurrent sessions policy applies to the switch user administrator, and not the impersonated user.

For example: if the user "brian" has a maximum concurrent sessions policy of 1 and is logged in to the WebSEAL server, a switch user administrator is still able to impersonate "brian." The maximum concurrent sessions policy for user "brian" does not apply to the impersonated session.

Single signon in a replica set

A distributed session cache environment provides a single sign-on solution by sharing sessions across servers in a replica set.

Topic index:

Replica set and session sharing concepts

A *replica set* is a collection of Web security servers that are configured to share sessions. Session sharing allows single signon among all servers in the replica set while enforcing concurrent session limitations and session terminations.

A user can log on to any server in the replica set without authenticating again.

For example, as a user, you log in to the main website for your company: **www.example.com**. The **www.example.com** site is handled by a WebSEAL cluster where all WebSEAL servers belong to the "**www.example.com**" replica set.

WebSEAL is configured to provide you (as an authenticated user) with a domain session cookie for **.example.com**.

Later in the session, you access **sales.example.com**, which is the main website for the company's sales department. The **sales.example.com** site is handled by a WebSEAL cluster where all WebSEAL servers belong to the "**sales.example.com**" replica set.

The distributed session cache configuration manages all of the replica sets. In this example, **www.example.com** and **sales.example.com** are both configured as replica sets.

The **sales.example.com** WebSEAL cluster uses your domain session cookie to acquire your session information at **www.example.com**. With this session information, you are not asked to authenticate again and single signon is achieved.

For session sharing to function correctly, all of the following conditions must be met:

- The values for session lifetime and inactivity timeouts on all servers in the replica sets must be identical.
- Authentication configuration and policy on all servers in the replica sets must be compatible.

  As an example of an incompatible configuration, consider the following:

  - **www.example.com** is configured for forms authentication.
  - **test.example.com** is configured for EAI authentication.
  - The resource **www.example.com/action.jsp** is protected by a POP requiring reauthentication.

  If a user logs on to **test.example.com** and then accesses **www.example.com**, the user is able to access most resources on **www.example.com**. However, the user is not able to access **www.example.com/action.jsp** because the user cannot perform an EAI reauthentication on **www.example.com**.

Configuring session sharing

There are two high level steps to configure session sharing in a distributed session cache environment.

About this task

- Configure all Web security servers in the replica set to use the same name for session cookies.

  See "Configuring session cookie names."
- Configure all Web security servers in the same replica set with a list of DNS domains that should be used for domain session cookies.

  See "Configuring DNS domains" on page 354.

Configuring session cookie names:

Configure all Web security servers in the replica set to use the same name for session cookies.

About this task

The cookie name used for WebSEAL session cookies is specified by the **`tcp-session-cookie-name`** and **`ssl-session-cookie-name`** stanza entries in the **`[session]`** stanza of the WebSEAL configuration file. For example (default WebSEAL cookie names for TCP and SSL sessions):

```
[session]
tcp-session-cookie-name = PD-H-SESSION-ID
ssl-session-cookie-name = PD-S-SESSION-ID
```

See also "Customization of the session cookie name" on page 314.

Configuring DNS domains:

Configure all of the Web security servers in the same replica set with a list of DNS domains that are used for domain session cookies.

About this task

WebSEAL session cookies are the required session ID data type used in a WebSEAL environment using the distributed session cache. Normally, session cookies are server-specific (or host) cookies. A browser only returns a host type cookie to the server that originally sent the cookie.

To enable session sharing across multiple clusters in the same DNS domain, it is necessary to use domain type cookies.

You define participating domains in the **domain** stanza entry, located in the **[session-cookie-domains]** stanza of the WebSEAL configuration file.

When a **domain** stanza entry is uncommented and provided with a value, WebSEAL session cookies automatically become domain type cookies. For example:

```
[session-cookie-domains]
domain = example1.com
domain = example2.com
```

WebSEAL decides what domain to use for domain session cookies based on the website the user connects to and the **[session-cookie-domains]** stanza. Add an entry to this stanza matching the domain used for the replica set.

As an example, consider a WebSEAL server with three virtual hosts:

```
www.example.com
www.abc.ibm.com
www.tivoli.com
```

The following conditions apply to this example:

- The **www.example.com** site participates in the "**example.com**" replica set.
- The **www.abc.ibm.com** site participates in the "**abc.ibm.com**" replica set.
- The **www.tivoli.com** site does not share sessions with other Web security servers, so is not assigned to a replica set.

To configure domain session cookies for both the **example.com** and **abc.ibm.com** domains, set the following configuration options in each participating Web security server:

```
[session-cookie-domains]
domain = example.com
domain = abc.ibm.com
```

Chapter 6. Authorization

Configuration for authorization

This chapter discusses WebSEAL functions that affect the authorization service and process.

Topic Index:

WebSEAL-specific ACL policies

The following security considerations apply for the `/WebSEAL` container in the protected object space:

- The `WebSEAL` object begins the chain of ACL inheritance for the WebSEAL region of the object space.
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire Web space.
- The traverse permission is required for access to any object below this point.

/WebSEAL/*host-instance_name*

This subdirectory entry represents the beginning of the Web space for a particular WebSEAL instance. The following security considerations apply for this object:

- The traverse permission is required for access to any object below this point.
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire object space on this machine.

/WebSEAL/*host-instance_name*/*file*

This subdirectory entry represents the resource object checked for HTTP access.

The permissions checked depend on the operation being requested.

WebSEAL ACL permissions

The following table describes the ACL permissions applicable for the WebSEAL region of the object space:

	Operation	Description
r	read	View the Web object.
x	execute	Run the CGI program.
d	delete	Remove the Web object from the Web space.
m	modify	PUT an HTTP object. (Place - publish - an HTTP object in the WebSEAL object space.)
l	list	Required by policy server to generate an automated directory listing of the Web space. This permission also governs whether a client can see the directory contents listing when the default "index.html" page is not present.

Default /WebSEAL ACL policy

Core entries for the WebSEAL ACL, **default-webseal**, include:

```
Group iv-admin           Tcmdbsvarxl
Group webseal-servers    Tgmdbsrxl
User sec_master          Tcmdbsvarxl
Any-other                Trx
Unauthenticated          T
```

At installation, this default ACL is attached to the **/WebSEAL** container object in the object space.

The group, **webseal-servers,** contains an entry for each WebSEAL server in the secure domain. The default permissions allow the servers to respond to browser requests.

The traverse permission allows expansion of the Web space. The list permission allows the Web Portal Manager to display the contents of the Web space.

Valid characters for ACL names

The following characters are valid for creating ACL names:

- A-Z
- a-z
- 0-9
- underscore (_)
- hyphen (-)
- backslash (\)
- Any character from a double-byte character set

For detailed information about creating ACL names, see the Web command reference topics in the IBM Knowledge Center.

Quality of protection POP

The protected object policy (POP) attribute for quality of protection allows you to specify what level of data protection is required when performing an operation on an object.

The quality of protection POP attribute is used to determine whether access will be granted to a requested resource. When an ACL check for a resource succeeds, the quality of protection POP is checked. If a quality of protection POP exists, and the resource manager (WebSEAL) cannot guarantee the required level of protection, the request is denied.

The syntax for setting the quality of protection POP attribute is as follows:

```
pdadmin> pop modify pop-name set qop {none|integrity|privacy}
```

When the quality of protection level is set to either integrity or privacy, WebSEAL requires data encryption through the use of Secure Socket Layer (SSL).

For example:

```
pdadmin> pop modify test set qop privacy
```

Configuration of authorization database updates and polling

This section contains the following topics:

Database update and polling concepts:

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

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

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

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

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

Configuration of update notification listening:

The **listen-flags** stanza entry, located in the **[aznapi-configuration]** stanza of the WebSEAL configuration file, enables and disables update notification listening by WebSEAL. By default, notification listening is enabled. To disable notification listening, enter "disable".

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

The **ssl-listening-port** stanza entry, located in the **[ssl]** stanza of the WebSEAL configuration file, specifies the SSL port for the notification listener:

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

Note: Do not change the **ssl-listening-port** entry directly; this option should be modified only by issuing the **scrsslcfg -chgport** command so that the policy server can detect that the listening port has been changed. Otherwise, the resource manager cannot receive policy update notifications or pdadmin server task commands.

Configuration of authorization database polling:

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

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

Configuring quality of protection levels

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

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

1. Enable quality of protection management:

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

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

   ```
   default = {ALL|NONE|cipher_level}
   ```

   Supported values for *cipher_level* are:

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

   The value "NONE" disables encryption.

   For example:

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

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

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

   Notes:

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

- FIPS-DES-56 and FIPS-DES-168 are available only when fips-mode-processing is enabled (set to yes). Otherwise they are ignored

Security Access Manager uses GSKit 8. The Cipher specifications supported by GSKIT 8 when used in SSLv2/TLS in Internet security are:

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

These TLS cipher specifications are also used with SSLV3.

Configuration of QOP for individual hosts and networks:

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

Note: The **[ssl-qop-mgmt-hosts]** and **[ssl-qop-mgmt-networks]** stanzas are provided for compatibility with prior versions of WebSEAL only. It is recommended that you not use them for Security Access Manager configuration. Additionally, Internet Protocol version 6 (IPv6) addresses are not supported by these stanzas.

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

Example configuration syntax for hosts:

```
[ssl-qop-mgmt-hosts]
xxx.xxx.xxx.xxx = ALL
yyy.yyy.yyy.yyy = RC2-128
```

Example configuration syntax for network/netmask:

```
[ssl-qop-mgmt-networks]
xxx.xxx.xxx.xxx/255.255.255.0 = RC4-128
yyy.yyy.yyy.yyy/255.255.0.0 = DES-56
```

Note that the entry for an IP address specified under **[ssl-qop-mgmt-hosts]** takes priority over an entry for the same address in **[ssl-qop-mgmt-networks]**. Likewise, an entry in **[ssl-qop-mgmt-networks]** takes priority over an entry for the same address in **[ssl-qop-mgmt-default]**.

If you must use **[ssl-qop-mgmt-hosts]** or **[ssl-qop-mgmt-networks]** for compatibility concerns, review the IP address settings under all stanzas to ensure that a specific IP address is not listed under more than one stanza. If an IP address is listed under more than one stanza, ensure that the order of evaluation yields the desired configuration.

Authorization decision information from HTTP requests

WebSEAL can pass configured elements from the HTTP request to the authorization framework for use when it is making authorization decisions.

The following HTTP request elements can be passed to the authorization framework:

- The HTTP method of the request
- The HTTP scheme of the request
- The request URI
- The client IP address
- Specific HTTP headers contained in the request
- Specific POST data elements that are contained in the request. WebSEAL supports two types of POST data:
  - Normal FORM data, which is the `application/x-www-form-urlencoded` content-type.
  - JavaScript Object Notation (JSON) data, which is the `application/json` content-type. For more information about the JSON syntax, see http://www.json.org.

The `[azn-decision-info]` stanza in the WebSEAL configuration file specifies the extra information that is passed to the authorization framework.

For more information about how to configure WebSEAL to pass extra information to the authorization framework, see the Web Reverse Proxy Stanza Reference topics.

Support for OAuth authorization decisions

OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end user). It also provides a process for end-users to authorize third party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

WebSEAL supports the EAS plug-in, which leverages OAuth 2.0 capabilities. This plug-in allows OAuth decisions to be made as a part of the standard authorization on WebSEAL requests. This functionality uses the authorization server of the Advanced Access Control Module to reject or authorize OAuth tokens in your environment. For more information, see OAuth 2.0 support.

High level overview of the OAuth EAS:

At a high level, the EAS framework allows a custom module to be called during an authorization decision to add customized decision making logic to the authorization decision. The configuration of the EAS controls whether the module is invoked when a specific POP is encountered, or whether it is invoked when a specific permission bit is applied to the authorization decision.

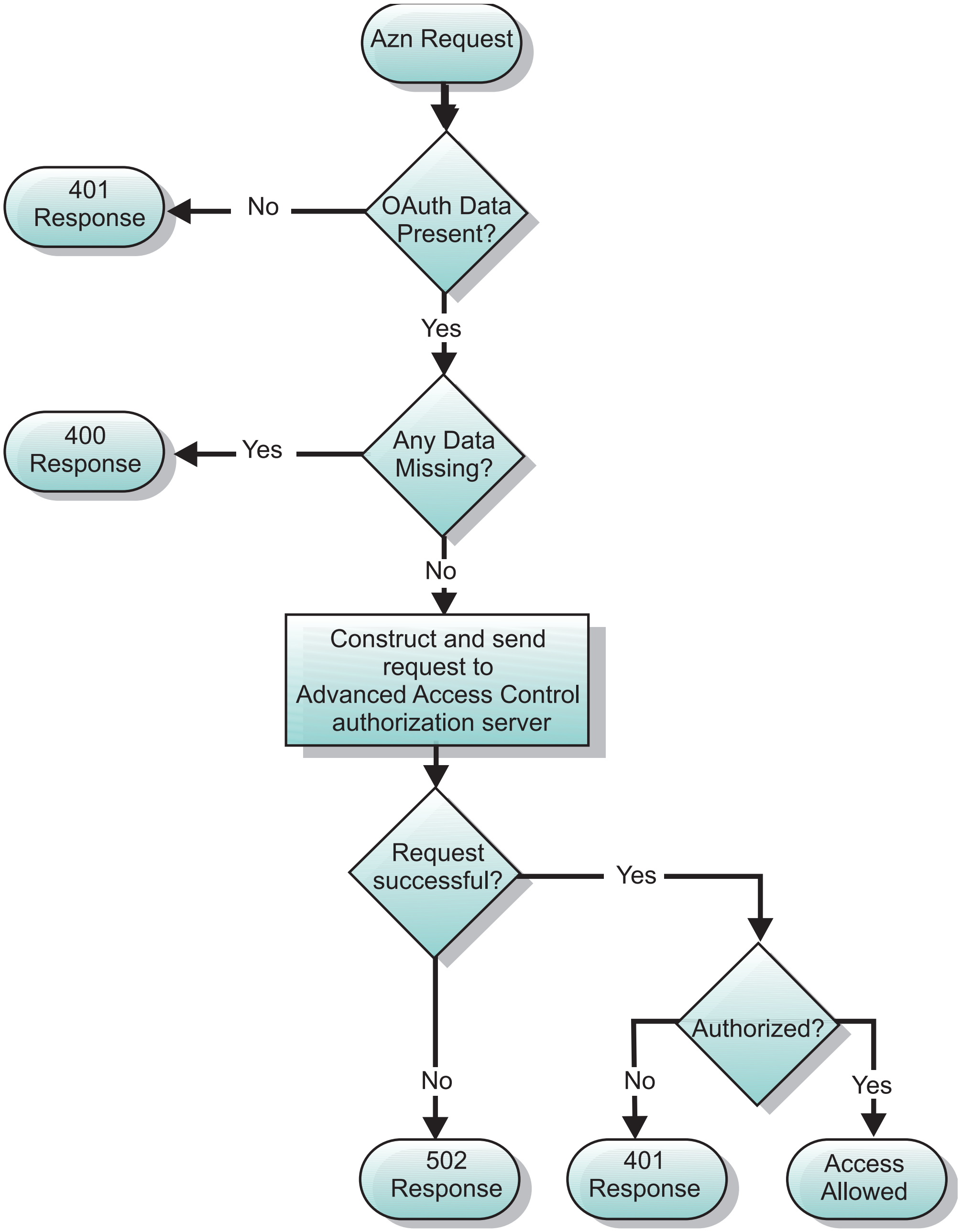


Figure 25. Logical flow of the OAuth EAS

Figure 25 shows the logical flow through the OAuth EAS when making an authorization decision. The following steps outline this authorization process:

1. The OAuth EAS receives the authorization request.
2. The OAuth EAS determines whether the required OAuth data is present.
3. If there is missing OAuth data, then a 401 response is generated and no further processing takes place. If the required OAuth data is available, then proceed to the next step.
4. EAS verifies that all required data is available in the request.
5. If there is missing data, then a 400 response is generated and no further processing takes place. If all of the data is available, then the EAS constructs a

Request Security Token (RST) and sends it to the authorization server, which is part of the Advanced Access Control Module.
6. The authorization server processes the request. If the processing fails, then a 502 response is generated and no further processing takes place. Otherwise, the authorization server returns the access decision to the OAuth EAS.
7. If the request is authorized, then access is granted to the requested resource. If the request is not authorized, then a 401 response is generated.

Configuring WebSEAL to include OAuth decisions:

To make an OAuth authorization decision, the authorization server requires specific information regarding the request. The required data includes the following:

- **Authorization data.** This data is obtained from either the authorization header, the query string or the POST data.
- **Resource information.** This data is obtained from the HTTP request and is used to validate the OAuth signature.

WebSEAL uses the EAS plug-in to provide this required data and to use the OAuth functionality in the Advanced Access Control Module.

To include OAuth decisions as part of the standard authorization on WebSEAL requests, you need to perform the following tasks:

1. Configure the required authorization decision data.
2. Configure the extra EAS specific data.

This configuration ensures that the correct data is passed to the EAS for each request.

Authorization decision data:

To correctly construct the RST, the EAS requires various information from the request itself. WebSEAL must be configured to provide this information to the EAS.

The majority of the required data is provided on every authorization request by specifying these HTTP request elements in the **[azn-decision-info]** stanza. See "Authorization decision information from HTTP requests" on page 360.

Note: In certain situations, the POST data is also required. For efficiency, the EAS plug-in does not provide the POST data on every authorization decision request. Instead, the plug-in uses the existing dynamic access decision information within WebSEAL to optionally request the POST data when required. WebSEAL recognizes the request for POST data based on the **resource-manager-provided-adi** configuration entry in the **[aznapi-configuration]** stanza.

It is vital that this configuration stanza is correct so that the data is passed to the EAS. The following configuration entries are required in order for the EAS to function correctly:

```
[azn-decision-info]

#
# The following information will be provided to the authorization
# framework for every authorization request.  This information
# is required by the OAuth EAS when validating an OAuth token.
```

```
#

HTTP_REQUEST_METHOD        = method
HTTP_REQUEST_SCHEME        = scheme
HTTP_REQUEST_URI           = uri
HTTP_HOST_HDR              = header:host
HTTP_CONTENT_TYPE_HDR      = header:content-type
HTTP_TRANSFER_ENCODING_HDR = header:transfer-encoding
HTTP_AZN_HDR               = header:authorization

[aznapi-configuration]

resource-manager-provided-adi = AMWS_pb_
```

EAS specific data:

The EAS requires specific configuration data to function correctly. This data is mostly contained in the **[oauth]** and **[oauth-eas]** stanzas.

One of the required configuration entries in the **[oauth]** stanza is **cluster-name**, which specifies the name of the authorization server. You must configure a corresponding **[tfim-cluster:<cluster>]** stanza to define the specified cluster.

The following excerpt provides an example of the required stanzas:

```
[oauth]

# This stanza contains definitions for OAuth specific information.

# ....

[tfim-cluster:oauth-cluster]

#
# This stanza contains definitions for the cluster of authorization
# servers that hosts the OAuth service.
#

# ....
```

For details of the required configuration entries for each of these stanzas, see the [aznapi-external-authzn-services] stanza, [oauth] stanza, and [oauth-eas] stanza documentation in the Web Reverse Proxy Stanza Reference topics.

Error responses:

In some circumstances HTTP error responses need to be returned to the client including:

- 400 Bad Request
- 401 Unauthorized
- 502 Bad Gateway

In the case of a 401 response, an additional `WWW-Authenticate` header is added to the response in the following format:

```
WWW-Authenticate: OAuth realm = <realm-name>
```

The HTML component of the responses are pre-loaded from files that have been specified in the EAS configuration. Namely the **[bad-request-rsp-file]**,**[unauthorized-rsp-file]** and **[bad-gateway-rsp-file]** configuration entries in the **[oauth-eas]** stanza. For more information about the **[oauth-eas]** stanza, see the Web Reverse Proxy Stanza Reference topics.

Troubleshooting:

The EAS provides trace information through the standard Security Access Manager tracing mechanism.

This mechanism is controlled using the Security Access Manager server task command: **trace**. You can use the **trace-component** configuration entry within the **[oauth-eas]** stanza to specify the name of the of the trace component that is associated with the EAS.

Using credential attributes in authorization decisions

The protected object policy (POP) attribute, `requires`, can be used to define rules which extends the authorization decision based on the contents of credential attributes.

The `requires` POP attribute is used to determine whether access will be granted to a requested resource. When an Access Control List (ACL) check for a resource succeeds, the POP is checked. If a POP with the `requires` attribute exists, and the attributes contained within the credential do not match the supplied attribute authorization rules, the request is denied.

Rule Format

The format of the rule is: `"<cred attr name>=<cred attr value> { OR <cred attr name>=<cred attr value>} { OR ... }"`.

Note: The credential attribute value must be surrounded by single or double quotes if the value contains any spaces.

A single condition within the attribute must match in order for the rule to pass. For example:

```
requires: SCOPE='usr:write' OR SCOPE='usr:admin' OR  AZN_CRED_AUTH_METHOD='password'
```

In this example, in order for the authorization decision to succeed, the user credential must contain a scope with `'usr:write'` or `'usr:admin'` or must contain an authentication method of `'password'`.

The `requires` attribute can be specified multiple times. The evaluation of each individual rule must pass in order for access to be granted. In other words, each instance of the `requires` attribute is a separate ‘and’ condition. For example, if 2 `requires` attributes are added to the POP:

```
requires: SCOPE='usr:write' OR SCOPE='usr:admin'
requires: AUTHENTICATION_LEVEL='2'
```

In this example, in order for the authorization decision to succeed, the user credential must contain a scope with a value of `'usr:write'` or `'usr:admin'` AND must also have an authentication level of 2.

Creating a policy:

Follow the instructions to add attribute checking to an authorization decision:

Procedure

1. A POP must be created and attached to the relevant object in the object space.
2. An attribute, with the name of 'eas-trigger' and the value of 'trigger_attr_eas', should be added to the POP to trigger the attribute-based authorization decision.
3. Attribute rules, with the name of requires, should be added to the POP. For example,

```
pdadmin> pop create attr-pop
pdadmin> pop attach /WebSEAL/ibm.com-default/junction_a attr-pop
pdadmin> pop modify attr-pop set attribute eas-trigger trigger_attr_eas
pdadmin> pop modify attr-pop set attribute requires "SCOPE='usr:write' OR SCOPE='usr:admin'"
pdadmin> pop modify attr-pop set attribute requires "AUTHENTICATION_LEVEL=2"
```

Troubleshooting:

Trace information for the rule evaluation is provided through the standard Security Access Manager tracing mechanism.

This mechanism is controlled using the Security Access Manager server task command: trace. The following trace components are useful when you are authoring and debugging attribute authorization rules:

Component	Description
pdweb.wan.azn	This trace component can be used to trace the overall authorization processing.
pdweb.azn.attr	This trace component can be used to trace the evaluation of attribute authorization rules.

Key management

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

Topic Index:

Key management overview

The LMI manages the keys that are required to enable SSL communication between WebSEAL and other components of the Security Access Manager domain.

You can use the LMI to create key database files and manage the digital certificates that are stored in these key database files.

Figure 26 on page 366 summarizes the key management configuration that WebSEAL uses for SSL communication with other components of the Security Access Manager environment. The configuration stanzas and stanza entries are in the WebSEAL configuration file.

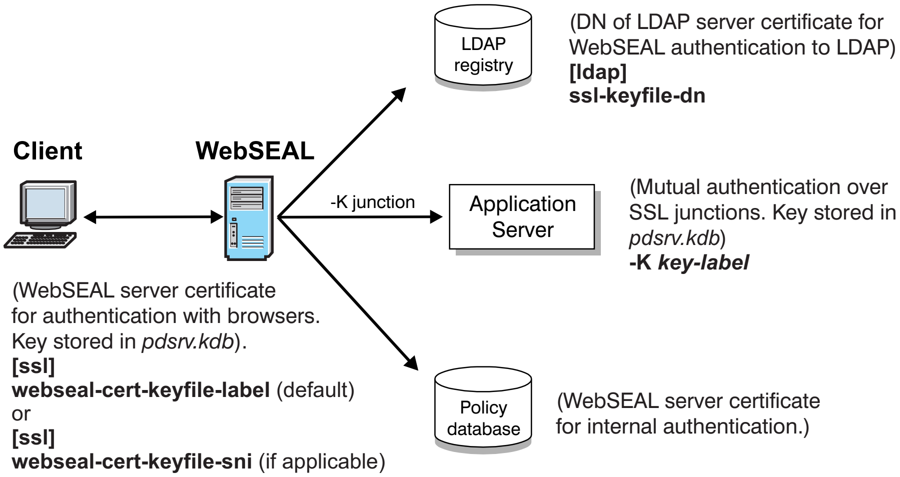

LDAP registry
(DN of LDAP server certificate for WebSEAL authentication to LDAP)
[ldap]
ssl-keyfile-dn
Client
WebSEAL
-K junction
Application Server
(Mutual authentication over SSL junctions. Key stored in *pdsrv.kdb*)
-K *key-label*
(WebSEAL server certificate for authentication with browsers. Key stored in *pdsrv.kdb*).
[ssl]
webseal-cert-keyfile-label (default)
or
[ssl]
webseal-cert-keyfile-sni (if applicable)
Policy database
(WebSEAL server certificate for internal authentication.)

Figure 26. Keyfile management configuration

Key management in the Local Management Interface

You can use the LMI to manage the digital certificates that WebSEAL uses. In the LMI, go to the **Secure Web Settings** > **Global Keys** menu to access the following key management pages:

SSL Certificates

Use the SSL Certificates management page to complete the following tasks:

- List or retrieve all current SSL certificate database names.
- Create a certificate database.
- Rename a certificate database.
- Describe a certificate database.
- Delete a certificate database.
- Import a certificate database.
- Export a certificate database.
- Manage signer certificates.
- Manage personal certificates.
- Manage certificate requests.

SSO Keys

Use the SSO Keys management page to complete the following tasks:

- List all current SSO key files.
- Create a new SSO key file.
- Import an existing SSO key file.
- Export an SSO key file.
- Delete an SSO key file.

LTPA Keys

Use the LTPA Keys management page to complete the following tasks:

- Retrieve all current LTPA key files.
- Rename an LTPA key file.
- Delete an LTPA key file.
- Export an LTPA key file.

- Import an LTPA key file.

See the Appliance administration topics for detailed information about using the LMI to complete these key management tasks.

Client-side and server-side certificate concepts

This section describes the administration and configuration tasks required to set up WebSEAL to handle client-side and server-side digital certificates used for authentication over SSL.

WebSEAL requires certificates for the following situations:

- WebSEAL identifies itself to SSL clients with its server-side certificate
- WebSEAL identifies itself to a junctioned back-end server (configured for mutual authentication) with a client-side certificate
- WebSEAL refers to its database of Certificate Authority (CA) root certificates to validate clients accessing with client-side certificates
- WebSEAL refers to its database of Certificate Authority (CA) root certificates to validate junctioned back-end servers

WebSEAL uses the IBM Global Security Kit (GSKit) implementation of SSL to configure and administer digital certificates. The appliance provides the LMI to set up and manage the certificate key database. This database contains one or more WebSEAL server/client certificates and the CA root certificates.

WebSEAL includes the following components at installation to support SSL authentication using digital certificates:

- A default key database (`pdsrv.kdb`)
- A default key database stash file (`pdsrv.sth`) and password ("pdsrv")
- Several common CA root certificates
- A self-signed test certificate that WebSEAL can use to identify itself to SSL clients

  Before using WebSEAL in a production environment, apply for a commonly recognized certificate from a known Certificate Authority to use instead of this test certificate.

Configuration of the WebSEAL key database file

This section contains the following topics:

WebSEAL key database file

During installation, WebSEAL provides a default certificate key database that is used to authenticate both clients and junctioned servers. WebSEAL also provides an optional, separate certificate key database that can be used to authenticate junctioned servers.

By default, the junction certificate key database option is commented out in the WebSEAL configuration file. Unless this option is enabled, junctions maintain the default behavior of using a shared key database for clients and junctioned servers.

Note: When a separate certificate key database is used for junctioned servers, it is not possible for a user to use a client certificate that is validated by a CA certificate stored in the junction key database. Similarly, it is not possible for a junctioned server to use a certificate that is validated by a CA certificate contained in the default certificate database.

The **webseal-cert-keyfile** stanza entry, located in the **[ssl]** stanza of the WebSEAL configuration file, identifies the default certificate key database. For example:

```
[ssl]
webseal-cert-keyfile = pdsrv.kdb
```

The **jct-cert-keyfile** stanza entry in the **[junction]** stanza for the WebSEAL configuration file, identifies the optional, separate junction certificate key database. For example:

```
[junction]
jct-cert-keyfile = pdjct.kdb
```

You can use the SSL Certificates management page of the LMI to create a new key database. However, you must enter the name and location of this new key file in the **webseal-cert-keyfile** stanza entry so that WebSEAL can find and use the certificates contained in that database.

Key database file password

WebSEAL provides a stash file that contains the password for the default certificate key database pdsrv.kdb.

The following stanza entry specifies the name of the stash file:

```
[ssl]
webseal-cert-keyfile-stash = pdsrv.sth
```

Random passwords can be used for all keystores in the pdsrv.sth stash file.

During installation, WebSEAL uses the stash file to obtain the key file password.

WebSEAL test certificate

During installation, WebSEAL provides a non-secure self-signed test certificate. The test certificate, acting as a server-side certificate, allows WebSEAL to identify itself to SSL clients.

To better control how this test certificate is used, the certificate is not installed as a default certificate. Instead, the **webseal-cert-keyfile-label** stanza entry designates the certificate as the active server-side certificate and overrides any other certificate designated as "default" in the keyfile database.

```
[ssl]
webseal-cert-keyfile-label = WebSEAL-Test-Only
```

Note: WebSEAL uses GSKit certificate handling functionality. GSKit allows but does not require that a certificate in keyfile databases be designated the default certificate.

Although this test certificate allows WebSEAL to respond to an SSL-enabled browser request, it cannot be verified by the browser (which does not contain an appropriate root CA certificate). Because the private key for this default certificate is contained in every WebSEAL distribution, this certificate offers no true secure communication.

You can use the LMI to generate a certificate request that can be sent to a Certificate Authority (CA). Use the LMI to install and label the returned server certificate.

If you use different certificates for other scenarios (such as **–K** junctions), you can use the LMI to create, install, and label these certificates. The keyfile label must not contain spaces.

WebSEAL (which by default runs as **user ivmgr**) must have read (r) permission on these key database files.

Server Name Indication

WebSEAL can use Server Name Indication to identify the host name in the request and send a server certificate that contains a matching host name. You can configure the certificate that WebSEAL uses for each host.

Server Name Indication is an extension to the SSL and TLS protocols. Server Name Indication identifies the host name to which the browser is requesting a connection.

By default, WebSEAL sends the same certificate to all hosts. However, by using Server Name Indication, WebSEAL can send a different certificate for each requested host.

To support Server Name Indication, the request must meet the following requirements:

- Use TLS over SSL to connect to WebSEAL. SSLv2 and SSLv3 are not supported.
- Use a browser that supports Server Name Indication.

Use the **webseal-cert-keyfile-sni** configuration entry in the **[ssl]** stanza of the WebSEAL configuration file to specify the certificate that WebSEAL sends for a particular host name. For example:

```
[ssl]
webseal-cert-keyfile-sni = <host_name>:<label>
```

where:

<host_name>
The name of the host to which WebSEAL returns the certificate.

<label>
The name of the certificate for WebSEAL to use.

Note: Specify the certificate that contains a **dn** value of `cn=`*<host_name>*.

You can specify this configuration entry multiple times. Specify a separate entry for each server certificate.

If WebSEAL does not find an entry for the host name in the browser request, WebSEAL sends the default certificate that is specified by the **webseal-cert-keyfile-label** entry. WebSEAL also uses the default certificate if the request does not meet the Server Name Indication requirements. For example, if the browser does not support Server Name Indication.

If you do not configure **webseal-cert-keyfile-sni** entries, WebSEAL can send only a single certificate, which means that WebSEAL cannot differentiate between different hosts. A certificate mismatch error results in the browser when a user uses SSL to connect to a host that does not match the default certificate.

Server Name Indication solves this problem. Use the **webseal-cert-keyfile-sni** to configure WebSEAL to provide a matching certificate for each host name.

Certificate revocation in WebSEAL

Certificates can be revoked for various reasons. Before using a certificate, WebSEAL must check an up-to-date source to ensure that it is valid.

Certificate revocation list (CRL)

The certificate revocation list (CRL) is a method of preventing the validation of unwanted certificates. The CRL contains the identities of certificates that are deemed untrustworthy. WebSEAL uses a GSKit implementation of SSL that supports CRL checking. WebSEAL can use GSKit to perform CRL checking on client-side certificates and certificates from SSL junctions.

A certificate authority (CA) provides a CRL that is valid for a limited amount of time. The CA specifies the lifetime validity of the CRL. The CA is responsible for maintaining this information. Contact the CA to find out their policies for updating the CRL.

You can configure WebSEAL to use OCSP, CRL, or both for managing certificates. By default, WebSEAL (using GSKit) tries OCSP first, followed by CRL. If these first two methods fail, WebSEAL can then try LDAP (if configured). This search order is defined by an RFC and cannot be changed.

WebSEAL must be able to connect to the CRL Distribution Point (CDP) as specified by the CA in the certificate. If WebSEAL is installed on a server behind a firewall, you must allow communication through to the CDP. Otherwise, performance could be affected and you risk certificates being validated against an out of date CRL.

There is no time limitation for using an outdated CRL. However, allowing the use of an outdated CRL creates security exposures. If GSKit determines that the CRL is out of date, it returns an UNDETERMINED status message. The application can then decide the best course of action. You can configure the course of action in WebSEAL by setting the configuration option **undetermined-revocation-cert-action** in the **[ssl]** stanza to one of: `ignore`, `log`, or `reject`.

Configuration of CRL checking

WebSEAL must know the location of the CRL list in order to perform CRL checking. Stanza entries for the location of the LDAP server that can be referenced for CRL checking during client-side certificate authentication are found in the **[ssl]** stanza of the WebSEAL configuration file:

```
[ssl]
#crl-ldap-server = server-name
#crl-ldap-server-port = port-id
#crl-ldap-user = webseal-admin-name
#crl-ldap-user-password = admin-password
```

Stanza entries for the location of the LDAP server that can be referenced for CRL checking during authentication across SSL junctions are found in the **[junction]** stanza of the WebSEAL configuration file:

```
[junction]
#crl-ldap-server = server-name
#crl-ldap-server-port = port-id
#crl-ldap-user = webseal-admin-name
#crl-ldap-user-password = admin-password
```

By default, CRL checking is disabled (stanza entries are commented out). To enable CRL checking during certificate authentication, uncomment each stanza entry and enter the appropriate values.

A null value for the **crl-ldap-user** stanza entry indicates that the SSL authentication mechanism should bind to the LDAP server as an anonymous user.

CRL distribution points

A CA specifies in the certificate where you can obtain revocation information. These details are not provided by WebSEAL or the GSKit library.

Although rare, a certificate can have more than one CDP. The primary reason for more than one CDP is to offer different protocols such as LDAP and HTTP. If a certificate is configured with more one CDP, WebSEAL contacts each CDP until a valid result is returned.

You can use Certificates from different CAs. Each CRL is signed by each CA so they cannot be confused. Each certificate contains its own CDP.

Configuration of the CRL cache

GSKit allows WebSEAL to perform CRL checking on client-side certificates and certificates from SSL junctions. To improve CRL checking performance, you can cache the CRL from a particular Certificate Authority (CA). Subsequent CRL checks are made against this cached version of the list.

The settings for the two configuration file stanza entries discussed in this section are passed directly to the GSKit utility. For further information about GSKit functionality, refer to the GSKit documentation.

Set the maximum number of cache entries

The **gsk-crl-cache-size** stanza entry specifies the maximum number of entries in the GSKit CRL cache. Each entry represents an entire CRL for a particular certificate authority. The default setting is "0". A value greater than "0" is required to activate the cache.

Note: CRL entries can use a large amount of memory. Therefore, try to specify the minimal value for the **gsk-crl-cache-size**.

```
[ssl]
gsk-crl-cache-size = 0
```

Set the GSKit cache lifetime timeout value

The **gsk-crl-cache-entry-lifetime** stanza entry specifies the lifetime timeout value for all entries in the GSKit CRL cache. The value is expressed in seconds and can have a range of 0-86400 seconds. The default value is `0`.

Note: There is no maximum limit imposed by either WebSEAL or GSKit, but the value must be contained in a 64-bit integer.

```
[ssl]
gsk-crl-cache-entry-lifetime = 0
```

Enable the CRL cache

When the **gsk-crl-cache-size** and **gsk-crl-cache-entry-lifetime** stanza entries are both set to "0" (default), CRL caching is disabled.

To enable the cache, change the setting for either or both of the **gsk-crl-cache-size** and **gsk-crl-cache-entry-lifetime** to a value other than zero. If both values are zero, the cache is disabled. The cache is enabled if one or both of these stanza entries has a non-zero value configured.

If either configuration entry has a value of `0` while the other is non-zero, GSKit automatically assigns a default value to the entry with the zero value. GSKit uses the following process:

- If **gsk-crl-cache-entry-lifetime** is configured with a non-zero value, but **gsk-crl-cache-size** is configured as `0` then the CRL cache is enabled. In this case, GSKit uses the following default value for the **gsk-crl-cache-size**:
  - **gsk-crl-cache-size** = `50`
- If **gsk-crl-cache-size** is configured with a non-zero value, but **gsk-crl-cache-entry-lifetime** is configured as `0` then the CRL cache is enabled. In this case, GSKit uses the following default value for the **gsk-crl-cache-entry-lifetime**:
  - **gsk-crl-cache-entry-lifetime** = 43200

If the CDP in the certificate specifies an HTTP source for the CRL then WebSEAL does not use the **gsk-crl-cache-size** and **gsk-crl-cache-entry-lifetime** configuration settings. CRLs from HTTP sources are never cached. If OCSP is not an option and a large CRL must be read using HTTP, you can use the GSKit environment variable **GSK_HTTP_CDP_MAX_RESPONSE_SIZE**.

Use of the WebSEAL test certificate for SSL connections

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

In order to enable the establishment of SSL sessions on a new WebSEAL server, WebSEAL contains a self-signed test server certificate. WebSEAL can present the self-signed certificate to the client. If the client accepts the certificate, the SSL session is established.

This test certificate is not suitable for permanent use by the WebSEAL server. Although this test certificate allows WebSEAL to respond to an SSL-enabled browser request, it cannot be verified by the browser. This is because the browser does not contain an appropriate root Certificate Authority (CA) certificate — as is the case for when the browser receives any self-signed certificate for which a root CA certificate does not exist. Because the private key for this default certificate is contained in every WebSEAL distribution, this certificate offers no true secure communication.

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

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

If you require different certificates for other scenarios (such as for mutually authenticated junctions), you can use the LMI to create, install, and label these additional certificates. See "Configuration of the WebSEAL key database file" on page 367.

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

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

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

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

Instructions for setting values that affect CRL access and handling, including valid ranges for cache settings, are in the Web Reverse Proxy Stanza Reference topics.

See also "Configuration of the CRL cache" on page 371.

Chapter 7. Standard WebSEAL Junctions

Standard WebSEAL junctions

This chapter provides information for configuring standard WebSEAL junctions.

Most standard junction options are also supported by virtual host junctions.

Topic Index:

WebSEAL junctions overview

A WebSEAL junction is an HTTP or HTTPS connection between a front-end WebSEAL server and a back-end Web application server. Junctions logically combine the Web space of the back-end server with the Web space of the WebSEAL server, resulting in a unified view of the entire Web object space.

A junction allows WebSEAL to provide protective services on behalf of the back-end server. WebSEAL performs authentication and authorization checks on all requests for resources before passing those requests across a junction to the back-end server. Junctions also allow a variety of single signon solutions between a client and the junctioned back-end applications.

You can create WebSEAL junctions with either the **pdadmin** command-line utility or the LMI.

Junction types

You can create the following WebSEAL junction types:

- WebSEAL to back-end server over TCP connection
- WebSEAL to back-end server over TCP connection using HTTP proxy server
- WebSEAL to back-end server over SSL connection
- WebSEAL to back-end server over SSL connection using HTTPS proxy server
- WebSEAL to WebSEAL over SSL connection
- WebSEAL to back-end server over mutual junction

You must address the following two concerns when creating any junction:

1. Decide where to junction (mount) the Web application server in the WebSEAL object space.
2. Choose the type of junction.

Applying coarse-grained access control: summary

About this task

A protective ACL placed on the junction object provides coarse-grained control over the back-end resources. The ACL provides a general overall coarse-grained set of permissions every individual resource accessed through the junction.

Procedure

1. Use the **pdadmin** utility to create a junction between WebSEAL and the back-end server.
2. Place an appropriate ACL policy on the junction point to provide coarse-grained control to the back-end server.

Applying fine-grained access control: summary

A protective ACL placed on the junction object provides coarse-grained control over the back-end resources. The ACL provides a general overall coarse-grained set of permissions every individual resource accessed through the junction.

About this task

You can also provide fine-grained protection to the resources accessed through the junction by explicitly placing ACLs on individual resource objects or groups of objects. WebSEAL cannot automatically see and understand a back-end file system. You must inform WebSEAL of the back-end object space using a special application, called **query_contents**, that inventories the back-end Web space and reports the structure and contents to WebSEAL.

Procedure

1. Use the **pdadmin** utility to create a junction between WebSEAL and the back-end server.
2. Copy the **query_contents** program to the back-end server.
3. Apply ACL policy to appropriate objects in the object space revealed by the **query_contents** program.

Additional references for WebSEAL junctions

See “Standard WebSEAL junctions” on page 11 for a conceptual overview of WebSEAL junctions.

See “Advanced junction configuration” on page 393 for advanced junction options.

See “Command option summary: standard junctions” on page 469 for a summary of the junction command options by functional categories.

Junction management in the Local Management Interface

You can use the LMI to manage standard and virtual junctions. To access the junction management page in the LMI, go to the Reverse Proxy Management page and select the appropriate WebSEAL instance. Click **Manage** > **Junction Management** to open the Junction Management page for the selected instance.

From the Junction Management page, you can complete the following tasks:

- Retrieve a list of standard and virtual junctions.
- Retrieve the parameters for a single standard or virtual junction.
- Delete a standard or virtual junction.
- Create a standard or virtual junction.
- Add a back-end server to an existing standard or virtual junction.

See the *IBM Security Web Gateway appliance: Administration Guide* for detailed information about how to use the LMI to complete these junction management tasks.

Managing junctions with the pdadmin utility

About this task

Before using the **pdadmin** utility, you must login to a secure domain as a user with administration authorization, such as **sec_master**.

For example:

UNIX or Linux:

```
# pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
pdadmin>
```

To create WebSEAL junctions, you use the **pdadmin server task create** command:

```
pdadmin> server task instance_name-webseald-host_name create options
```

For example, if the configured name of a single WebSEAL instance is **web1** , installed on a host named **www.example.com**, the complete server name would be expressed as follows:

```
web1-webseald-www.example.com
```

Use the **pdadmin server list** command to display the correct format of the complete server name:

```
pdadmin> server list
web1-webseald-www.example.com
```

For more information, see the reference page for **pdadmin server task create**, "The pdadmin server task create command."

Standard WebSEAL junction configuration

This section contains the following topics:

The pdadmin server task create command

WebSEAL supports both standard non-secure TCP (HTTP) and secure SSL (HTTPS) junctions between WebSEAL and back-end Web application servers.

The junction between WebSEAL and the back-end server is independent of the type of connection (and its level of security) between the client (browser) and the WebSEAL server.

The mandatory command options required to create a basic WebSEAL junction using **pdadmin server task create** include:

- Host name of the back-end application server (**–h** option)
- Junction type: **tcp**, **ssl**, **tcpproxy**, **sslproxy**, **local** (**–t** option)
- Junction point (mount point)

Command syntax (entered as one line):

```
pdadmin> server task instance_name-webseald-host-name
 create -t type -h host_name jct_point
```

For example:

```
pdadmin> server task web1-webseald-cruz create -t tcp -h doc.ibm.com /pubs
```

Note: Always use the fully qualified domain name of the back-end server when specifying the argument to the **–h** option.

WebSEAL also supports HTTP/2 connections. To create an HTTP/2 junction, use the **-o** option. The following parameters are available:

h2 Enable HTTP/2 protocol to junction server.

pch2c Enable HTTP/2 protocol to proxy server.

sni=<hostname>
Send the string <hostname> in the SSL/TLS handshake as the Server Name Indicator(SNI) value.

Creating TCP type standard junctions

About this task

A WebSEAL junction over a TCP connection provides the basic properties of a junction but does not provide secure communication across the junction.

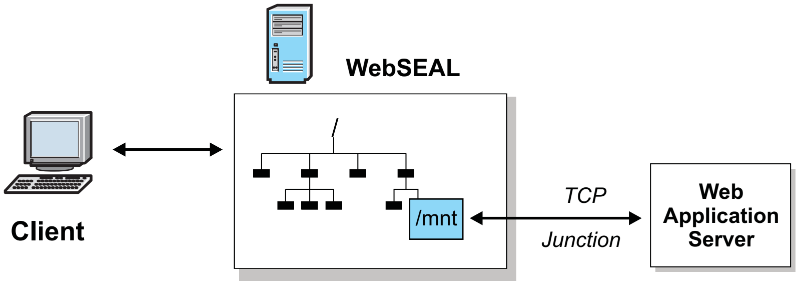


Figure 27. Non-secure TCP (HTTP) junction

Procedure

To create a secure TCP junction and add an initial server, use the **create** command with the **–t tcp** option (entered as one line):

```
pdadmin> server task instance_name-webseald-host_name create -t tcp
-h host-name [-p port] jct-point
```

The default port value for a TCP junction (if not specified) is **80**.

Creating SSL type standard junctions

About this task

SSL junctions function exactly like TCP junctions, with the added value that all communication between WebSEAL and the back-end server is encrypted.

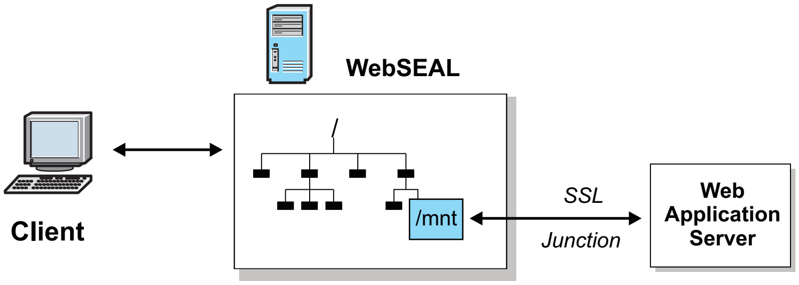


Figure 28. Secure SSL (HTTPS) junction

SSL junctions allow secure end-to-end browser-to-application transactions. You can use SSL to secure communications from the client to WebSEAL and from WebSEAL to the back-end server. The back-end server must be HTTPS-enabled when you use an SSL junction.

Procedure

To create a secure SSL junction and add an initial server, use the **create** command with the **–t ssl** option (entered as one line):

```
pdadmin> server task instance_name-webseald-host-name create -t ssl
-h host_name [-p port] jct_point
```

The default port value for an SSL junction (if not specified) is **443**.
For more information on configuring SSL-based standard junctions, see "SSL-based standard junctions" on page 380.

Creating mutual junctions

About this task

Mutual junctions provide the ability to send junction requests over HTTP or HTTPS, governed by the communication protocol over which the request was received.

If a request comes in to a mutual junction over HTTP, then the request goes to the junctioned server via HTTP. If the request comes in over HTTPS then it goes to the junctioned server over HTTPS.

Procedure

To create a mutual junction and add an initial server, use the **create** command with the **–t mutual** option. Use the **-p** option for a HTTP port and the **-P** option for a HTTPS port. Similarly, use the **-V** option to specify the virtual host name for HTTP requests, and the **-V** option to specify the virtual host name for HTTPS requests. For example (entered as one line):

```
pdadmin> server task instance_name-webseald-host-name create -t mutual
-h host_name [-p HTTP_port] [-P HTTPS_port] [-v HTTP_virtual_host_name]
[-V HTTPS_virtual_host_name]jct_point
```

The default HTTP port value for a mutual junction (if not specified) is **80**.
The default HTTPS port value for a mutual junction (if not specified) is **443**.

SSL-based standard junctions

Verification of the back-end server certificate:

When a client makes a request for a resource on the back-end server, WebSEAL, in its role as a security server, performs the request on behalf of the client. The SSL protocol specifies that when a request is made to the back-end server, that server must provide proof of its identity using a server-side certificate.

When WebSEAL receives this certificate from the back-end server, it must verify its authenticity by matching the certificate against a list of root CA certificates stored in its certificate database.

Security Access Manager uses the IBM Global Security Kit (GSKit) implementation of SSL. You can use the LMI to add the root certificate of the CA who signed the back-end server certificate to the WebSEAL certificate keyfile (`pdsrv.kdb`).

Examples of SSL junctions:

Junction host **sales.ibm.com** at junction point `/sales` over SSL (entered as one line):

```
pdadmin> server task web1-webseald-cruz  create -t ssl -h
sales.ibm.com /sales
```

Note: In this sales example, the **–t ssl** option dictates a default port of 443.

Junction host **travel.ibm.com** on port **4443** at junction point `/travel` over SSL (entered as one line):

```
pdadmin> server task web1-webseald-cruz create -t ssl -p 4443
-h travel.ibm.com /travel
```

Disabling SSL protocol versions for junctions:

About this task

You can optionally disable one or more SSL protocol versions for junction connections. By default, SSL v2, SSL v3, and TLS v1.3 are disabled. All other supported SSL versions are enabled. The WebSEAL configuration file provides the following entries by default:

```
[junction]
disable-ssl-v2 = yes
disable-ssl-v3 = yes
disable-tls-v1 = no
disable-tls-v11 = no
disable-tls-v12 = no
disable-tls-v13 = yes
```

Note: When TLS version 1.3 is enabled, SSLv2 and SSLv3 are disabled regardless of their configuration in accordance with RFC 8446 The Transport Layer Security (TLS) Protocol Version 1.3.

Procedure

To disable an SSL protocol version for junctions, set the corresponding entry to `yes`.

Adding multiple back-end servers to a standard junction

About this task

See "Adding multiple back-end servers to the same junction" on page 385.

Local type standard junction

A local type junction (**-t local**) is a mount point for specific content located locally on the WebSEAL server. Like the content from junctioned remote servers, local junction content is incorporated into WebSEAL's unified protected object space view.

The following junction options are appropriate for local type junctions:

Table 53. Local type junction options

Option	Description
–t *type*	Type of junction (**local**).
–f	Force the replacement of an existing junction. See "Forcing a new junction" on page 401.
–l *percent-value*	Defines the soft limit for consumption of worker threads.
–L *percent-value*	Defines the hard limit for consumption of worker threads.

Disable local junctions

Disable the local junction functionality so that the WebSEAL instance cannot serve locally stored pages.

Set the `disable-local-junctions` entry in the `[junction]` stanza of the WebSEAL configuration file to `yes` to disable the local junction functionality:

```
[junction]
disable-local-junctions = yes
```

If you enable the `disable-local-junctions` configuration item, new local junctions are not created. If existing local junctions are in the WebSEAL instance, those junctions are not loaded when the instance starts.

Transparent path junctions

This section contains the following topics:

Filtering concepts in standard WebSEAL junctions

In standard junction configuration, a configured junction represents a specific back-end host machine. The junction and its name are represented as a subdirectory in the WebSEAL protected object space.

The following example uses the **pdadmin server task** command to create a junction (`/jct`) to the back-end server `pubs.ibm.com`:

```
pdadmin> server task web1-webseald-www.cruz.com create -t tcp -h pubs.ibm.com /jct
```

A subdirectory named `/jct` is created in the WebSEAL object space. This junction mount point represents the back-end server `pubs.ibm.com`.

A response page returned from `pubs.ibm.com` contains the following link (URL) in the HTML of that page:

```
http://pubs.ibm.com/docs/readme.html
```

WebSEAL's standard filtering mechanism for standard junctions parses the HTML in the response page and modifies this link by adding the name of the junction configured for this back-end server. Additionally, the original absolute expression of the URL is changed to a server-relative expression. This is the link as it now appears to the user:

```
/jct/docs/readme.html
```

Note: If **rewrite-absolute-with-absolute** was set to "yes", the link would appear as:

```
http://www.cruz.com/jct/docs/readme.html
```

See “Configuring the rewrite-absolute-with-absolute option” on page 431.

Now the user clicks the link to access the back-end resource (`readme.html`).

The portion of the URL representing the junction name (`/jct`) is mapped by WebSEAL to the back-end server, `pubs.ibm.com`. By design, a junction points to the **root** of the back-end server document space. Any path expression following the junction name in the URL represents the path to the resource from the server's **root** location.

As a conclusion to the example, WebSEAL successfully locates the resource at:

```
http://pubs.ibm.com/docs/readme.html
```

Notice that in the end, the junction name has been removed from the path and the URL reads as it originally appeared on the response page.

Transparent path junction concepts

For standard WebSEAL junctions, a link to a resource on a back-end junctioned server can only succeed if the URL in the request received by WebSEAL contains the identity of the junction. Junctions use both default and optional filtering solutions to force URLs found in HTML response pages to appear correct when viewed as a part of WebSEAL's single host document space. See “Modification of URLs to junctioned resources” on page 419.

There are three parts of a URL that must be considered in a filtering solution:

- protocol
- host name:port
- path

Of the three parts of a URL, the path is often the most problematic and restricting part to filter. The transparent path junction option (**-x**) implements a variation on the standard junction mechanism that eliminates the need to filter the path portion of a URL.

A transparent path junction observes a crucial requirement: the configured junction name must match the name of a subdirectory under the **root** of the back-end server document space. All resources accessed through this junction must be located under this subdirectory. The transparent path junction name represents the name of the actual subdirectory on the back-end server.

Transparent path junctions are really the same as standard junctions except that the junction name, instead of being an addition to the URL path, is based on the path already present on the back-end application. Transparent path junctions allow WebSEAL to route requests to a junction based on the URL path of the back-end server resources rather than based on a junction name added to the path.

For example, if the configured junction name is `/docs`, all resources controlled by this junction must be located on the back-end server under a subdirectory called `/docs`.

The transparent path junction mechanism prevents WebSEAL from filtering the path portion of links to the resources protected by this junction. The junction name has now become part of the actual path expression describing the location of a resource and no longer requires filtering. The junction name is not added to or removed from the path portion of URLs, as it is in junctions created without the transparent path option.

WebSEAL does support nested paths. For example, the following three junctions are all valid and can be made on the same WebSEAL system:

```
/financing/tools
/financing/tools/gars
/financing/tools/gars/custom
```

The pattern-matching within WebSEAL is sensitive enough to map to the most "specific" junction first, `/financing/tools/gars/custom` in this example.

Configuring transparent path junctions

About this task

To configure a transparent path junction:

Procedure

1. Use the **pdadmin server task** command to create the transparent path junction. Create the junction like you would a standard WebSEAL junction, but include the **-x** option. In the following example (entered as one line), the junction name is `/files/docs`:`pdadmin> server task web1-webseald-www.cruz.com create -t tcp -x-h pubs.ibm.com /files/docs`
2. Ensure that all resources protected by this junction are contained on the back-end server (in this example, `pubs.ibm.com`) under a subdirectory called `/files/docs/`.

Results

The junction name (and its associated subdirectory on the back-end server) must be unique among all other protected servers in the WebSEAL environment.

Example transparent path junction

The following example uses the **pdadmin** command (entered as one line) to create a transparent path junction (`/docs`) to the back-end server `pubs.ibm.com`:

```
pdadmin> server task web1-webseald-www.cruz.com create -t tcp -x
-h pubs.ibm.com /docs
```

After a client request for a back-end resource is made (via the WebSEAL proxy server), a response page is returned from `pubs.ibm.com` containing the following link (URL) in the HTML of that page:

```
http://pubs.ibm.com/docs/readme.html
```

WebSEAL's standard filtering mechanism for junctions parses the HTML in the response page and modifies this link by changing the original absolute expression of the URL to a server-relative expression. However, the path is not filtered, because this is a transparent path junction (**-x**). This is the link as it now appears to the user:

```
/docs/readme.html
```

Note: If **rewrite-absolute-with-absolute** was set to "yes", the link would appear as:

```
http://www.cruz.com/docs/readme.html
```

See “Configuring the rewrite-absolute-with-absolute option” on page 431.

Now the user clicks the link to access the back-end resource (`readme.html`).

The portion of the URL representing the junction name (`/docs`) is recognized by WebSEAL as associated with the `/docs` subdirectory on the back-end server, `pubs.ibm.com`.

As a conclusion to the example, WebSEAL successfully locates the resource at:

```
http://pubs.ibm.com/docs/readme.html
```

Some benefits of transparent path junctions include:

- Several different transparent path junctions to the same back-end server can be created to point to different regions (subdirectories) of that server.
- Each individual transparent path junction can handle a different authentication requirement and ACL control.

Technical notes for using WebSEAL junctions

This section contains the following topics:

Guidelines for creating WebSEAL junctions

The following guidelines summarize the "rules" for junctions:

- You can add a junction anywhere in the primary WebSEAL object space.
- You can junction multiple replica back-end servers at the same mount point.

  Multiple replica back-end servers mounted to the same junction point must be of the same type.
- ACL policies are inherited across junctions to back-end Web servers.
- The junction name should not match any directory name in the Web space of the back-end server if HTML pages from that server contain programs (such as JavaScript or applets) with server-relative URLs to that directory. For example, if pages from the back-end server contain programs with a URL of form `/path/...`, do not create a junction name using `/path`.
- Creating multiple WebSEAL junctions that point to the same back-end application server/port is not a secure junction configuration. Each junction can be control by unique ACLs. One junction secured with more permissive ACLs can compromise another junction secured with less permissive ACLs. This type

of configuration can cause unintended control of access to resources and is therefore not a supported configuration strategy for Security Access Manager.

- WebSEAL supports HTTP/1.1 and HTTP/2 across junctions.

Adding multiple back-end servers to the same junction

About this task

To increase high availability of the resources protected by Security Access Manager, you can junction multiple replica back-end servers to the same junction point. There can be any number of replica servers mounted at the same point.

- Multiple back-end servers added to the same junction point must be replica servers with identical (mirrored) Web document spaces.
- Multiple back-end servers added to the same junction point must use the same protocol.
- Do not add dissimilar servers to the same junction point.
- WebSEAL uses a least busy algorithm to determine which back-end replica server has the fewest number of request connections and forwards any new request to that server.

Procedure

1. Create the initial junction. For example:

   ```
   pdadmin> server task web1-webseald-cruz create -t tcp -h server1 /sales
   ```

2. Add an additional back-end server replica. For example:

   ```
   pdadmin> server task web1-webseald-cruz add -h server2 /sales
   ```

3. From the primary Security Access Manager server Web space, test the access to pages belonging to the junctioned servers. You must be able to access these pages (subject to permissions) and the pages must appear consistent. If a page cannot be found occasionally, or if it changes occasionally, it means that page was not replicated properly.
4. Check that the document exists and is identical in the document tree of both replicated servers.

Exceptions to enforcing permissions across junctions

Certain Security Access Manager permissions are not enforceable across a junction. You cannot control, for example, the execution of a CGI script with the **x** permission, or a directory listing with the **l** permission. WebSEAL has no means of accurately determining whether or not a requested object on a back-end server is, for example, a CGI program file, a dynamic directory listing, or a regular HTTP object.

Access to objects across junctions, including CGI programs and directory listings, is controlled only through the **r** permission.

Certificate authentication across junctions

At installation, WebSEAL is configured with a non-default test certificate. The test certificate is designated as the active server-side certificate by the **webseal-cert-keyfile-label** stanza entry in the **[ssl]** stanza of the WebSEAL configuration file.

If a junctioned back-end application server requires WebSEAL to identify itself with a client-side certificate, you must first create, install, and label this certificate using the Local Management Interface (LMI). Then, configure the junction using the **–K** ***key-label*** option. See "Mutually authenticated SSL junctions" on page 393.

If the junction is not configured with **–K**, GSKit handles a request for mutual authentication by automatically sending the "default" certificate contained in the keyfile database. If this is not the required response, you must ensure that there are no certificates marked as "default" (an asterisk mark) in the keyfile database (`pdsrv.kdb`, or the junctions keyfile, if a separate junction keyfile is configured).

In summary:

- Identify all required certificates by label name.
- Do not mark any certificate in the keyfile database as "default".
- Control the WebSEAL server-side certificate response with the **`webseal-cert-keyfile-label`** stanza entry.
- Control the WebSEAL client-side certificate response through the –K junction option.

Handling domain cookies

About this task

The **allow-backend-domain-cookies** stanza entry in the **[junction]** stanza of the WebSEAL configuration file controls how WebSEAL handles domain attributes in cookie headers.

When this stanza entry value is set to "no" (default), WebSEAL performs "tail matching" to determine if the domain (contained as an attribute in the cookie header) is valid. If the domain in the cookie header is valid, the cookie is sent to the browser with the domain attribute removed from the cookie header. When a browser receives a cookie with no domain attribute, it can return the cookie only to the originating server. If "tail matching" determines that the domain in the cookie header is not valid, the cookie is not sent to the browser. The browser has no cookies to return.

```
[junction]
allow-backend-domain-cookies = no
```

When this stanza entry value is set to "yes", WebSEAL does not perform "tail matching" and allows all cookies, regardless of the domain attribute value, to be sent to the browser. The browser can return the cookies to the appropriate server or servers.

```
[junction]
allow-backend-domain-cookies = yes
```

Procedure

Customize the **allow-backend-domain-cookies** configuration item for a particular junction by adding the adjusted configuration item to a **[junction:*{junction_name}*]** stanza.
{junction_name} refers to the junction point for a standard junction (including the leading / character) or the virtual host label for a virtual host junction.

Supported HTTP versions for requests and responses

HTTP/1.1 and HTTP/1.0

HTTP/1.0 requests are sent to junctioned back-end servers only if those servers return a status of 400 (Bad Request), return a status of 504 (HTTP version not supported), or if the client browser specifies HTTP/1.0 in the request.

Otherwise, if the back-end server accepts HTTP/1.1, WebSEAL sends HTTP/1.1 requests.

However, even when WebSEAL sends an HTTP/1.0 request to a junctioned back-end server (and the back-end server returns an HTTP/1.0 response), WebSEAL always returns an HTTP/1.1 response to the client browser.

HTTP/2

HTTP/2 support can be enabled on a per junction basis.

WebSEAL only uses HTTP/2 to junction servers or proxies that are used to access the junctions when the junctions are configured to use HTTP/2. It will not attempt to use or fall back to HTTP/1.1 or earlier versions.

For SSL junctions, the connection uses ALPN with only "h2". The junction server must accept the "h2".

For TCP junctions, the connection uses the "prior knowledge" method (RFC 7540 Section 3.4).

Junctioned application with Web Portal Manager

Problem: Web Portal Manager sends absolute or server-relative URLs in its Javascript. These addresses are not resolved successfully by the browser and require junction cookie information to complete the path name.

Solution: If an application server with Web Portal Manager is junctioned to WebSEAL, you must use the **–j** option when creating this junction. The junction cookie provided by the **–j** option allows the browser (client) to successfully issue commands to Web Portal Manager.

In addition to using the **–j** option, you must also use the **–c iv_user,iv_creds** option.

How to generate a back-end server Web space (query_contents)

This section contains the following topics:

query_contents overview

If you want to use the Security Access Manager security service to protect the resources on a back-end application Web server, you must provide WebSEAL with information about the contents of the back-end Web space.

A CGI program called **query_contents** provides this information. The **query_contents** program searches the back-end Web space contents and provides this inventory information to WebSEAL. A default version of the program comes with the WebSEAL installation, but must be manually installed on the back-end Web server. There are different program file types available, depending on whether the back-end server is running UNIX or Windows.

The Object Space manager of the Web Portal Manager automatically runs **query_contents** any time the portion of the protected object space belonging to the junction is expanded in the Object Space management panel. When the Web Portal Manager knows about the contents of the back-end Web space, you can display this information and apply policy templates to appropriate objects.

The pdadmin **object show** command uses the **query_contents** program to browse Web space and enables an administrator to identify resources to which a policy can be attached. In this way, pdadmin and **query_contents** are used to manage and control access to objects in the Security Access Manager object space.

WebSEAL includes source files for **query_contents**, a sample configuration file, and an HTML help file. Administrators can use these files to configure **query_contents** and, when needed, to modify its behavior.

Custom query_contents program:

When a **query_contents** program is run, WebSEAL sends a simple HTTP GET request to the program and expects a simple HTTP reply along with a response body containing the requested information. The **query_contents** program can be written in any language and can be hosted on any HTTP-compliant application server.

Program inputs:

The **query_contents** program is called with an HTTP GET request with the following syntax:

```
<URL>?dirlist=<resource root>
```

where <*URL*> is the location of the **query_contents** program and <*resource root*> is the root of the object space that should be searched.

The GET request is accompanied by an HTTP cookie with the name `X_QUERY_CONTENTS_URIENCODED`. This cookie contains a *yes* or *no* value to indicate whether the path is URI-encoded.

The following sample is an example of a GET request that WebSEAL might send to a **query_contents** program:

```
GET /appserver/cgi-bin/custom_query_contents.exe?dirlist=/directory
Cookie: X_QUERY_CONTENTS_URIENCODED=no
```

Program outputs:

The **query_contents** program must return a response that is UTF-8 encoded, and the response must include the following header:

```
content-type: text/plain;charset=utf8
```

The body of the response must contain one line with a return code followed by the directory listing, with one object or container name per line:

```
<return-code>
<file or directory>
<file or directory>
<file or directory>
```

where `<return-code>` is `100`, `102`, or `103`, as defined in the following table.

Table 54. Return codes

Query Status	Description
QUERY_STATUS_OK = 100	The directory listing operation succeeded.
QUERY_STATUS_NO_EXISTS = 102	The directory queried does not exist.
QUERY_STATUS_NOT_DIR = 103	The directory queried is not a directory.

Object names and containers in the response must follow these rules:

- Containers end in two forward slashes, for example,: `/path/directory//`
- Punctuation "." and ".." are not included in the directory entries.
- Object or container data must be UTF-8 encoded and then URI encoded.

query_contents components

The **query_contents** components are available through the LMI. Go to **Manage System Settings** > **Secure Settings** > **File Downloads**. Then, expand the **isam** directory.

The following table shows the **query_contents** components.

Table 55. Query_contents component descriptions

File	Description
`query_contents.exe` (Windows)	Main executable program for Windows systems. Install this file in the `cgi-bin` directory of the back-end Web server.
`query_contents.cfg` (Windows)	The Windows configuration file that identifies the document root for the Web server.
`query_contents.c` (Windows)	Source code for Windows systems. The source is provided in case you need to modify the behavior of **query_contents**. In most cases, this will not be necessary.
`query_contents.sh` (UNIX)	Main executable program for UNIX systems. Install this file in the `cgi-bin` directory of the back-end Web server.

Installing and configuring query_contents on UNIX-based Web servers

The following steps describe the installation of **query_contents** on back-end UNIX-based Web servers.

Procedure

1. In the LMI, go to **Secure Web Settings** > **Tools** > **Query Site Contents** to locate the shell script named `query_contents.sh`.
2. Ensure that the back-end Web server has a CGI directory correctly configured.
3. For testing purposes, ensure that a valid document exists in the document root of the back-end Web server.

4. Copy query_contents.sh into a functioning cgi-bin directory on the back-end Web server. Consult the appropriate documentation for the Web server to identify the location of this directory.
5. You must specify query_contents.sh as the correct file name for WebSEAL to use by using the **–q** *location* option when creating a junction to the back-end Web server.

   The **–q** *location* option and argument provides WebSEAL with the correct name of the file and where to find the file. For example (entered as one line and assuming other options are additionally selected):

   ```
   pdadmin> server task default-webseald-www.example.com create -t tcp -h
   host-name <...> -q /cgi-bin/query_contents.sh /junction-name
   ```

   The *location* argument value is used in the actual URL string that calls the **query_contents** program. For example:

   ```
   http://unix-machine-name/cgi-bin/query_contents.sh?dirlist=/
   ```
6. Manually edit the script file to correctly specify the document root directory.

   Change the value of the DOCROOTDIR variable (default is /usr/local/html) to the document root of the back-end Web server. For example (IBM HTTP Server, version 6.0):

   ```
   ADD_TO_ROOT=
   DOCROOTDIR=/opt/IBMIHS/htdocs
   #ADD_TO_ROOT="cgi-bin//"
   ```

   Note: The blank ADD_TO_ROOT line is required for the variable initialization.

   Additionally, if the back-end Web server's cgi-bin directory is not located below the document root on the Web server's file system, uncomment the line for the ADD_TO_ROOT variable. For example:

   ```
   ADD_TO_ROOT=
   DOCROOTDIR=/opt/IBMIHS/htdocs
   ADD_TO_ROOT="cgi-bin//"
   ```
7. Set the UNIX execute bit for the administration account of the back-end Web server.

Results

Testing the configuration (UNIX):

Procedure

1. From a command prompt on the back-end UNIX-based machine, execute the **query_contents** program from the CGI directory as follows:

   ```
   # query_contents dirlist=/
   ```

   You should see something similar to the following output:

   ```
   100
   index.html
   cgi-bin//
   pics//
   ```

   The number 100 is a return status that indicates success. It is most important to see at least the number 100 as the first (and perhaps only) value.

   If you see an error code instead, the document root entry might be incorrect. Check the configuration of the query_contents.sh file and make sure that the document root is specified correctly.
2. From a browser, enter the following URL:

   ```
   http://unix-machine-name/cgi-bin/query_contents.sh?dirlist=/
   ```

This URL should return the same result as the preceding step. If it does not return this result, the CGI configuration of your Web server is not correct. See the server's documentation to correct the problem.

Installing and configuring query_contents on Windows-based Web servers

The following steps describe the installation of **query_contents** on back-end Windows-based Web servers.

Procedure

1. Locate the executable program named `query_contents.exe` and the configuration file named `query_contents.cfg` in the LMI. Go to **Secure Web Settings** > **Tools** > **Query Site Contents**.
2. Ensure that the back-end Web server has a CGI directory correctly configured.
3. For testing purposes, ensure that a valid document exists in the document root of the back-end Web server.
4. Copy `query_contents.exe` into the `cgi-bin` directory of the back-end Web server. Consult the appropriate documentation for the Web server to identify the location of this directory.
5. Copy `query_contents.cfg` into the "Windows" directory.

   Default values for this directory are shown in the table below:

Operating System	Windows Directory
Windows	`c:\windows`

6. Edit the `query_contents.cfg` file to correctly specify the document root directory for the back-end Web server. The file contains a default value for the IBM HTTP Server (version 6.0):

   ```
   [server]
   docroot="C:/Program Files/IBM HTTP Server/htdocs/en_US"
   ```

7. You must specify `query_contents.exe` as the correct file name for WebSEAL to use by using the **–q** *location* option when creating a junction to the back-end Web server.

   The **–q** *location* option and argument provides WebSEAL with the correct name of the file and where to find the file. For example (entered as one line and assuming other options are additionally selected):

   ```
   pdadmin> server task default-webseald-www.example.com create -t tcp -h
     host-name <...> -q /cgi-bin/query_contents.exe \junction-name
   ```

   The *location* argument value is used in the actual URL string that calls the **query_contents** program. For example:

   ```
   http://windows-machine-name/cgi-bin/query_contents.exe?dirlist=/
   ```

Testing the configuration (Windows):

Procedure

1. From an MS-DOS prompt on the Windows machine, execute the **query_contents** program from the CGI directory as follows:

   ```
   MSDOS> query_contents dirlist=/
   ```

   You should see something similar to the following output:

   ```
   100
   index.html
   cgi-bin//
   pics//
   ```

The number 100 is a return status that indicates success. It is most important to see at least the number 100 as the first (and perhaps only) value.

If you see an error code instead, then the configuration file is not in the correct place, or does not contain a valid document root entry. Check the configuration of the `query_contents.cfg` file and make sure that the document root exists.

2. From a browser, enter the following URL:

   ```
   http://windows-machine-name/cgi-bin/query_contents.exe?dirlist=/
   ```

   This URL should return the same result as the preceding step. If it does not return this result, the CGI configuration of your Web server is not correct. See the server's documentation to correct the problem.

General process flow for query_contents

The job of **query_contents** is to return the contents of directories included in a URL request.

For example, to get the contents of the root directory of a server's Web space, the browser runs **query_contents** on a URL such as (UNIX example):

```
http://back-end-server/cgi-bin/query_contents.sh?dirlist=/
```

The **query_contents** script performs the following actions:

1. Reads the configured value for the **DOCROOTDIR** variable.
2. Reads the value of the **QUERY_STRING** variable from the requested URL to obtain the requested **operation** and get the **object path**.

   The operation value is stored in the **OPERATION** variable. The object path value is stored in the **$OBJPATH** variable. In the example, the value of the **OPERATION** variable is `dirlist`. The value of the **OBJPATH** variable is /.
3. Performs a directory listing (**DO**) on the object path and sends the results to standard output for use by the Security Access Manager server. Entries that indicate subdirectories have a double slash (//) appended to them.

   Typical output looks like:

   ```
   100
   index.html
   cgi-bin//
   pics//
   ```

   The number 100 is a return status that indicates success.

Securing the query_contents program

About this task

The **query_contents** CGI program is used by Security Access Manager to display junctioned Web server object spaces in the Web Portal Manager. It is very important to secure this file to prevent unauthorized users from running it.

Procedure

You must set a security policy that allows only the policy server (**pdmgrd**) identity to have access to the **query_contents** program. The following example ACL (**query_contents_acl**) meets this criteria:

```
group ivmgrd-servers Tl
```

```
user sec_master dbxTrlcam
```

Use the **pdadmin** utility to attach this ACL to the query_contents.sh (UNIX) or query_contents.exe (Windows) object on the junctioned servers. For example (UNIX):

```
pdadmin> acl attach /WebSEAL/host/junction-name/cgi-bin/query_contents.sh
query_contents_acl
```

Advanced junction configuration

Most standard junction options are also supported by virtual host junctions.

Topic Index:

Mutually authenticated SSL junctions

This section contains the following topics:

Mutually authenticated SSL junctions process summary

WebSEAL supports mutual authentication between a WebSEAL server and a back-end server over an SSL junction (**–t ssl** or **–t sslproxy** or **–t mutual**).

The following outline summarizes the supported functionality for mutual authentication over SSL (command options are listed where appropriate):

1. WebSEAL authenticates the back-end server (normal SSL process)
   - WebSEAL validates the server certificate from the back-end server.

     See "Validation of the back-end server certificate."
   - WebSEAL verifies the distinguished name (DN) contained in the certificate (**–D**) (optional, but provides a higher level of security).

     See "Matching the distinguished name (DN)" on page 394.
2. Back-end server authenticates WebSEAL (two methods)
   - Back-end server validates client certificate from WebSEAL (**–K**).

     See "Authentication with a client certificate" on page 394.
   - Back-end server validates WebSEAL identity information in a basic Authentication (BA) header (**–B**, **–U**, **–W**).

     See "Authentication with a BA header" on page 395.

The command options that control mutual authentication over SSL provide the following features:

- You can specify client certificate or BA authentication method.
- You can apply authentication methods on a per-junction basis.

Special considerations for combining the **–b** options (for handling BA information) with mutual authentication over SSL are described in "Client identity information across junctions" on page 520.

Mutual authentication over SSL virtual host junctions is also supported.

Validation of the back-end server certificate

WebSEAL verifies a back-end server certificate according to the standard SSL protocol. The back-end server sends its server certificate to WebSEAL. WebSEAL validates the server certificate against a pre-defined list of root Certificate Authority (CA) certificates.

The Certificate Authority (CA) certificates that form the trust chain for the application server certificate (from the signing CA up to and including the root certificate) must be included in the key database in use by WebSEAL.

You use the LMI to create and manage the database of root CA certificates.

Matching the distinguished name (DN)

About this task

You can enhance server-side certificate verification through distinguished name (DN) matching. To enable server DN matching, you must specify the back-end server DN when you create the SSL junction to that server. Although DN matching is an optional configuration, it provides a higher degree of security with mutual authentication over SSL junctions.

During server-side certificate verification, the DN contained in the certificate is compared with the DN defined by the junction. The connection to the back-end server fails if the two DNs do not match.

Procedure

To enable the server DN matching, specify the back-end server DN when you create the SSL-based junction using the **–D "*DN*"** option. To preserve any blank spaces in the string, surround the DN string with double quotation marks. For example:

```
-D "CN=Access Manager,OU=SecureWay,O=Tivoli,C=US"
```

The **–D** option is appropriate only when used with the **–K** or **–B** option.

Authentication with a client certificate

Use the **–K** option to enable WebSEAL to authenticate to the junctioned back-end server using its client certificate.

```
-K "key_label"
```

The conditions for this scenario include:

- The back-end server is set up to require verification of WebSEAL's identity with a client certificate.
- Using the LMI to create, label, and store a special key that is used solely as WebSEAL's client certificate when authenticating to a junctioned back-end server.
- For greater security, additionally configure the junction for DN matching (**–D**).

The **–K** option uses an argument that specifies the key-label of the required certificate as stored in the GSKit key database. Use the LMI to add new certificates to the key database.

You must surround the key-label argument with quotation marks. For example:

```
-K "cert1_Tiv"
```

If the key is located on cryptographic hardware, you must specify the WebSEAL token device with the key label.

```
-K "token_name:key-label"
```

For example:

```
-K "websealtoken:junctionkey"
```

See “Configuration of the WebSEAL key database file” on page 367.

Authentication with a BA header

Use the **–B–U "username"–W "password"** option to enable WebSEAL authentication using basic authentication.

```
-B -U "username" -W "password"
```

The conditions for this scenario include:

- The back-end server is set up to require verification of WebSEAL's identity with a BA header.
- Do not configure the junction with any **–b** option. (Internally, however, the **–B** option uses **–b filter**.)
- WebSEAL is configured to pass its identity information in a BA header to authenticate to the back-end server.
- For greater security, additionally configure the junction for DN matching (**–D**).

You must surround the user name and password arguments with double quotation marks. For example:

```
-U "WS1" -W "abCde"
```

TCP and SSL proxy junctions

You can create WebSEAL junctions that allow communication to traverse network topologies that use HTTP or HTTPS proxy servers. You can configure the junction to handle requests as standard TCP communication or protected SSL communication.

The **create** command requires one of the following arguments to the **type** option to establish either a TCP-based or SSL-based junction through a proxy server:

- **–t tcpproxy**
- **–t sslproxy**

Both **create** and **add** commands require the following options and arguments to identify the proxy server and the target Web server:

Table 56. Options for the create and add commands

Options	**Description**
–H *host-name*	The DNS host name or IP address of the proxy server.
–P *port*	The TCP port of the proxy server.
–h *host-name*	The DNS host name or IP address of the target Web server.
–p *port*	The TCP port of target Web server. Default is **80** for TCP junctions; **443** for SSL junctions.

Example TCP proxy junction (entered as one line):

```
pdadmin> server task web1-webseald-cruz create -t tcpproxy
-H clipper -P 8081 -h www.ibm.com -p 80 /ibm
```

Example SSL proxy junction (entered as one line):

```
pdadmin> server task web1-webseald-cruz create -t sslproxy
-H clipper -P 8081 -h www.ibm.com -p 443 /ibm
```

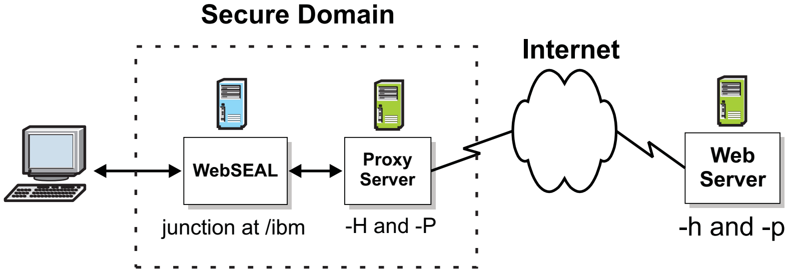


Figure 29. Example proxy junction

TCP and SSL proxy virtual host junctions are also supported.

WebSEAL-to-WebSEAL junctions over SSL

Security Access Manager supports SSL junctions between a front-end WebSEAL server and a back-end WebSEAL server. Use the **–C** option with the **create** command to junction the two WebSEAL servers over SSL and provide mutual authentication.

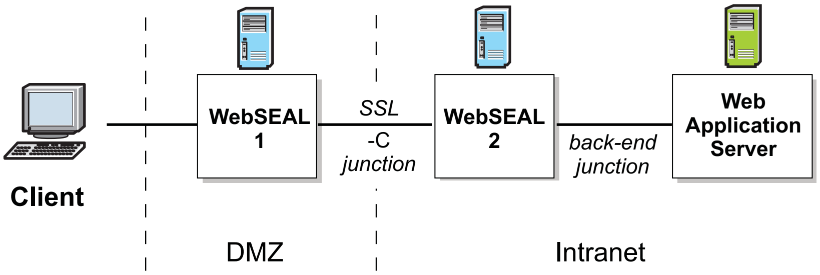


Figure 30. WebSEAL-to-WebSEAL junction scenario

Example:

```
pdadmin> server task web1-webseald-cruz create -t ssl -C -h serverA /jctA
```

Mutual authentication occurs in the following two stages:

- The SSL protocol allows the back-end WebSEAL server to authenticate to the front-end WebSEAL server through its server certificate.
- The **–C** option enables the front-end WebSEAL server to pass its identity information to the back-end WebSEAL server in a Basic Authentication (BA) header.

Additionally, the **–C** option enables single signon functionality provided by the **–c** option. The **–c** option allows you to place Security Access Manager-specific client identity and group membership information into the HTTP header of the request destined for the back-end WebSEAL server. The header names include iv-user, iv-groups, and iv-creds. See “Client identity in HTTP headers (–c)” on page 520.

The following conditions apply to WebSEAL-to-WebSEAL junctions:

- The junction is appropriate only with the **–t ssl** or **–t sslproxy** junction type.

- Both WebSEAL servers must share a common user registry. This configuration allows the back-end WebSEAL server to authenticate the front-end WebSEAL server identity information.
- If the WebSEAL-to-WebSEAL junction and the back-end application server junction both use the **–j** junction option (for junction cookies), a naming conflict can occur between the two junction cookies created by each of the two WebSEAL servers. (Refer to the diagram at the beginning of this section.) To prevent this conflict, you must configure the intermediary WebSEAL server (WebSEAL 2 in the diagram) to uniquely identify its junction cookie. On the intermediary WebSEAL server only, set the value of the **hostname-junction-cookie** stanza entry in the **[script-filtering]** stanza of the WebSEAL configuration file to "yes" (default is "no"):

  ```
  [script-filtering]
  hostname-junction-cookie = yes
  ```

  Junction cookies allow WebSEAL to handle server-relative URLs generated on the client-side. These URLs lack knowledge of the junction point of the destination application. The junction cookie provides this information. For complete information on junction cookies, see “Modification of server-relative URLs with junction cookies” on page 435.

Stateful junctions

This section contains the following topics.

Stateful junction concepts

Most Web-enabled applications maintain a "state" for a sequence of HTTP requests from a client. This state is used, for example, to:

- Track a user's progress through the fields in a data entry form generated by a CGI program
- Maintain a user's context when performing a series of database inquiries
- Maintain a list of items in an online shopping cart application where a user randomly browses and selects items to purchase

Back-end servers that run Web-enabled applications can be replicated in order to improve performance through load sharing. By default, Security Access Manager balances back-end server load by distributing requests across all available replicated servers. Security Access Manager uses a "least-busy" algorithm. This algorithm directs each new request to the server with the fewest connections already in progress.

However, when WebSEAL processes a request over a stateful junction, WebSEAL must ensure that all subsequent requests from that client during that session are forwarded to the same server, and not distributed among the other replicated back-end servers according to the load balancing rules.

Configuration of stateful junctions

Use the **pdadmin server task create** command with the **–s** option to override load balancing rules and create a stateful junction. A stateful junction ensures that a client's requests are forwarded to the same server throughout an entire session. When the initial client request occurs over a stateful junction, WebSEAL places a cookie on the client system that contains the UUID of the designated back-end server. When the client makes future requests to the same resource during the same session, the cookie's UUID information ensures that the requests are consistently routed to the same back-end server.

The **–s** option is appropriate for a single front-end WebSEAL server with multiple back-end servers junctioned at the same junction point. Note that as soon as the initial junction is created as stateful, the **pdadmin server task add** command is used without the **–s** option to junction the remaining replicated back-end servers to the same junction point.

Stateful virtual host junctions are also supported.

If the scenario involves multiple front-end WebSEAL servers, all junctioned to the same back-end servers, you must use the **–u** option to correctly specify each back-end server UUID to each front-end WebSEAL server. See "Specifying back-end server UUIDs for stateful junctions."

You can also control how WebSEAL handles a stateful server that becomes unavailable. See "Handling an unavailable stateful server" on page 400.

Specifying back-end server UUIDs for stateful junctions

About this task

When a new junction is created to a back-end Web application server, WebSEAL normally generates a Unique Universal Identifier (UUID) to identify that back-end server. This UUID is used internally and also to maintain stateful junctions (**create –s**).

When the initial client request occurs, WebSEAL places a cookie on the client system that contains the UUID of the designated back-end server. When the client makes future requests to the same resource, the cookie's UUID information ensures that the requests are consistently routed to the same back-end server.

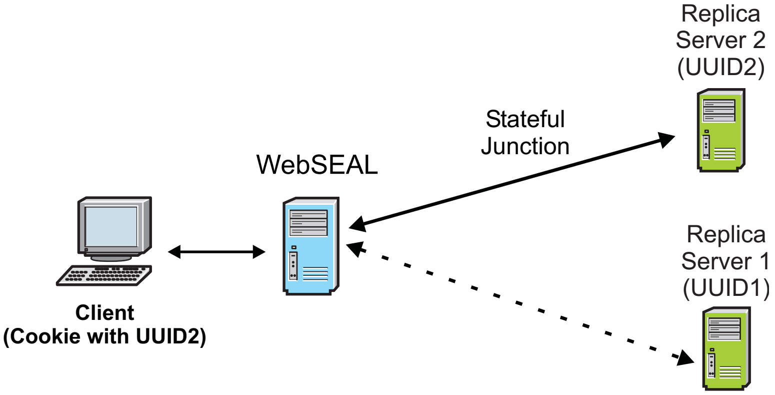


Figure 31. Stateful junctions use back-end server UUIDs

The handling of stateful junctions becomes more complex when there are multiple front-end WebSEAL servers junctioned to multiple back-end servers. Normally, each junction between a front-end WebSEAL server to a back-end server generates a unique UUID for the back-end server. This means that a single back-end server will have a different UUID on each front-end WebSEAL server.

Multiple front-end servers require a load balancing mechanism to distribute the load between the two servers. For example, an initial "state" could be established to a back-end server through WebSEAL server 1 using a specific UUID.

However, if a future request from the same client is routed through WebSEAL server 2 by the load balancing mechanism, the "state" will no longer exist, unless WebSEAL server 2 uses the same UUID to identity the same back-end server. Normally, this will not be the case.

The **–u** option allows you to supply the same UUID for a specific back-end server to each front-end WebSEAL server.

The **–u** option is also supported on virtual host junctions.

As an example, consider two replicated front-end WebSEAL servers, each with a stateful junction to two back-end servers. When you create the stateful junction between WebSEAL server 1 and back-end server 2, a unique UUID (UUID A) is generated to identify back-end server 2. However, when a stateful junction is created between WebSEAL server 2 and back-end server 2, a new and different UUID (UUID B) is generated to identify back-end server 2.

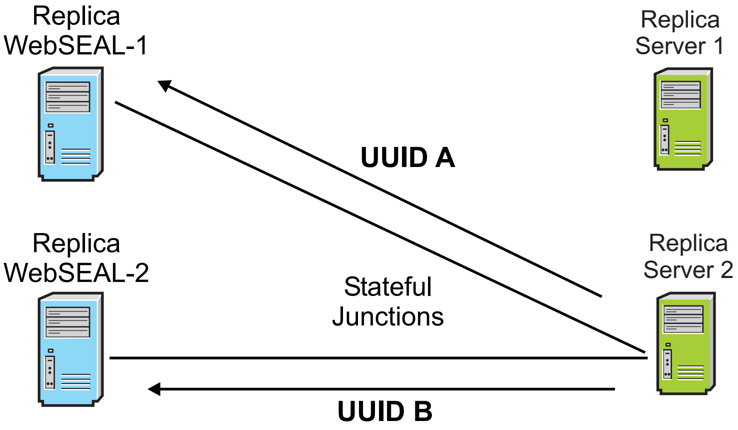


Figure 32. Dissimilar UUIDs

A "state" established between a client and back-end server 2, via WebSEAL server 1 will fail if a subsequent request from the client is routed through WebSEAL server 2.

In the following figure, back-end server 1 is known by both WebSEAL-1 and WebSEAL-2 as UUID 1. Back-end server 2 is known by both WebSEAL-1 and WebSEAL-2 as UUID 2.

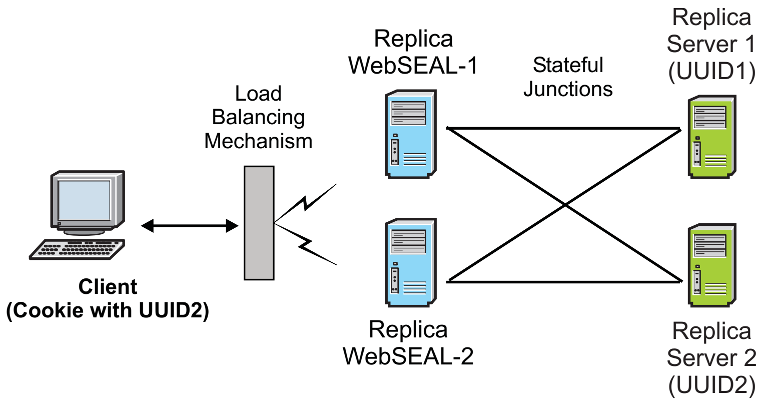


Figure 33. Specifying back-end server UUIDs for stateful junctions

Procedure

Apply the following process for specifying a UUID during the creation of a junction:

1. Create a junction from WebSEAL server 1 to each back-end server. Use **create –s** and **add**.
2. List the UUID generated for each back-end server during step 1. Use **show**.
3. Create a junction from WebSEAL server 2 to each back-end server and specify the UUIDs identified in Step 2. Use **create –s –u** and **add –u**.

Stateful junction example:

In the following example,

- WebSEAL-1 instance is called WS1. Host name is **host1**.
- WebSEAL-2 instance is called WS2. Host name is **host2**.
- Back-end server 1 is called APP1
- Back-end server 2 is called APP2

```
pdadmin> server task WS1-webseald-host1 create -t tcp -h APP1 -s /mnt
pdadmin> server task WS1-webseald-host1 add -h APP2 /mnt
pdadmin> server task WS1-webseald-host1 show /mnt

(output of this command reveals UUID1 and UUID2)

pdadmin> server task WS2-webseald-host2 create -t tcp -h APP1 -u UUID1 -s /mnt
pdadmin> server task WS2-webseald-host2 add -h APP2 -u UUID2 /mnt
```

When a client establishes a stateful connection with back-end server 2, it receives a cookie containing UUID2. This example now ensures that the client will always connect to back-end server 2, regardless of whether future requests are routed through WebSEAL-1 or WebSEAL-2.

Handling an unavailable stateful server

About this task

You can use the **use-new-stateful-on-error** stanza entry in the **[junction]** stanza of the WebSEAL configuration file to control how WebSEAL responds to a stateful server that becomes unavailable.

When **use-new-stateful-on-error** is set to "yes" and the original server becomes unavailable during a session, WebSEAL directs the next request of the user to a new replica server on the same stateful junction. If a new replica server is found on that stateful junction, and is responsive to the request, WebSEAL sets a new stateful cookie on the browser of the user. Subsequent requests during this same session are directed to this same new server.

When **use-new-stateful-on-error** is set to "no" and the original server becomes unavailable during a session, WebSEAL does not direct the subsequent requests of the user to a new replica server on the same stateful junction. Instead, WebSEAL returns an error and attempts to access the same server for subsequent requests by the user during this session. The "no" value is the default setting and maintains compatibility with versions of WebSEAL prior to version 6.0. For example:

```
[junction]
use-new-stateful-on-error = no
```

To end the access attempts on an unresponsive server, the user must restart the browser, or the administrator must remove the unresponsive server.

This configuration item may be customized for a particular junction by adding the adjusted configuration item to a `[junction:{`*`junction_name`*`}]` stanza.

where *{junction_name}* refers to the junction point for a standard junction (including the leading / character) or the virtual host label for a virtual host junction. For example:

```
[junction:/WebApp]
```

Forcing a new junction

About this task

You must use the **–f** junction option when you want to force a new junction to overwrite an existing junction.

The following example (WebSEAL instance name = **cruz**) illustrates this procedure:

Procedure

1. Log in to **pdadmin**:

   ```
   # pdadmin
   pdadmin> login
   Enter User ID: sec_master
   Enter Password:
   pdadmin>
   ```

2. Use the **server task list** command to display all current junction points:

   ```
   pdadmin> server task web1-webseald-cruz list
   /
   ```

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

   ```
   pdadmin> server task web1-webseald-cruz show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /opt/pdweb/www/docs
   ...
   ```

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

```
pdadmin> server task web1-webseald-cruz create -t local -f -d /tmp/docs /
Created junction at /
```

5. List the new junction point:

   ```
   pdadmin> server task web1-webseald-cruz list
   /
   ```

6. Display the details of this junction:

   ```
   pdadmin> server task web1-webseald-cruz show /
   Junction point: /
   Type: Local
   Junction hard limit: 0 - using global value
   Junction soft limit: 0 - using global value
   Active worker threads: 0
   Root Directory: /tmp/docs
   ...
   ```

   The **–f** option is also supported on virtual host junctions.

Use of /pkmslogout with virtual host junctions

Policies can be attached to pkmslogout, but WebSEAL does not always apply the policies.

For example, if a user authenticated to WebSEAL and tries to access pkmslogout, the pkmslogout page ends the user session without an authorization check. ACL policies are not applied to such requests. However, if a user has not authenticated to WebSEAL and tries to access pkmslogout, the request is treated as a normal request. WebSEAL conducts an authorization check.

If the authorization check fails, the request proceeds as a normal authorization failure. In the default WebSEAL configuration, the user is prompted to login.

If the authorization check passes, WebSEAL attempts to retrieve an object called /pkmslogout from the root junction, and this typically results in a 404 Not Found response from WebSEAL.

The **allow-unauthenticated-logout** option in the **[acnt-mgmt]** stanza determines whether unauthenticated users are able to request the pkmslogout resource without authenticating first. If set to *yes*, WebSEAL behaves in the same manner whether the user logging out is authenticated or unauthenticated.

There are several methods to achieve single logout using Security Access Manager. One method is to embed <IMG> or <IFRAME> HTML tags in a logout page so that the browser simultaneously logs the user out of multiple servers when the page is viewed. For example, the following HTML tags send requests to /pkmslogout on three different virtual hosts:

```
<img src="https://www.example.com/pkmslogout" height="0" width="0">
<img src="https://sales.example.com/pkmslogout" height="0" width="0">
<img src="https://accts.example.com/pkmslogout" height="0" width="0">
```

If this technique is used for single logout, it can be beneficial to either attach ACLs to /pkmslogout or to use the **[acnt-mgmt] allow-unauthenticated-logout** option to control WebSEAL behavior. For more information about the **allow-unauthenticated-logout** option, see allow-unauthenticated-logout.

Junction throttling

This section contains the following topics:

Junction throttling concepts

Regular maintenance on equipment in a computer network environment is a crucial and necessary task. In a WebSEAL environment, data storage and application programs typically reside on junctioned back-end host machines that are protected by WebSEAL. High demand WebSEAL environments usually rely on server clusters made up of multiple machines hosting replicated content and applications.

A replica server environment allows you to take individual servers offline to perform regular maintenance. The network load is redistributed across the remaining replicas allowing the user experience to proceed without disruption.

Junction throttling allows you to gradually take a junctioned back-end Web server offline without interrupting the transactions of users with existing sessions. The throttling action on a junction is particularly useful for allowing stateful sessions, such as shopping cart transactions, to continue until completed.

Junction throttling accomplishes the following actions:

- The throttled server continues to process current and subsequent requests from users with sessions created before the throttle action was taken.
- The throttled server blocks all requests from unauthenticated users and new authenticated users and directs these requests to other available replica servers on the same junction.
- As the current users finish their sessions, the throttled server eventually becomes idle and can be taken offline.
- Junction throttling does not require you to stop WebSEAL and does not interrupt user access to other junctioned Web servers.

The **pdadmin** utility provides commands to place junctioned servers in one of three operational states:

- Throttle
- Offline
- Online

Note: These operational states are different from the run states of a junctioned server: running, not running, unknown, not an http server. The server run states is reported in the "Server State" field of the **pdadmin server task show** and **pdadmin server task virtualhost show** commands.

The commands allow you to individually or collectively control the servers on a junction. Collective control might be required, for example, in the case of a security breach.

The junction throttling feature is supported on standard WebSEAL junctions and virtual host junctions. Junction throttling is not available for standard local junctions or virtual host local junctions.

Placing a junctioned server in a throttled state

About this task

Place a junctioned server in a throttled operational state when you want a controlled and gradual transition of the server to an offline state. The throttled operational state places the following conditions on the junction:

- A throttled server continues to process current and subsequent requests from users with sessions created before the throttle action was taken.
- The throttled server blocks all requests from unauthenticated users and new authenticated users and directs these requests to other available replica servers on the same junction.
- When a request is blocked by a throttled server, it is retried on another replica server. If no replica servers are configured or available, an error page is returned to the user indicating the application server is offline.
- The throttled operational state for a junction is stored in the junction database and is maintained across any restarts of the WebSEAL server.

Throttle command usage for standard WebSEAL junctions:

The following **pdadmin** command syntax is appropriate for use with standard WebSEAL junctions:

```
server task instance_name-webseald-host_name throttle [-i server_uuid] jct_point
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in a throttled operational state. If a server is not specified using this option, then all servers located at the junction are placed in a throttled operational state.

Example:

The following example places the `backappl` server located at the `/pubs` junction point in a throttled operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to the following:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Online
Hostname: backappl.diamond.example.com
...
Current requests: 0
...
```

Then, place this server in a throttled operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz throttle
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

Throttle command usage for virtual host junctions:

The following **pdadmin** command syntax is appropriate for use with virtual host junctions:

```
server task instance_name-webseald-host_name virtualhost throttle [-i server_uuid]
vhost_label
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in a throttled operational state. If a server is not specified using this option, then all servers located at the junction are placed in a throttled operational state.

In the following example, the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`.

There is a requirement to place the `int3.ibm.com` server in a throttled operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command:

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Online
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in a throttled operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost throttle
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

Junctioned server in an offline state

Place a junctioned server in an offline operational state when you want to block all requests from all users. The offline operational state places the following conditions on the junction:

- An offline server continues to process existing requests.
- An offline server blocks all subsequent requests from all users.
- When a request is blocked by an offline server, it is retried on another replica server. If no replica servers are configured or available, an error page is returned to the user indicating the application server is offline.
- The offline operational state for a junction is stored in the junction database and is maintained across any restarts of the WebSEAL server.

Offline command usage for standard WebSEAL junctions:

The following **pdadmin** command syntax is appropriate for use with standard WebSEAL junctions:

```
server task instance_name-webseald-host_name offline [-i server_uuid] jct_point
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in an offline operational state. If a server is not specified using this option, then all servers located at the junction are placed in an offline operational state.

Example:

The following example places the `backappl` server located at the `/pubs` junction point in an offline operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to the following:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Throttled
Throttled at: 2005-03-01-17:07:24
Hostname: backapp1.diamond.example.com
...
Current requests: 0
...
```

Then, place this server in an offline operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz offline
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

Offline command usage for virtual host junctions:

The following **pdadmin** command syntax is appropriate for use with virtual host junctions:

```
server task instance_name-webseald-host_name virtualhost offline [-i server_uuid]
vhost_label
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in an offline operational state. If a server is not specified using this option, then all servers located at the junction are placed in an offline operational state.

In the following example, the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`.

There is a requirement to place the `int3.ibm.com` server in an offline operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command:

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Throttled
Throttled at: 2005-03-01-17:07:24
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in an offline operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost offline
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

Junctioned server in an online state

Place a junctioned server in an online operational state when you want to return the server to normal operation. The online operational state places the following conditions on the junction:

- An online server resumes normal operations.
- The online operational state for a junction is stored in the junction database and is maintained across any restarts of the WebSEAL server.

Online command usage for standard WebSEAL junctions:

The following **pdadmin** command syntax is appropriate for use with standard WebSEAL junctions:

```
server task instance_name-webseald-host_name online [-i server_uuid] jct_point
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in an online operational state. If a server is not specified using this option, then all servers located at the junction are placed in an online operational state.

Example:

The following example places the `backappl` server located at the `/pubs` junction point in an online operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to the following:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Offline
```

```
Hostname: backapp1.diamond.example.com
...
Current requests: 0
...
```

Then, place this server in an online operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz online
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

Online command usage for virtual host junctions:

The following **pdadmin** command syntax is appropriate for use with virtual host junctions:

```
server task instance_name-webseald-host_name virtualhost online [-i server_uuid]
vhost_label
```

Use the **-i** option to specify the UUID of the junctioned server that is being placed in an online operational state. If a server is not specified using this option, then all servers located at the junction are placed in an online operational state.

In the following example, the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`.

There is a requirement to place the `int3.ibm.com` server in an online operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command:

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Offline
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in an online operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost online
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

Junction throttle messages

This section contains the following topics:

Junction throttle error page:

An error page (38b9a4b0.html) is returned to the user when a request is blocked due to a junctioned server that has been placed in a throttled or offline operational state. This error page is only sent if all servers on the junction are placed in a throttled or offline operational state.

You can customize the content of this HTML error page. See "HTML server response page modification" on page 65.

Alternatively, WebSEAL can be configured to redirect error reporting to an external error page service. See "Local response redirection" on page 79.

Monitoring of throttled server status and activity:

You can monitor the status and activity of a throttled server using three new fields to the **pdadmin server task show** and **pdadmin server task virtualhost show** commands:

- Operational state: {Throttled|Offline|Online}
- Current requests: *number*

  This field displays the number of worker threads actively using a junctioned server and allows you to determine when the server has become idle.
- Throttled at: *date-time*

  This field only displays when the server is in the throttled operational state.

For example:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to the following:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Throttled
Throttled at: 2005-03-01-17:07:24
Hostname: backapp1.diamond.example.com
...
Current requests: 3
...
```

The "Current requests" field helps to determine if a server is idle. However, there are numerous conditions that prevent a completely accurate determination of the state of server activity. For example:

- The"Current requests" field is not updated during the lag time between the reading of the body of the response from the junctioned server and the return of the response to the client.
- If session lifetime resets and extensions have been configured, it becomes almost impossible to be completely sure no future operations will occur on the server.

Use of junction throttling with existing WebSEAL features

Junction throttling has an impact on the following WebSEAL functions.

- Failover authentication

  Failover authentication transparently supports failed over sessions that continue to use a throttled junction if the original session was created before the junction was throttled. The session creation time is added as an attribute to the failover cookie so it can be restored when a failover cookie is used to authenticate. When the failover cookie is used for authentication, the session creation time from the cookie is set for the newly created failover session.
- Distributed session cache

  The distributed session cache makes the session creation time available to all processes that are sharing the session. The session creation time is important because only sessions created before a junction server is throttled are allowed continued access to the throttled junction server.
- Reauthentication

  Reauthenticated sessions are allowed continued access to a throttled junction server if the sessions are initially created before the junction was throttled. The additional effect of session lifetime extensions or resets can make it difficult for you to determine when the throttled junction is truly idle.
- Switch user

  When a switch user event occurs, a new session creation time is generated. This new creation time is used to determine accessibility to a throttled junction server. When the switch user logs out and returns to the original identity, the original session creation time becomes effective again and is used to determine accessibility to a throttled junction server.
- Stateful junctions

  Stateful junctions allow requests from a specific session to always be sent to the same server on a junction. If the junctioned server being used is throttled, the stateful session is allowed to continue accessing that server. However, new stateful sessions are blocked from using that server.

  If a junctioned server is taken offline, then stateful sessions are no longer allowed to access the server. These sessions must choose a new junctioned server and possibly loose the original state information.
- Step-up authentication

  Step-up authentication does not create a new session. The session creation time is therefore not affected, and the ability of the session to access a throttled junction does not change.

Management of cookies

WebSEAL can host cookies on behalf of browsers and provide them to backend applications in forwarded requests. These stored cookies are held in the session cache, or cookie jar, rather than being sent to the browser.

The WebSEAL cookie jar is instantiated on a per-user session basis. Cookies not stored in the cookie jar are passed back to the client for storage.

The cookie jar stores and handles cookies as defined by the following configuration entries in the `[junction]` stanza:

managed-cookies-list

Contains a comma-separated list of pattern matched cookie names. Cookies that match the patterns on this list are stored in the cookie jar. If this list is empty, the cookie jar is effectively disabled.

Note: WebSEAL processes the cookies found in the response before attempting to add the cookies to the cookie jar. It can sometimes, based on the junction configuration, prefix the original cookie name with a junction identifier. This new cookie name will be used when WebSEAL decides whether to add a cookie to the cookie jar.

reset-cookies-list

Determines which cookies to reset when the user session is logged out. The request received from the client and the response sent back to the client are both examined for matching cookies. The reset clears the cookie in the client's cache by returning an empty and expired cookie in the logout response. This essentially implements a basic logout for junctioned applications.

Note: WebSEAL processes the cookies found in the response before deciding which cookies to reset. It can sometimes, based on the junction configuration, prefix the original cookie name with a junction identifier. This new cookie name will be used when WebSEAL decides which cookies to reset.

share-cookies

Determines whether or not cookies stored in the cookie jar will be shared between different junctions.

validate-backend-domain-cookie

Domain checking on cookies is only performed if this entry is set to *true*.

allow-backend-domain-cookies

During domain validation, if this entry is set to *false*, the domain is removed from the cookie.

Note: All the preceding configuration items, with the exception of **share-cookies**, can be customized for a particular junction by adding the adjusted configuration item to a **[junction:*{junction_name}*]** stanza.

where *{junction_name}* refers to the junction point for a standard junction (including the leading / character) or the virtual host label for a virtual host junction.

All response cookies pass through the WebSEAL cookie jar. Cookies that match the patterns defined in **managed-cookies-list** are stored in the cookie jar and removed from the response stream to the browser. Those that are not stored in the cookie jar are passed back to the client.

When a request to a junctioned server is sent from the browser to WebSEAL, the cookie jar is checked to see if the request requires cookies to be sent to the junctioned server. If the request does require a cookie from the cookie jar, the cookie is added to the request. If the cookie has expired, the cookie is removed from the cookie jar and not sent.

Persistent cookies are not persisted to disk on the WebSEAL machines.

When a user performs a logout, a reset for selected cookies that are not stored in the cookie jar is sent back in the response. WebSEAL resets any cookies with names that match the list of patterns in the **reset-cookies-list** stanza entry. The reset essentially implements a basic logout for junctioned applications.

Note: The distributed session cache should be deployed in situations where the cookie jar is used by multiple replicated WebSEAL servers. The distributed session cache is the mechanism by which the cookie jar can be distributed amongst the multiple replicated WebSEAL servers. In this type of environment, be careful which cookies you place in the cookie jar. Do not include cookies which get updated on a regular basis, as this will put additional load on the distributed session cache which in turn will have performance implications in the environment.

Passing of session cookies to junctioned portal servers

A Web portal is a server that offers a broad array of personalized resources and services. The **–k** junction option allows you to send the Security Access Manager session cookie (originally established between the client and WebSEAL) to a back-end portal server.

When a client requests a personal resource list from the portal server, the portal server builds this list by accessing resources located on other supporting application servers, also protected by WebSEAL. The session cookie allows the portal server to perform seamless single signon to these application servers, on behalf of the client.

You include the **–k** option, without arguments, when you create the junction between WebSEAL and the back-end portal server.

The **–k** option is also supported on virtual host junctions.

The WebSEAL configuration file includes options that provide some control over how session cookies are handled during step-up authentication. The **verify-step-up-user** option in the **[step-up]** stanza determines whether the identity of the user performing the step-up operation must match the identity of the user that performed the previous authentication. If this option is set to yes, then the **retain-stepup-session** option can be used to determine whether the session cookie issued during the step-up operation can be reused or if a new cookie must be issued. If **verify-step-up-user** is set to no, then a new cookie will always be issued after step-up.

The **send-constant-sess** option in the **[session]** stanza enhances the ability to track authenticated sessions. Setting this option to yes enables WebSEAL to send a separate cookie to the junctioned server in addition to the session cookie. The value of this cookie remains constant across a single session, regardless of whether the session key changes. The name of the cookie is configurable. For more details regarding the **send-constant-sess** option, send-constant-sess.

Conditions to consider for a portal server configuration:

- For access using user name and password, forms authentication is required. Do not use basic authentication (BA).
- The value of the **ssl-id-sessions** stanza entry in the **[session]** stanza of the WebSEAL configuration files must be set to no. For HTTPS communication, this setting forces the use of a session cookie, instead of the SSL session ID, to maintain session state.
- If the portal server is behind a front-end WebSEAL cluster, enable the failover type cookie. The failover cookie contains encrypted credential information that allows authentication to succeed with any replicated WebSEAL server that processes the request.
- The **retain-stepup-session** option in the **[step-up]** stanza is only in effect if the **verify-step-up-user** option is set to yes.

- If WebSEAL is configured to use the distributed session cache for session storage, the **verify-step-up-user** option must be set to yes to enable step-up operations. If this option is set to no, then WebSEAL does not update the distributed session cache when user identification changes during step-up authentication.

For more information about step-up authentication, see "Authentication strength concepts" on page 186.

Support for URLs as not case-sensitive

By default, Security Access Manager treats URLs as case-sensitive when performing checks on access controls. The **–i** junction option is used to specify that WebSEAL treat URLs as not case-sensitive when performing authorization checks on a request to a junctioned back-end server.

The **–i** option is also supported on virtual host junctions.

When you set this option on the junction, WebSEAL does not distinguish between uppercase and lowercase characters when parsing URLs. By default, Web servers are expected to be case-sensitive.

Although most HTTP servers support the HTTP specification that defines URLs as case-sensitive, some HTTP servers treat URLs as not case-sensitive.

For example, on not case-sensitive servers, the following two URLS:

```
http://server/sales/index.htm
```

```
http://server/SALES/index.HTM
```

are viewed as the same URL. This behavior requires an administrator to place the same access controls (ACLs) on both URLs.

By junctioning a third-party server with the **–i** option, WebSEAL treats the URLs directed to that server as not case-sensitive.

To correctly authorize requests for junctions that are not case sensitive, WebSEAL does the authorization check on a lowercase version of the URL. For example, a Web server running on Windows treats requests for INDEX.HTM and index.htm as requests for the same file.

Junctions to such a Web server should be created with the **-i** [or **-w**] flags. ACLs or POPs that are attached to objects beneath the junction point should use the lower case object name. An ACL attached to /junction/index.htm will apply to all of the following requests if the **-i** or **-w** flags are used:

```
/junction/INDEX.HTM
/junction/index.htm
/junction/InDeX.HtM
```

This option is valid for all junctions except for the type of local. Local junctions are not case-sensitive only on Win32 platforms; all other platforms are case-sensitive.

Attention: When using the –i option, object names must be lower case in order for WebSEAL to be able to find any ACLs or POPs attached to those objects. For more information, see “ACLs and POPs must attach to lower-case object names” on page 415.

Junctions to Windows file systems

WebSEAL performs security checks on client requests to junctioned back-end servers based on the file paths specified in the URL. A compromise in this security check can occur because Win32 file systems allow two different methods for accessing long file names.

The first method acknowledges the entire file name. For example:

```
abcdefghijkl.txt
```

The second method recognizes the old 8.3 file name format for backward compatibility. For example:

```
abcdef~1.txt
```

When you create junctions in a Windows environments, it is important to restrict access control to one object representation only and not allow the possibility of "back doors" that bypass the security mechanism.

The **–w** option on a junction provides the following measures of protection:

- Prevents the use of the 8.3 file name format

  When the junction is configured with the **–w** option, a user cannot avoid an explicit ACL on a long file name by using the short (8.3) form of the file name. The server returns a "403 Forbidden" error on any short form file name entered.
- Disallows trailing dots in directory and file names

  If a file or directory contains trailing dots, a 403 "Forbidden" error is returned.
- Enforces case-insensitivity by setting the **–i** option

  The **–w** option automatically invokes the **–i** option. This option specifies that WebSEAL treat URLs as case-insensitive when performing authorization checks on a request to a junctioned back-end server. After a successful ACL check, the original case of the URL is restored when the request is sent to the back-end server.

  Note: If you require control over case-insensitivity only for file names, use only the **–i** option on the junction instead of the **–w** option.

The **–w** option is also supported on virtual host junctions.

Example

In a Windows environment, the file:

```
\Program Files\Company Inc\Release.Notes
```

can also be accessed through the following paths:

1. `\progra~1\compan~2\releas~3.not`
2. `\Program Files\Company Inc.\Release.Notes`
3. `\program files\company inc\release.notes`

Example 1 illustrates how Windows can create an alias (for DOS compatibility) that contains no spaces in the file names and conforms to the 8.3 format. The **–w** option causes WebSEAL to reject this format for ACL checks.

Example 2 illustrates how Windows can include trailing extension dots. The **–w** option causes WebSEAL to reject this format for ACL checks.

Example 3 illustrates how Windows allows case-insensitivity on the file name. The **–w** option invokes the **–i** option to ensure a case-insensitive ACL check.

ACLs and POPs must attach to lower-case object names

When a junction is created with the **–w** or **–i** option, WebSEAL performs ACL and POP comparisons as not case-sensitive. This means that the name of any object being evaluated for an ACL is placed into lowercase before WebSEAL checks it against the object list to which ACLs are attached.

As a result, protected objects with names that contain uppercase letters are not found during the ACL or POP checks. If these objects are not found, the ACL or POP is not applied to the protected object, and the parent policy is applied instead.

To avoid the possible misapplication of policy in this configuration, you must create lowercase versions of the same names of the real protected objects to which you want to attach explicit ACLs or POPs.

Standard junctions to virtual hosts

Virtual hosting refers to the practice of maintaining more than one WebSEAL instance on the same physical machine. Virtual hosting is used to run multiple Web services, each with different host names and URLs, and which appear to be on completely separate sites. Virtual hosting is used by ISPs, hosting sites, and content providers who need to manage multiple sites but do not want to buy a new machine for each one. Multiple companies can share a single Web server but have their own unique domains, such as **www.company1.com** and **www.company2.com**.

Users of virtual host domains do not need to know any extra path information. In requests to a virtual host domain, browsers simply provide the required hostname through its **Host:** header.

In a standard junction create command, the **-h host** option and argument identifies the remote junctioned server machine. WebSEAL normally places this server address in the **Host:** header of requests to this server.

The additional **-v host** option and argument identifies a virtual host instance on this machine. The virtual host information provided by **-v** is used in the **Host:** header instead of the **-h** information. Requests are now directed to the specific virtual host instance. The **-h** information is still used by WebSEAL to locate the physical host machine.

For mutual junctions the **-v host** option corresponds to the virtual host name when the HTTP protocol is used to communicate with the junctioned server. The **-V host** option is used as the virtual host name when the HTTPS protocol is used.

The following example illustrates the configuration of two virtual hosts. The junctioned application server is **server.xyz.com**. It runs on port 80 and hosts two virtual servers: **site1.xyz.com** and **site2.xyz.com**.

Basic junctions are created as follows (each command entered as one line):

```
pdadmin> server task default-webseald-surf.xyz.com create -t tcp
-h server.xyz.com -v site1.xyz.com /jct1
pdadmin> server task default-webseald-surf.xyz.com create -t tcp
-h server.xyz.com -v site2.xyz.com /jct2
```

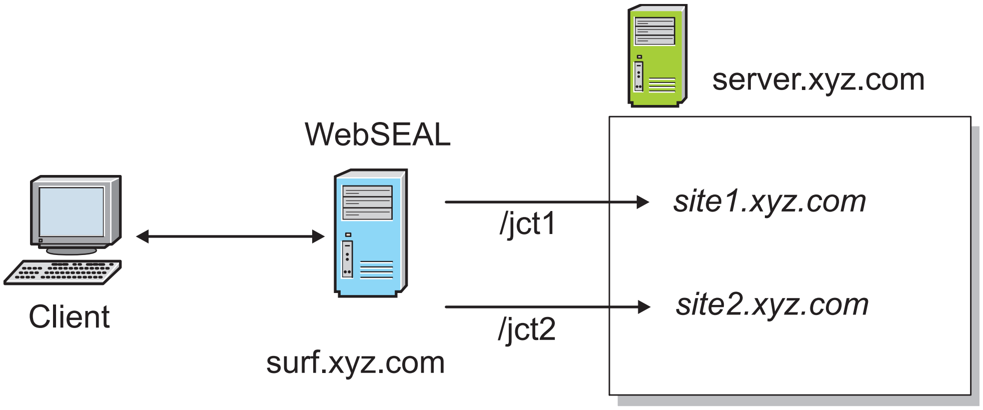


Figure 34. Configuring virtual hosts

See also "Complete Host header information with -v" on page 590.

UTF-8 encoding for HTTP header data

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

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

The **-e** junction option specifies the encoding of user name, groups, and other extended attributes which are sent within the HTTP header to the backend server.

The **–e** option is also supported on virtual host junctions.

The **-e** encode option can take one of the following arguments:

Argument	Description
utf8_uri	URI encoded UTF-8 data. All white space and non-ASCII bytes are encoded %XY, where X and Y are hex values (0–F).

Argument	Description
utf8_bin	Unencoded UTF-8 data. This setting allows data to be transmitted without data loss, and the customer does not need to URI-decode the data. This setting should be used with caution, because it is not part of the HTTP specification
lcp_uri	URI encoded local code page data. Any UTF-8 characters that cannot be converted to a local code page will be converted to question marks (?). Use this option with caution and only in environments where the local code page produces the desired strings.
lcp_bin	Unencoded local code page data. This mode was used by versions of WebSEAL prior to Version 5.1. Use of this mode enables migration from previous versions, and is used in upgrade environments. Use with caution, because data loss can potentially occur with this mode.

Bypassing buffering on a per-resource basis

WebSEAL uses an internal buffer when processing data sent in requests to WebSEAL and responses from junction applications.

About this task

This buffering typically provides performance improvement. For certain applications that send or return small amounts of data, the buffering can cause the data to be held temporarily at WebSEAL while the buffer is being filled. For some applications, it might be preferable to bypass the buffering and stream the data directly to the junctioned server or to the clients. This scheme is not efficient for general web traffic; apply it only to particular resources that require streamed data. For example, apply it to junctions configured for RPC over HTTP communication. See “Microsoft RPC over HTTP” on page 466.

You can apply a protected object policy (POP) to individual resources that directs WebSEAL to bypass buffering for those resources. To bypass buffering for a particular resource response, attach a POP to the resource with an attribute named **response-buffer-control** set with the value **bypass**. To bypass buffering for a particular resource request, attach a POP to the resource with an attribute named **request-buffer-control** set with the value **bypass**.

The following example

- Creates a POP named *bypassPOP*.
- Sets the **response-buffer-control** and **request-buffer-control** attributes to **bypass**.
- Attaches the POP to a resource named *smallCGI*

Procedure

1. Create a POP named *bypassPOP* with the appropriate attributes.

```
pdadmin> pop create bypassPOP
pdadmin> pop modify bypassPOP set attribute response-buffer-control bypass
pdadmin> pop modify bypassPOP set attribute request-buffer-control bypass
```

2. Attach the POP to the chosen resource.

   ```
   pdadmin> pop attach /WebSEAL/myinstance/myjunction/cgi-bin/smallCGI bypassPOP
   ```

   This POP only affects the data in the body of the request or response that is received from the client or junction. WebSEAL still buffers the request and response headers.

   When buffering HTTP requests using this POP technique, there are limitations. Certain WebSEAL functions require the entire request body, and this body is not available when streaming a request to a junctioned server.

   The following WebSEAL functionality cannot be used when using *request* streaming:

   Note: WebSEAL *response* streaming can still be applied to resources that use this WebSEAL functionality.

   - Caching of POST data during the authentication process.
   - Dynamic authorization decision information (dynADI) when POST data is part of the decision evaluation.
   - Dynamic URLs (dynURL) when POST data is part of the decision evaluation.

WebSockets

WebSEAL can proxy WebSocket connections between clients and junctioned web servers. In the WebSEAL default configuration, all WebSocket requests are rejected.

To enable WebSocket proxy support, update the WebSEAL configuration file and configure the **[websocket] max-worker-threads** to a value larger than zero.

Each WebSocket created between the client and the junctioned server requires two WebSocket worker threads. The **max-worker-threads** setting must be configured to allow for this thread usage.

If the **max-worker-threads** limit is reached, then any additional requests to proxy a WebSocket connection is rejected and a warning message is logged.

WebSocket connections can have a non-trivial lifespan as a client can keep the connection open for extended periods of time with data traveling in both directions asynchronously. The **max-worker-threads** entry decides how many concurrent clients can be handled by WebSEAL. To help reduce the number of idle or blocked WebSocket connections, WebSEAL provides the following timeout settings:

- **[websocket] jct-read-inactive-timeout**
- **[websocket] jct-writeblocked- timeout**
- **[websocket] clt-write-blocked-timeout**
- **[websocket] clt-read-inactive-timeout**

The settings that begin with **jct** impact the connection between WebSEAL and the junctioned server. The settings that begin with **clt** impact the connection between WebSEAL and the client or browser. Carefully evaluate the WebSocket data transmission behavior before you set these timeouts.

WebSEAL's **pdweb.snoop** trace also applies to WebSocket data. Enabling **pdweb.snoop** trace allows tracing of the raw WebSocket data that is sent and received on each WebSocket connection. The **pdweb.snoop** trace can be enabled for the client traffic, the junction traffic, or both, by using **pdweb.snoop.client** and **pdweb.snoop.jct** trace elements.

Statistics can be gathered on WebSockets by enabling the **pdweb.websocket** stats component. The provided statistics are shown in the following table.

Statistic label	Description
requests	Total WebSocket proxy requests received while statistics gathering is enabled.
rejected	Total WebSocket proxy requests rejected while statistics gathering is enabled. The rejection is typically due to an insufficient number of available worker threads.
timeout	The number of timeouts that have occurred when reading or writing through a proxied WebSocket connection while statistics gathering is enabled.
active	The current number of WebSocket connections that are proxied.
client bytes	The number of bytes read from the client side.
junction bytes	The number of bytes read from the junction side.

Modification of URLs to junctioned resources

This chapter discusses the configuration options available to ensure that URL links sent from back-end application servers across standard WebSEAL junctions are reconstructed appropriately for use by clients.

The challenges of URL filtering are specific to standard WebSEAL junctions. WebSEAL also supports virtual hosting and, through virtual host junctions, can eliminate the limitations of URL filtering. WebSEAL uses the HTTP **Host** header in client requests to direct those requests to the appropriate document spaces located on junctioned servers or on the local machine.

Topic Index:

URL modification concepts

Pages returned to the client from back-end application servers often contain links to resources located on those servers. It is important that these URLs are constructed to correctly direct any client requests back to these resources.

For example, in a non-WebSEAL environment, the URL entered by a client for a resource on an application server might appear as follows:

```
http://www.example.com/file.html
```

WebSEAL, as a front-end reverse proxy, provides security services to back-end application servers via the WebSEAL junctioning feature. This feature requires that the original URLs to these resources be modified to include the junction information.

The standard junction feature of WebSEAL changes the server and path information that must be used to access resources on junctioned back-end systems. A link to a resource on a back-end junctioned server succeeds only if the URL contains the identity of the junction.

If this same back-end server is a junctioned server in a WebSEAL environment, the URL used to access the same resource on a junctioned back-end application server must appear as follows:

```
http://webseal.example.com/jct/file.html
```

To support the standard junction feature and maintain the integrity of URLs, WebSEAL must, where possible:

1. Modify the URLs (links) found in **responses** sent to clients
2. Modify **requests** for resources resulting from URLs (links) that WebSEAL could not change

Note: WebSEAL's rules and mechanisms for modifying URLs do not apply to links that point to resources external to the Security Access Manager junctioned environment.

The following diagram summarizes the solutions available to WebSEAL for modifying URLs to junctioned back-end resources:

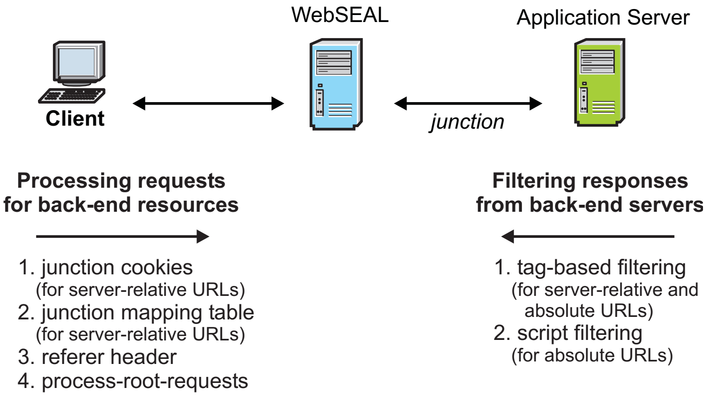

WebSEAL
Application Server
Client
junction
Processing requests for back-end resources
1. junction cookies (for server-relative URLs)
2. junction mapping table (for server-relative URLs)
3. referer header
4. process-root-requests
Filtering responses from back-end servers
1. tag-based filtering (for server-relative and absolute URLs)
2. script filtering (for absolute URLs)

Figure 35. Summary: Modifying URLs to back-end resources

Path types used in URLs

Any HTML page is likely to contain URLs (links) to other resources on that back-end server or elsewhere. URL expressions can appear in the following formats:

- relative
- server-relative
- absolute

Links containing URLs expressed in **relative** format never require any modification by WebSEAL. By default, the browser handles relative URLs in links by adding the correct scheme (protocol), server name, and directory information (including the junction) as a prefix to the relative URL. The browser derives this information from the location information of the page on which the link is located.

Example **relative** URL expressions:

```
abc.html
../abc.html
./abc.html
sales/abc.html
```

However, difficulties arise with **server-relative** and **absolute** path formats.

A server-relative URL contains a full path expression (beginning at /), but contains to scheme (protocol) or server name. Example **server-relative** URL expressions:

```
/abc.html
/accounts/abc.html
```

An absolute URL contains the scheme (protocol), server name, and full path. Example **absolute** URL expression:

```
http://www.example.com/doc/abc.html
```

Links to back-end resources expressed in absolute or server-relative formats succeed only if WebSEAL is able to modify the URL path expression to include junction information. WebSEAL URL modification techniques apply to absolute and server-relative URLs.

Note: To ensure successful location of resources, all programmers of Web scripts are encouraged to use **relative** links (not absolute or server-relative) for dynamically generated URLs.

Special characters in URLs

HTML pages can contain ASCII special characters. The web content contains HTML codes that represent the special characters or symbols.

The HTML codes for special characters start with "&#" and end with ";".

The HTML codes use the following format:

```
&#<numeric_id>;
```

where

<numeric_id>
A number or hex representation for the character. The maximum value of the number representation is 255 (Hex: 0xFF).

For example, the percent sign (%) has a numerical code of &#37; and a hex code of &#x25;.

Note: For details about how WebSEAL can filter any special character encoding that exists in HTTP responses, see “Modification of encoded or escaped URLs” on page 424.

Modification of URLs in responses

This section describes options for modifying URLs in responses from junctioned back-end application servers.

Filtering of tag-based static URLs

WebSEAL uses a set of default rules to scan for (or filter) tag-based static URLs contained in pages that are responses to client requests. This default filtering mechanism examines static URLs located within tag-based content (such as HTML or XML).

The term "filtering" is used to indicate WebSEAL's process of scanning Web documents for absolute and server-relative links and modifying the links to include junction information.

An important requirement for this mechanism is that the URLs must be visible to WebSEAL. For example, tag-based content filtering cannot handle URLs that are dynamically generated on the client-side.

When the **preserve-base-href2** option is also set to "yes" WebSEAL only performs the minimum filtering of the BASE HREF tag necessary to insert the WebSEAL host and junction names. The value of the **preserve-base-href2** option impacts the processing of any BASE HREF tags that are missing the trailing slash.

REMINDER: This option has no effect if **preserve-base-href** is set to "no".

Example 5 (BASE HREF that contains no trailing slash matches a junctioned server)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering

NOTE: There is no trailing slash in the BASE HREF value:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example.com">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering when **preserve-base-href2** is set to "no".

NOTE: WebSEAL maps the HREF to `/jct` and then eliminates the `/jct` because there is no trailing slash.

```
<HTML>
<HEAD>
<BASE HREF="http://www.webseal.com/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering when **preserve-base-href2** is set to "yes".

NOTE: WebSEAL performs the minimum filtering of the BASE HREF tag necessary to insert the WebSEAL host and junction names. The missing trailing slash does not cause the last component in the base HREF URL to be stripped.

```
<HTML>
<HEAD>
<BASE HREF="http://www.webseal.com/jct/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

This section contains the following topics:

Filter rules for tag-based static URLs:

Filter rules for relative URLs:

Relative URLs are always handled appropriately by the browser. Therefore, WebSEAL does not filter relative URLs.

By default, the browser add the correct scheme, server, and directory information (including the junction) to the beginning of the relative URL. The added information is derived from the request URL for the page on which the link is located.

Filter rules for server-relative URLs:

WebSEAL must add the junction name to the path of server-relative URLs that refer to resources located on junctioned servers.

Server-relative URLs indicate a URL position in relation to the document root of the junctioned server, for example:

```
/dir/file.html
```

Server-relative URLs are modified by adding the junction point of the junctioned server to the path name. For example:

```
/jct/dir/file.html
```

Filter rules for absolute URLs:

WebSEAL must add the junction name to the path of absolute URLs that refer to resources located on junctioned servers.

Absolute URLs indicate a URL position in relation to a host name or IP address (and, optionally, a network port). For example:

```
http://host-name[:port]/file.html, or
https://host-name[:port]/file.html
```

Absolute URLs are modified according to the following set of rules:

- If the URL is HTTP and the host/port matches a TCP junctioned server, the URL is modified to be server-relative to WebSEAL and reflect the junction point. For example:

  ```
  http://host-name[:port]/file.html
  ```

  becomes:

  ```
  /tcpjct/file.html
  ```

- If the URL is HTTPS and the host/port matches an SSL junctioned server, the URL is modified to be server-relative to WebSEAL and reflect the junction point. For example:

```
https://host-name[:port]/file.html
```

becomes:

```
/ssljct/file.html
```

See also "Configuring the rewrite-absolute-with-absolute option" on page 431.

See "Modifying absolute URLs with script filtering" on page 430 for an alternative absolute URL filtering mechanism.

Default filtering of tag-based static URLs:

WebSEAL filters tag-based static URLs that are located in pages with content (MIME) types specified in the **[filter-content-types]** stanza of the WebSEAL configuration file.

By default, this stanza specifies MIME types `text/html` and `text/vnd.wap.wml`.

```
[filter-content-types]
type = text/html
type = text/vnd.wap.wml
```

The **[filter-url]** stanza of the WebSEAL configuration file specifies the tags and tag attributes to be filtered in the content. Because most commonly used HTML tags and tag attributes are listed in this stanza by default, tag-based filtering for the default `text/html` and `text/vnd.wap.wml` MIME types typically operates without any additional configuration.

Modification of encoded or escaped URLs:

WebSEAL modifies and rewrites the URL references in outgoing responses to ensure that subsequent requests are sent through WebSEAL.

WebSEAL rewrites each URL in its original form with the URI encoding or slash escaping. If WebSEAL encounters special character encoding, it replaces the HTML code with the special character that the code represents.

The following table describes the URI encoding types that WebSEAL filters during this process.

Table 57. Filtered encoding types. WebSEAL filters the following encoding types.

Encoding type	Example of encoded URL
Ampersand encoded	`HTTP://host&#x3a;port/path?V1=D1 &amp;V2=D2`
Ampersand - hex encoded	`HTTP&#x3a;&#x2f;&#x2f;host&#x3a;port &#x2f;`
Ampersand - dec encoded	`HTTP&#58;&#57;&#57;host&#58;port&#57;`
Backslash encoded	`HTTP:\/\/host:port\/`
Percent hex encoded	`HTTP%3A%2F%2Fhost%3Aport%2F`

Example: Filtering of URLs

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

The HTML before the WebSEAL filtering process:

```
<HTML>
<BODY>
<A HREF="http%3A%2F%2Fwww.example.com%2Findex.html">index.html</A>
<A HREF="http:\/\/www.example.com\/page2.html">page2.html</A>
<A HREF="http://www.example.com/page3.html">page3.html</A>
<A HREF="http&#x3A;&#x2F;&#x2F;www.example.com&#x2F;page4.htm">page4.html</A>
</BODY>
</HTML>
```

HTML after the WebSEAL filtering process with the configuration item `[script-filtering] rewrite-absolute-with-absolute` set to `no`:

```
<HTML>
<BODY>
<A HREF="%2Fjct%2Findex.html">index.html</A>
<A HREF="\/jct\/page2.html">page2.html</A>
<A HREF="/jct/page3.html">page3.html</A>
<A HREF="&#x2F;jct&#x2F;page4.htm">page4.html</A>
</BODY>
</HTML>
```

HTML after the WebSEAL filtering process with the configuration item `[script-filtering] rewrite-absolute-with-absolute` set to `yes`:

```
<HTML>
<BODY>
<A HREF="http%3A%2F%2Fwww.webseal.com%2Fjct%2Findex.html">index.html</A>
<A HREF="http:\/\/www.webseal.com\/jct\/page2.html">page2.html</A>
<A HREF="http://www.webseal.com/jct/page3.html">page3.html</A>
<A HREF="http&#x3A;&#x2F;&#x2F;www.webseal.com&#x2F;jct&#x2F;page4.htm">
page4.html</A>
</BODY>
</HTML>
```

Configuration of filtering for new content (MIME) types:

Additional MIME types can be configured for URL filtering by adding appropriate entries to the **[filter-content-types]** stanza.

The default filtering mechanism for new MIME types added to the **[filter-content-types]** stanza is script filtering. You must edit the WebSEAL configuration file to enable script filtering (script filtering is disabled by default):

```
[script-filtering]
script-filter = yes
```

The script filtering mechanism examines the entire contents of the response and is not restricted to tag-based content. However, script filtering only examines and modifies absolute URLs. See “Modifying absolute URLs with script filtering” on page 430.

Alternatively, you can enable tag-based filtering of static URLs for new MIME types added to the **[filter-content-types]** stanza. To enable tag-based static URL filtering for new configured MIME types, set the value of the **filter-nonhtml-as-xhtml** stanza entry in the **[server]** stanza of the WebSEAL configuration file to "yes":

```
[server]
filter-nonhtml-as-xhtml = yes
```

Specify any additional tags and tag attributes to be filtered, as described in “Tags and attributes for tag-based filtering” on page 426.

Tags and attributes for tag-based filtering:

Tag-based static URL filtering is constrained to the specific tags and tag attributes defined in the **[filter-url]** stanza of the WebSEAL configuration file. The filtering mechanism looks only for URLs located within the specified tags and tag attributes.

You can add entries to the **[filter-url]** stanza to specify any additional tags and tag attributes to be filtered. Tags and tag attributes can include HTML, XML, XHTML, or custom data definitions.

HTML META tags:

HTML META refresh tags are always filtered. For example:

```
<META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://server/url">
```

HTML BASE HREF tags:

You can use the **preserve-base-href** and the **preserve-base-href2** entries in the **[server]** stanza to control how WebSEAL handles the HREF attributes of HTML BASE tags in filtered HTML documents.

Note: The value of **preserve-base-href2** has no effect unless the **preserve-base-href** option is set to "yes".

When **preserve-base-href** is set to "no", the following statements apply:

- WebSEAL removes the BASE HREF attributes from filtered HTML documents.
- When the BASE HREF URL matches a junctioned server, WebSEAL prepends the junction point, and any subdirectories, to server-relative and relative links found on the page.
- When the BASE HREF URL does not match a junctioned server, WebSEAL prepends the entire BASE HREF URL attribute to server-relative and relative links on the page. This action is exactly what the browser does when it renders HTML.

Example 1 (BASE HREF matches a junctioned server)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example.com/dir1/dir2/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE>
</HEAD>
```

```
<BODY>
<A HREF="/jct/dir1/dir2/index.html">index.html</A>
</BODY>
</HTML>
```

Example 2 (BASE HREF does not match a junctioned server)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example2.com/dir1/dir2/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE>
</HEAD>
<BODY>
<A HREF="http://www.example2.com/dir1/dir2/index.html">index.html</A>
</BODY>
</HTML>
```

When **preserve-base-href** is set to "yes", the following statements apply:

- WebSEAL does not remove the BASE HREF attributes from filtered HTML documents.
- When the BASE HREF URL matches a junctioned server, WebSEAL modifies the BASE HREF URL. WebSEAL replaces the name of the junctioned server with the WebSEAL server name, plus the junction point of the junctioned server. When the browser renders the HTML, and prepends the modified HREF URL to server-relative and relative links, the resulting links can find the resources.
- When the BASE HREF URL does not match a junctioned server, WebSEAL does not modify the HTML. When the browser renders the HTML, and prepends the original HREF URLs to server-relative and relative links, the resulting links can find the resources.
- When WebSEAL filters an HTML document with a BASE HREF tag, WebSEAL does not preserve the original escaping or encoding of the URL if it contains:
  - URI encoding
  - Escaped slashes
  - Special character encoding

Example 3 (URI Encoded, slash-escaped and Special Character URLS)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example2.com/dir1/">
</HEAD>
<BODY>
<A HREF="dir2%2Findex.html">index.html</A>
<A HREF="dir2\/contents.html">contents.html</A>
<A HREF="dir2&#x2F;index2.html">index2.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE>
</HEAD>
<BODY>
<A HREF="http://www.example2.com/dir1/dir2/index.html">index.html</A>
<A HREF="http://www.example2.com/dir1/dir2/contents.html">contents.html</A>
<A HREF="http://www.example2.com/dir1/dir2/index2.html">index2.html</A>
</BODY>
</HTML>
```

Example 4 (BASE HREF matches a junctioned server)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example.com/dir1/dir2/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.webseal.com/jct/dir1/dir2/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

Example 5 (BASE HREF does not match a junctioned server)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example2.com/dir1/dir2/">
</HEAD>
```

```
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example2.com/dir1/dir2/">
</HEAD>
<BODY>
<A HREF="index.html">index.html</A>
</BODY>
</HTML>
```

Example 6 (URI Encoded, slash-escaped and Special Character URLs)

- WebSEAL server: `www.webseal.com`
- The following junction was created:

  ```
  server task webseal-server create -tcp -h www.example.com /jct
  ```

HTML before filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.example.com/dir1/">
</HEAD>
<BODY>
<A HREF="dir2%2Findex.html">index.html</A>
<A HREF="dir2\/contents.html">contents.html</A>
<A HREF="dir2&#X2F;index2.html">index2.html</A>
</BODY>
</HTML>
```

HTML after filtering:

```
<HTML>
<HEAD>
<BASE HREF="http://www.webseal.com/jct/dir1/">
</HEAD>
<BODY>
<A HREF="dir2/index.html">index.html</A>
<A HREF="dir2/contents.html">contents.html</A>
<A HREF="dir2/index2.html">index2.html</A>
</BODY>
</HTML>
```

Schemes to ignore in pages using the BASE tag:

The **[filter-schemes]** stanza lists those URL schemes that are not to be filtered by WebSEAL in responses from junctioned application servers. A scheme is a protocol identifier. This list is used when WebSEAL encounters documents containing HREF attributes in HTML BASE tags.

The **[filter-schemes]** stanza contains a set of default schemes.

- You can extend the list with additional protocols.
- The recommended practice is to not delete entries from this list.
- The "http" and "https" schemes are hard-coded to always be filtered.
- When adding entries to the stanza, the trailing colon (:) on the scheme name is optional. When the colon is missing, WebSEAL assumes it.

Default scheme entries:

```
[filter-schemes]
scheme = file
scheme = ftp
scheme = mailto
scheme = news
scheme = telnet
```

If a URL in a response does not use a scheme from this stanza, and the scheme is not "http" or "https", then WebSEAL assumes the URL is the same scheme as the junctioned server (HTTP: or HTTPS:) with it's scheme missing.

Exception! If the response contains an HTML BASE tag with an HREF URL attribute that uses a scheme from the stanza, WebSEAL filters the URL.

Modifying absolute URLs with script filtering

About this task

WebSEAL requires additional configuration to handle the processing of absolute URLs embedded in scripts. Web scripting languages include JavaScript, VBScript, ASP, JSP, ActiveX, and others. The **script-filter** stanza entry in the **[script-filtering]** stanza of the WebSEAL configuration file enables or disables filtering of embedded absolute URLs. Script filtering is disabled by default:

```
[script-filtering]
script-filter = no
```

Procedure

To enable script filtering, set this the value of this stanza entry to "yes":

```
[script-filtering]
script-filter = yes
```

The script filtering mechanism examines the entire contents of a response and is not restricted to, for example, tag-based content.
The **script-filter** mechanism expects absolute URLs with a standard scheme, server, resource format:

```
http://server/resource
```

The **script-filter** mechanism replaces the scheme and server portions of the link with the correct junction information (as a relative pathname):

```
/junction-name/resource
```

This filtering solution parses a script embedded in HTML code and therefore requires additional processing overhead that can negatively impact performance. The setting applies to all junctions. Only enable the **script-filter** stanza entry when your WebSEAL environment requires filtering of embedded absolute URLs.
The following diagram illustrates this absolute URL filtering solution:

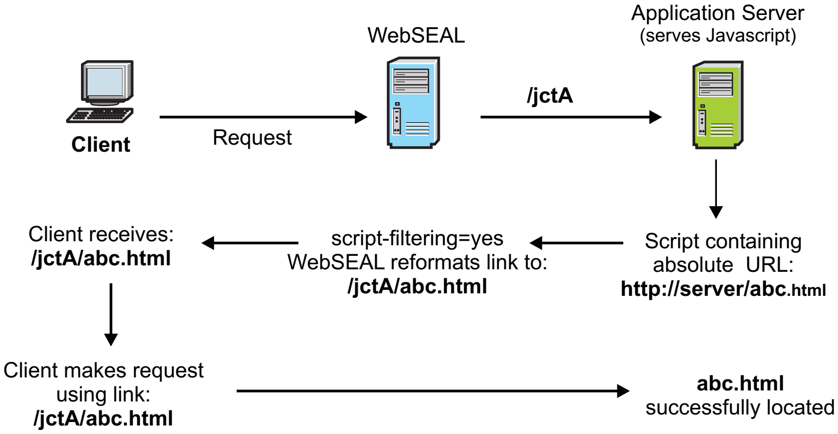


Figure 36. Filtering absolute URLs

Configuring the rewrite-absolute-with-absolute option

About this task

WebSEAL normally filters absolute URLs by adding the junction point and changing the format to a server-relative expression. This rule for filtering absolute URLs applies to tag-based filtering and script filtering. You can optionally configure WebSEAL to rewrite the original absolute URL as an absolute URL, instead of a relative URL.

Procedure

To enable this type of filtering, set the value of the **rewrite-absolute-with-absolute** stanza entry in the **[script-filtering]** stanza of the WebSEAL configuration file to equal "yes":

```
[script-filtering]
rewrite-absolute-with-absolute = yes
```

When **rewrite-absolute-with-absolute** is enabled, the following example URL in a response from a back-end server (connected to WebSEAL through `jctA`):

```
http://server/abc.html
```

is modified as follows:

```
http://webseal-hostname/jctA/abc.html
```

Enabling **rewrite-absolute-with-absolute** affects both tag-based filtering and script filtering. See:

- "Modifying absolute URLs with script filtering" on page 430 and
- "Filter rules for tag-based static URLs" on page 423

Filtering changes the Content-Length header

About this task

Normally, the **Content-Length** header in a response from a back-end server indicates the size of the content being returned. When WebSEAL filters URLs and adds junction information to the path of URLs contained in the page, the actual size of the page becomes larger than indicated in the **Content-Length** header.

WebSEAL has no way of knowing what the new content length is until it actually writes the stream to the client. At this point, it is too late to insert a new Content-Length header. WebSEAL responds to this situation in the following manner:

Procedure

1. WebSEAL places the value of the original **Content-Length** header in a new header called **X-Old-Content-Length**

   Any applets or applications written to look for this header can have access to the original (pre-filtered) Content-Length value.
2. WebSEAL logs the modified (post-filtered) **Content-Length** value in the `request.log` file.
3. The **Content-Length** header no longer appears.

Limitation with unfiltered server-relative links

Problem:

WebSEAL provides solutions for processing client-side, script-generated, server-relative URLs to resources on back-end junctioned application servers. The server-relative URLs generated on the client-side by applets and scripts initially lack knowledge of the junction point in the path expression. During a client request for a resource, WebSEAL can attempt to reprocess a server-relative URL using junction cookies or a junction mapping table.

However, before the processing takes place, the request actually specifies a resource located on the local Web space of the WebSEAL server itself. The corrective reprocessing of the URL occurs only after WebSEAL receives the request and performs an ACL check.

An ACL check on the unprocessed request that specifies an incorrect or nonexistent local resource, might result in an error. The error might stop the request.

For example, the following sequence takes place during processing:

1. The client makes a request for a resource using a client-side, script-generated, server-relative URL.
2. The server-relative URL is received by WebSEAL as a request.

   The unprocessed URL specifies a resource located in the Web space of the WebSEAL server itself (obviously, this is not the intended resource).
3. WebSEAL performs an ACL check on this local resource specified in the request URL.
   - If the ACL check fails, all processing of the request stops and the client receives a 403 error (Forbidden). This error occurs because the ACL check was performed for the incorrect (and probably nonexistent) resource.
   - If the ACL check succeeds and the resource exists in the local Web space, it is returned. This error results in the client receiving the incorrect resource.
   - If the ACL check succeeds and the resource does not exist in the local Web space, WebSEAL modifies the request URL (using the junction cookie or junction mapping table method) and performs an internal reprocessing of the request. This behavior is correct.

4. WebSEAL performs another ACL check on the modified URL that contains the corrected path that includes the junction point. This modified URL now allows an ACL check for the correct resource.

Workaround::

To solve this problem:

1. Always write scripts that generate relative URL links. Avoid absolute and server-relative URL links.
2. If you must use server-relative links, do not duplicate resource names and paths on both the WebSEAL server and the junctioned application server.
3. If you must use server-relative links, design your ACL model so that more prohibitive ACLs do not affect false resources specified by unfiltered URLs.

Modification of URLs in requests

Difficulties arise when URLs are dynamically generated by client-side applications (such as applets) or embedded in scripts in the HTML code. Web scripting languages include JavaScript, VBScript, ASP, JSP, ActiveX, and others. These applets and scripts execute when the page arrives at the client browser. WebSEAL never has an opportunity to apply its standard filtering rules to these URLs that are dynamically generated on the client-side.

Three options for modifying URLs in requests are available to WebSEAL and are applied in the following order of precedence:

1. Junction mapping table
2. Junction cookies
3. HTTP Referer header

This section describes the options for processing **server-relative** links (used to make requests for resources located on junctioned back-end application servers) that are dynamically generated on the client-side.

Note: There are no solutions available for handling **absolute URLs** generated on the client-side.

Topic index:

Modification of server-relative URLs with junction mapping

Server-relative URLs generated on the client-side by applets and scripts initially lack knowledge of the junction point. WebSEAL cannot filter the URL because it is generated on the client-side.

During a client request for a resource using this URL, WebSEAL can attempt to reprocess the server-relative URL using a junction mapping table. A junction mapping table maps specific target resources to junction names. Junction mapping is an alternative to the cookie-based solution for filtering dynamically generated server-relative URLs.

WebSEAL checks the location information in the server-relative URL with the data contained in the junction mapping table. WebSEAL begins searching from the top of the table and continues downward through the table. If the path information in the URL matches any entry in the table during the top-down search, WebSEAL directs the request to the junction associated with that location.

The table is an ASCII text file called `jmt.conf`. The name of this file is specified in the **`[junction]`** stanza of the WebSEAL configuration file:

```
jmt-map = jmt.conf
```

The format for data entry in the table consists of the junction name, a space, and the resource location pattern. You can also use wildcard characters to express the resource location pattern.

In the following example of the junction mapping configuration file, two back-end servers are junctioned to WebSEAL at **/jctA** and **/jctB**:

```
#jmt.conf
#junction-name resource-location-pattern
/jctA /documents/release-notes.html
/jctA /travel/index.html
/jctB /accounts/*
/jctB /images/weather/*.jpg
```

You must create the `jmt.conf` mapping table. This file does not exist by default. After you create the file and add data, use the **jmt load** command to load the data so that WebSEAL has knowledge of the new information.

```
pdadmin> server task server-name jmt load
JMT table successfully loaded.
```

The following conditions apply to the junction mapping table solution:

- The junction mapping solution handles inbound requests intercepted by WebSEAL. Requests made using unfiltered absolute URLs that point to a server external to the WebSEAL environment (and therefore never intercepted by WebSEAL) are not handled by the junction mapping table solution.
- This solution does not require the **–j** option or junction cookie.
- The mapping table requires setup and activation by a security administrator.
- Resource location pattern matching must be unique across the local Web space and across junctioned Web application servers.
- If there is a duplicate pattern entry in the file, the mapping table does not load. However, WebSEAL continues to run.
- If there is an error loading the mapping table, the mapping table is not available. However, WebSEAL continues to run.
- If the mapping table is empty or there is an error in the table entries, the mapping table does not load. However, WebSEAL continues to run.
- Any errors that occur while loading the mapping table result in serviceability entries in the WebSEAL server log file (`webseald.log`).
- By default, WebSEAL modifies the names of non-domain cookies (returned in responses from back-end applications) across junctions listed in the junction mapping table. WebSEAL creates unique cookie names to prevent possible naming conflicts with cookies returned across other junctions. There are two methods for disabling this feature:

  See "Handling cookies from servers across multiple -j junctions" on page 441.

See also "Controlling server-relative URL processing in requests" on page 440.

Modification of server-relative URLs with junction cookies

This section contains the following topics:

Junction cookie concepts:

HTML pages from back-end junctioned application servers can contain embedded applets or scripts that dynamically generate server-relative links on the client-side. WebSEAL cannot filter these URLs because they are dynamically generated on the client-side. Therefore, these server-relative URLs are expressed without knowledge of the junction point where the application server resides.

This section describes a cookie-based solution to modifying server-relative URLs dynamically generated on the client-side. When a client receives a page from a junctioned server, and requests a resource using a dynamically generated server-relative URL on this page, WebSEAL can attempt to reprocess the URL using a special cookie. The cookie contains the appropriate junction information.

This solution requires that you initially create the junction to the back-end application server using the **-j** option. The following sequence of steps explains the process flow:

1. Client makes a request for an HTML page from a back-end junctioned application server.

   In addition to other content, the page contains an embedded applet that generates a server-relative URL once the page is loaded on the client's browser.
2. The page is returned to the client across the junction that was created with the **-j** option.

   The **-j** option causes WebSEAL to prepend a JavaScript block at the beginning of the HTML page.

   The purpose of the JavaScript is to set a junction-identifying cookie on the browser.
3. When the page is loaded on the client's browser, the JavaScript runs and sets the junction-identifying cookie in the browser's cookie cache.

   The cookie is a session cookie containing the name of the junction.
4. The embedded applet on the page dynamically runs and generates the server-relative URL.
5. The client makes a request for a resource using this server-relative URL. The junction cookie information is sent as an HTTP header in this request:

   ```
   IV_JCT = /junction-name
   ```
6. Because the server-relative URL in the client request has not been filtered, it appears to WebSEAL as a request for a local resource.
7. When it fails to locate the resource locally, WebSEAL immediately retries the request using the junction information supplied by the cookie.
8. With the correct junction information in the URL expression, the resource is successfully located on the back-end application server.

The following diagram illustrates the junction cookie solution for filtering server-relative URLs

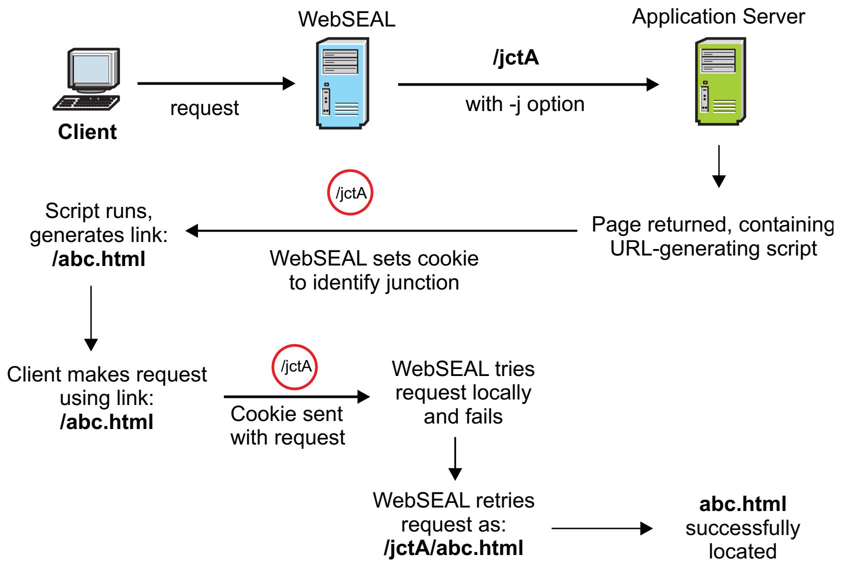

WebSEAL
Application Server
Client
request
/jctA
with -j option
/jctA
Script runs, generates link: /abc.html
WebSEAL sets cookie to identify junction
Page returned, containing URL-generating script
Client makes request using link: /abc.html
/jctA
Cookie sent with request
WebSEAL tries request locally and fails
WebSEAL retries request as: /jctA/abc.html
abc.html successfully located

Figure 37. Processing server-relative URLs with junction cookies

Configuration of WebSEAL junctions to support junction cookies:

Use the following general syntax to create a junction that supports junction cookies:

```
pdadmin> server task instance-webseald-host create ... -j ...
```

The following additional references contain related information:

- "Control on the junction cookie JavaScript block."
- "Controlling server-relative URL processing in requests" on page 440.
- "Handling cookies from servers across multiple -j junctions" on page 441.

WebSEAL provides an alternative, non-cookie-based solution for handling dynamically generated server-relative URLs. See "Modification of server-relative URLs with junction mapping" on page 433.

See also: "Modification of server-relative URLs using the HTTP Referer header" on page 439

Control on the junction cookie JavaScript block

This section describes two additional configuration options that are available when you use the **-j** junction option to modify server-relative URLs (see "Modification of server-relative URLs with junction cookies" on page 435):

Appending the junction cookie JavaScript block (trailer):

About this task

The **-j** junction option modifies HTML documents returned from junctioned servers by inserting a JavaScript block that sets a junction identification cookie on the browser interpreting the document. By default, the JavaScript block is inserted at the beginning of the page, before the <html> tag.

This prepended location of the JavaScript on the page can cause HTML rendering problems in some environments. If this type of problem is encountered, you can configure WebSEAL to append the JavaScript block to the end of the document instead.

Procedure

To configure WebSEAL to append the junction cookie JavaScript block to the end of pages returned by the back-end server, add the **-J** option with the **trailer** argument when creating the **-j** junction. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J trailer ...
```

If you are concerned with inserting the JavaScript block in the correct location for HTML 4.01 compliance, see "Inserting the JavaScript block for HTML 4.01 compliance (inhead)."

Inserting the JavaScript block for HTML 4.01 compliance (inhead):

The **-J** junction option modifies HTML documents returned from junctioned servers by inserting a JavaScript block that sets a junction identification cookie on the browser interpreting the document. By default, the JavaScript block is inserted at the beginning of the page, before any HTML code.

About this task

This prepended location of the JavaScript on the page can cause HTML rendering problems in some environments. Additionally, the default prepended location does not comply with HTML 4.01 specifications. The HTML 4.01 specification requires <script> tags to be located within the <head> </head> tags.

Procedure

To configure WebSEAL to insert the junction cookie JavaScript block between<head> </head> tags (HTML 4.01 compliant), add the **-J** option with the **inhead** argument when creating the **-J** junction. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J inhead ...
```

The **xhtml10** argument also addresses compliance with other HTML 4.01 and XHTML 1.0 specifications. See "Inserting an XHTML 1.0 compliant JavaScript block (xhtml10)" on page 438.
The **trailer** argument can be used when compliance with HTML 4.01 specifications is not required. See "Appending the junction cookie JavaScript block (trailer)" on page 436.

Resetting the junction cookie for multiple -j junctions (onfocus):

About this task

In environments where multiple instances of a single client access multiple **-j** junctions simultaneously, the most recent IV_JCT cookie created by the JavaScript may erroneously refer to a different junction than the one being currently accessed. In such a situation, WebSEAL receives the wrong junction information and fails to correctly resolve links.

For example, consider a scenario where a user has two browser windows open, each pointing to one of two junctions, **jctA** and **jctB**. Both junctions were created with the **-j** junction option.

Procedure

1. In the first browser window, the user requests a page from an application server located on **jctA**.
   The IV_JCT cookie for **jctA** is set in the browser.
2. The user then leaves the first browser window open, switches to the other browser window, and requests a page from an application server located on **jctB**.
   The IV_JCT cookie for **jctB** is set in the browser (replacing **jctA**).
3. If the user then returns to the first browser window and clicks a link to a resource located on **jctA**, the wrong IV_JCT cookie is sent to WebSEAL.

Results

To eliminate this problem, you can configure WebSEAL to use the onfocus event handler in the JavaScript. The onfocus handler resets the IV_JCT cookie whenever users switch the browser focus from one window to another.

To use the JavaScript onfocus event handler, add the **-J** option with the **onfocus** argument when creating the **-j** junction. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J onfocus ...
```

If you create a junction using the **onfocus** argument, it is best practise to use the **trailer** argument as well. The **trailer** argument ensures that the JavaScript inserted by WebSEAL does not interfere with the rendering of HTML frame sets. Use a comma character (,) and no spaces between the two arguments. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J trailer,onfocus ...
```

See also “Appending the junction cookie JavaScript block (trailer)” on page 436.

If compliance with HTML 4.01 and XHTML 1.0 specifications is required, see “Inserting an XHTML 1.0 compliant JavaScript block (xhtml10).”

Note: No error message is provided if the arguments specified for the **-J** option are invalid. If the **-J** junction option does not perform as expected, make sure you are providing the correct argument.

Inserting an XHTML 1.0 compliant JavaScript block (xhtml10):

The **-j** junction option modifies HTML documents returned from junctioned servers by inserting a JavaScript block that sets a junction identification cookie on the browser interpreting the document. The default script is not compliant with XHTML 1.0 specifications.

Procedure

To configure WebSEAL to insert a junction cookie JavaScript block that is compliant with XHTML 1.0 specifications (and HTML 4.01 specifications), add the **-J** option with the **xhtml10** argument when creating the **-j** junction. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J xhtml10 ...
```

Note:

- The JavaScript inserted using this option may not execute if the Content-Type of the document it is inserted into is not `text/html`. This will typically not be a problem because most sites issue XHTML with `Content-Type: text/html`.
- The **onfocus** and **xhtml10** arguments for the **-J** option are mutually exclusive. WebSEAL silently ignores **xhtml10** if **onfocus** has also been specified.

If you create a junction using the **xhtml10** argument, it is best practise to use the **inhead** argument as well. The **inhead** argument ensures that the placement of the JavaScript within the HTML code is compliant with HTML 4.01 specifications. For example (command line fragment):

```
pdadmin> server task instance-webseald-host create ... -j -J inhead,xhtml10 ...
```

See also "Inserting the JavaScript block for HTML 4.01 compliance (inhead)" on page 437.

Modification of server-relative URLs using the HTTP Referer header

HTML pages from back-end junctioned application servers can contain embedded applets or scripts that dynamically generate server-relative links on the client-side. WebSEAL cannot filter these URLs because they are dynamically generated on the client-side. Therefore, these server-relative URLs are expressed without knowledge of the junction point where the application server resides.

This section describes a solution for modifying server-relative URLs dynamically generated on the client-side. This solution involves use of the standard Referer header in an HTTP request. WebSEAL uses this solution only if a junction cookie cannot be found in a request or a junction mapping table entry does not match the request.

The information in the Referer header of an HTTP request can be used to identify the junction point of the application server responsible for the embedded applet or script. This solution assumes that the dynamically generated links point to resources located on the same application server (and therefore would require the same junction used by that application server)

A page returned from the back-end application server (and containing the links generated by the embedded applet or script) would provide knowledge of the junction name. The junction name will appear in the URL value of the Referer header of a request that results when the user clicks on one of the client-side-generated links located on this page. For example:

```
GET /back_end_app/images/logo.jpg
Referer: http://webseal/jctA/back_end_app
...
```

WebSEAL would not be able to find the resource using the request URL above (`/back_end_app/images/logo.jpg`). By using the information in the Referer header of that request, WebSEAL can modify the request URL to additionally include the junction name **jctA**. For example:

```
GET /jctA/back_end_app/images/logo.jpg
```

Using the modified URL, WebSEAL can successfully locate the resource. This of course assumes the resource (`logo.jpg`) is located on the same server.

If the environment results in client-side-generated links that point to resources across multiple junctions, the Referer header method for modifying URLs will not be reliable. In these environments, you must use either the junction mapping table solution or the junction cookie solution.

See also:

- "Modification of server-relative URLs with junction mapping" on page 433
- "Modification of server-relative URLs with junction cookies" on page 435

Controlling server-relative URL processing in requests

About this task

This section contains the following topics:

Process root request concepts:

This section discusses the process followed by WebSEAL when handling server-relative URLs in requests.

The **process-root-requests** stanza entry allows you to control the order in which WebSEAL processes a request involving a server-relative URL. For example, you can instruct WebSEAL to look for the requested resource at the WebSEAL root junction first. If the resource is "Not Found", WebSEAL continues processing the request by using any configured post-processing mechanism (such as junction cookie, junction mapping table (JMT), or Referer header).

Alternatively, you can have WebSEAL initially processes a server-relative URL request using any configured post-processing mechanism (such as junction cookie, junction mapping table (JMT), or Referer header). If the resource is "Not Found" by this method, WebSEAL then searches the root junction for the resource.

A third configuration allows you to filter specific path patterns.

For any request involving a server-relative URL, WebSEAL searches for the resource according to the configured setting of the **process-root-requests** stanza entry. If a "Not Found" error is returned, WebSEAL continues to process the request according to the **process-root-requests** configuration.

If any other error is returned, such as "Server Error" (500), WebSEAL returns the error to the client, assumes the resource is found, and stops further processing of the request. WebSEAL does not attempt to process the request on another junction. WebSEAL is functioning as intended in this situation.

The purpose of the **process-root-requests** configuration is to prevent server-relative URL processing from being performed for incorrect resources before the intended resource is identified. This action has performance benefits and prevents false authorization or file type check failures.

Configuring root request processing:

About this task

Use the **process-root-requests** entry in the **[server]** stanza of the WebSEAL configuration stanza to configure root junction processing.

For information on junction mapping mechanisms, see the following sections:

- "Modification of server-relative URLs with junction cookies" on page 435
- "Modification of server-relative URLs with junction mapping" on page 433

Procedure

Set the **process-root-requests** entry in the **[server]** stanza of the WebSEAL configuration stanza to one of the values provided. The default value is "always":

```
[server]
process-root-requests = always
```

Valid values are:

- **never**

  WebSEAL initially processes a server-relative URL request using any configured post-processing mechanism. Mechanisms are tried in the following order: 1) junction mapping table (JMT), 2) junction cookie, 3) HTTP Referer header. If the resource is not found by this method, WebSEAL then searches the root junction for the resource.
- **always**

  WebSEAL attempts to process server-relative URL requests at the root junction first before attempting to use any configured post-processing mechanism (such as junction cookie, junction mapping table (JMT), or Referer header). This setting should not be used unless the root junction serves a large set of resources or no post-processing junction mapping mechanisms are configured for the set of junctions served by this WebSEAL server.
- **filter**

  All root junction requests are examined to determine whether they start with the path patterns specified in the **[process-root-filter]** stanza. If they do, they are processed at the root junction first. If they do not start with path patterns specified in the **[process-root-filter]** stanza, they are remapped immediately.

When `process-root-requests = filter`, you must specify the patterns for which you want root junction requests processed at the root junction. Use the **[process-root-filter]** stanza. The syntax for specifying a pattern is:

```
root = path_pattern
```

Path pattern must be expressed as a standard WebSEAL wildcard pattern. For example:

```
[process-root-filter]
root = /index.html
root = /cgi-bin*
```

Handling cookies from servers across multiple -j junctions

About this task

This section describes how WebSEAL handles cookies generated by back-end applications and returned to clients across multiple **-j** junctions.

Cookie handling: -j modifies Set-Cookie path attribute

In addition to providing a junction identifier cookie to the browser, junctions configured with the **–j** option, or listed in a junction mapping table, also support the handling of non-domain cookies sent with responses from the back-end application.

Cookie handling by the browser:

If a **Set-Cookie** header in a response from the server contains a **path** attribute (such as `path=/xyz`), the browser returns the cookie only when a request URL (activated from the returned page) begins with this path (such as `/xyz/memo.html`).

Problem:

When the junction environment contains mixed solutions for handling visible and embedded URLs in responses, the ability of the browser to return cookies can be compromised. For example, standard WebSEAL filtering of visible server-relative URLs normally adds the junction name to the value of the **path** attribute of a server cookie (for example, `path=/jct/xyz`), in addition to modifying the URL itself. This match between URL path name and the cookie **path** value allows the browser to return the cookie when the link is activated by the user.

However, the **-j** junction-cookie-based solution adds the junction name to a URL only *after* the link (URL) has been activated by the user. When the link is activated, the pre-modified URL path name (`/xyz/memo.html`) does not match the **Set-Cookie path** attribute value (`path=/jct/xyz`). The server cookie is not returned with the request.

Solution:

The **-j** option converts the value of the **path** attribute for any server cookie to "/" (for example, `path=/`). Because all server-relative path names begin with a "/", all server cookies are returned regardless of the requirements of the original **path** attribute specifications.

Cookie handling: -j modifies Set-Cookie name attribute

Junctions configured with the **–j** option, or listed in a junction mapping table, also provide a solution for preserving cookies returned from servers across multiple junctions.

Cookie handling by the browser:

Browsers always replace any stored cookie with a newly arrived cookie that contains the same **Set-Cookie name** attribute, unless the **path** or **domain** attributes, or both, are unique.

Problem:

The previous section describes how the **-j** junction option modifies the value of the **path** attribute of a **Set-Cookie** header. This modification allows the browser to return cookies in an environment where WebSEAL is applying different filtering rules for visible and embedded URLs contained in the response pages.

In a scenario where multiple back-end servers are connected to WebSEAL across different junctions (such as in a WebSphere environment), it is possible for each server to send cookies with the same **name** attribute.

If the junctions use the **-j** option, the values of the **path** attribute for each cookie become identical (`path=/`). Because the same WebSEAL server is the point of contact for the browser, the **domain** attribute likewise becomes identical. Although these identical cookies arrive from unique back-end applications, the browser overwrites the identically named cookies.

Solution:

The **-j** junction option provides an additional feature that uniquely renames any cookie returned with a response from a back-end application server. A special string is added to the beginning of the **name** attribute of a **Set-Cookie** header. The string contains the identifier AMWEBJCT, plus the name of the specific junction responsible for delivering the response (with cookie). The exclamation point (!) is used as a separator character in the string.

```
AMWEBJCT!jct-name!
```

For example, if a cookie with the **name**, ORDERID, arrives across a junction called **/jctA**, the cookie **name** is changed to :

```
AMWEBJCT!jctA!ORDERID
```

To disable this default cookie-renaming feature, see “Preservation of cookie names.”

Preservation of cookie names

By default, WebSEAL modifies the name of cookies returned in responses from back-end applications across multiple junctions created with the **-j** option, or listed in the junction mapping table. WebSEAL creates unique cookie names to prevent possible naming conflicts with cookies returned across other **-j** junctions.

WebSEAL adds a special string to the beginning of the **name** attribute of a **Set-Cookie** header. The string contains the identifier AMWEBJCT, plus the name of the specific junction responsible for delivering the response (with cookie).

```
AMWEBJCT!jct-name!
```

For example, if a cookie with the **name**, ORDERID, arrives across a junction called **/jctA**, the cookie **name** is changed to:

```
AMWEBJCT!jctA!ORDERID
```

However, if front-end browsers and applications depend on the specific cookie name generated by the application, you can disable cookie-renaming for:

- All cookies across any junction configured with the **-n** option.
- Specific cookies, configured in the **[preserve-cookie-names]** stanza of the WebSEAL configuration file, across all junctions.

Preserving names of all cookies:

About this task

You can prevent renaming of non-domain cookies across a specific **-j** junction by additionally configuring that junction with the **-n** option. The **-n** option specifies that no modification of the names of non-domain cookies are to be made. For example, use this option when client-side scripts depend on the specific names of cookies.

Preserving names of specified cookies:

You can list the names of specific cookies that are not to be modified when returned from back-end application servers across junctions.

About this task

The **name** stanza entry in the **[preserve-cookie-names]** stanza of the WebSEAL configuration file allows you to list the specific cookie names that are not to be renamed by WebSEAL. The specified cookie names are protected across any existing junction.

Example

```
[preserve-cookie-names]
name = cookie-name1
name = cookie-name2
```

Cookie handling: -I ensures unique Set-Cookie name attribute

A junction configured with a **-j** option causes **Set-Cookie** headers in responses from back-end servers to have their **path** attribute value converted to "/", and their **name** attribute modified by including the junction point. Sometimes the modification of the **name** attribute with the junction point does not result in mutually exclusive cookies.

Standard -j option operation:

If the following header:

```
Set-Cookie: ORDERID=123456; path=/orders
```

is received from a backend server for the **-j** junction `/sales`, then the modified header sent to the browser would be:

```
Set-Cookie: AMWEBJCT!/sales!ORDERID=123456; path=/
```

However, if another **Set-Cookie** header with the same **name** attribute, but a different **path** value, is received over the same junction, the modified header would result in the exactly same name and path information.

For example:

```
Set-Cookie: ORDERID=123456; path=/invoices
```

is modified to:

```
Set-Cookie: AMWEBJCT!/sales!ORDERID=123456; path=/
```

Because the second modified **Set-Cookie** header has the same cookie **name** and **path** as the first header, it overwrites the first. The junction point is not enough to uniquely identify the **Set-Cookie** header.

Solution:

You can configure a **-j** junction with the additional **-I** option to add the original **path** attribute value (for example, /orders) to the modified **name** of the cookie. Now the cookie names are unique. The following rules apply when using the **-I** option:

- If the **Set-Cookie** header from the junctioned server contains a **path** attribute, the value of that **path** is URI-encoded and used to modify the **name** attribute.
- If the **Set-Cookie** header from the junctioned server does not contain a **path** attribute, the **basedir** of the request URI is extracted, URI-encoded, and used to modify the **name** attribute.

  For example, if the client request was for /dir1/dir2/mypage.html, then the value /dir1/dir2 would be URI-encoded and used.
- The **Set-Cookie name** attribute is then modified using the junction point (unless this is the root junction "/") plus the URI-encoded **path** value (or **basedir** value).
- The value of the **Set-Cookie path** attribute is still converted to "/"

For example, if the following header:

```
Set-Cookie: ORDERID=123456; path=/orders
```

is received from a backend server for the **-j -I** junction /sales, then the modified header sent to the browser would be:

```
Set-Cookie: AMWEBJCT!/sales/orders!ORDERID=123456; path=/
```

HTTP transformations

You can modify HTTP requests and responses as they pass through WebSEAL with HTTP transformation rules. XSLT is used for this function. You can trigger specific rules with a Protected Object Policy (POP) or a request line pattern match.

WebSEAL administrators can configure the following modifications. You can apply these transformations to HTTP requests and HTTP responses (except where otherwise noted):

- Add a header
- Remove a header
- Modify an existing header
- Modify the URI (request only)
- Modify the method (request only)
- Modify the authorization object name (request only)
- Modify the HTTP version
- Modify the HTTP status code (response only)
- Modify the status reason (response only)
- Add a cookie
- Remove a cookie
- Modify an existing cookie
- Add a body (response only)
- Modify the ACL bits used in the authorization decision (request only)

Note:

1. It is not possible to modify the body of the request or response. Similarly, you cannot modify cookies or headers that are inserted by WebSEAL. For example, the **Host**, **iv-user** and **iv-creds** junction headers.

2. WebSEAL pages under the `lib/html` directory are referred to as *HTML server response pages*. These response pages are grouped into:
   - Account management pages.
   - Error message pages.

   You can configure the names of these response pages in the **[acnt-mgt]** stanza.

Extensible Stylesheet Language Transformation (XSLT)

Extensible Stylesheet Language (XSL) is the language used to specify rules, while Extensible Markup Language (XML) is the language used for the data that forms an input to the rules. The combination of XML and XSL provides a platform-independent way to express both inputs to the rules evaluator and rules themselves.

You can use XML to express complex data types in a structured and standard manner in text format. Using this text format, you can write rules for processing the XML data without having to cater to platform and programming language specifics.

You can use XSL Transformation (XSLT) rules to convert an XML document into another document in XML, PDF, HTML, or other format. A transformation rule must be defined as an XSL template in an XSL stylesheet. The rule must be written in a valid XSL template rule format. XSL possesses an inherent ability to analyze and evaluate XML data, which is becoming the standard for data representation in e-business models.

The HTTP requests and responses received by WebSEAL are expressed as XML objects and can be manipulated using XSL transformations. The attributes from the credential can be added to the XML object, but the credential attributes cannot be manipulated by the XSL transformations.

HTTP transformation rules

The HTTP requests and responses received by WebSEAL are expressed as XML objects and can be manipulated using XSL transformations.

You can use XSLT rules to represent the changes that you want to apply to the HTTP requests and responses as they pass through WebSEAL. WebSEAL uses the following two inputs for the HTTP transformations:

- An XML representation of the HTTP request or HTTP response. Attributes from the credential can be added to the XML object, but the credential attributes cannot be manipulated by the XSL transformations.
- An XSLT that determines how the request or response is modified.

The output from the transformation is an XML document that outlines the changes required to the HTTP request or HTTP response.

Note:

- The XSLT rules are contained in a rules file. If a rules file is changed, you must restart the WebSEAL server for the changes to take effect.
- Header fields must be URL encoded to avoid any XML issues. WebSEAL uses URL encoded header values during the transformation process.

- The XML representation of the HTTP Response object includes an element `<HTTPRequest>` that contains all of the elements of the HTTP Request. This allows request elements to be used as part of the transformation of the HTTP Response.

HTTP request objects

The following example shows the XML representation of a **HTTPRequest**:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPRequest>
<Scheme>https</Scheme>
 <RequestLine>
  <Method>GET</Method>
  <URI>/en/us/</URI>
  <Version>HTTP/1.1</Version>
 </RequestLine>
 <Headers>
  <Header name="User-Agent">curl%2F7.18.2%20(i486-pc-linux-gnu)%20libcurl
   %2F7.18.2%20OpenSSL%2F0.9.8g%20zlib%2F1.2.3.3%20libidn%2F1.8</Header>
  <Header name="Host">www.ibm.com</Header>
  <Header name="Accept">*%2F*</Header>
  </Headers>
 <Cookies>
  <Cookie name="PD-S-SESSION-ID">2_orQUNJCbjdxqIEdDPMXj31UHMXuU3hRCU</Cookie>
 </Cookies>
 <Credential>
    <Attribute name=AZN_CRED_PRINCIPAL_NAME>testuser</Attribute>
 </Credential>
</HTTPRequest>
```

HTTP response objects

The following example shows the XML Representation of a **HTTPResponse**:

```
<?xml version="1.0" encoding="UTF⌂-8"?>
<HTTPResponse>
    <Scheme>https</Scheme>
    <ResponseLine>
        <Version>HTTP/1.1</Version>
        <StatusCode>200</StatusCode>
        <Reason>OK</Reason>
    </ResponseLine>
    <Headers>
        <Header name="Date">Thu%2C%2016%20Sep%202010%2010%3A57%3A52%20GMT</Header>
        <Header name="Server">IBM_HTTP_Server</Header>
        <Header name="ContentLength">4096</Header>
        <Header name="Content-Type">text%2Fhtml%3Bcharset%3DUTF⌂-8</Header>
        <Header name="Content-Language">en-US</Header>
    </Headers>
    <Cookies>
        <Cookie name="PD-S-SESSION-ID">
            <Content>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Content>
            <Path>/</Path>
            <Domain>domainA.ibm.com</Domain>
            <Expires>Wed, 09 Jun 2021 10:18:14 GMT</Expires>
            <Secure>1</Secure>
            <HTTPOnly>0</HTTPOnly>
        </Cookie>
    </Cookies>
    <Credential>
        <Attribute name=AZN_CRED_PRINCIPAL_NAME>testuser</Attribute>
    </Credential>
    <HTTPRequest>
        <Scheme>https</Scheme>
        <RequestLine>
            <Method>GET</Method>
            <URI>/en/us/</URI>
            <Version>HTTP/1.1</Version>
        </RequestLine>
        <Headers>
            <Header name="User-Agent">curl%2F7.18.2%20(i486-pc-linux-gnu)%20libcurl
            %2F7.18.2%20OpenSSL%2F0.9.8g%20zlib%2F1.2.3.3%20libidn%2F1.8</Header>
            <Header name="Host">www.ibm.com</Header>
            <Header name="Accept">%2F</Header>
        </Headers>
        <Cookies>
            <Cookie name="PD-S-SESSION-ID">2_orQUNJCbjdxqIEdDPMXj31UHMXuU3hRCU</Cookie>
        </Cookies>
        <Credential>
            <Attribute name=AZN_CRED_PRINCIPAL_NAME>testuser</Attribute>
        </Credential>
    </HTTPRequest>
</HTTPResponse>
```

XSL transformation rules

A valid XSLT document can be used to transform the contents of the HTTP requests and responses.

The XSL transformation must output an XML document that defines the required changes. The output document contains a series of XML elements describing changes that must be made to the HTTP request or HTTP response.

Important: Author the XSLT documents carefully. Review and test the XSL transformation rules thoroughly before you implement it in a production environment. Incorrect syntax or badly formed XSLT might cause errors, or unexpected behavior.

The following table describes the base XML elements that WebSEAL requires in the transformed document:

Table 58. Base elements

Source document	Base XML element
HTTP Request	<HTTPRequestChange>
HTTP Response	<HTTPResponseChange>

The XSL transformation rules must handle the contents of the HTTP input. The content includes:

- The **ResponseLine**/**RequestLine** element.
- The **Headers** element.
- The **Cookies** element.
- The **Body** element. (*HTTPResponseChange only*)

If elements of the **RequestLine/ResponseLine** are included in the transformed XML document, WebSEAL applies the corresponding changes to the HTTP request/response.

Header elements require an **action** attribute in the XSLT document to determine how WebSEAL transforms the header. The available actions are:

1. **add** - adds a new header with a specific name and value.
2. **update** - updates the value of an existing header (if the header does not exist, it is added).
3. **remove** - removes the header with a specific name and value.

The **Cookie** elements require an **action** attribute in the XSLT document to determine how WebSEAL transforms the cookie. The available actions are:

1. **add** - adds a new cookie with the specified name and values.
2. **update** - updates the value of an existing cookie. (If the cookie does not exist, it is added).
3. **remove** - removes the cookie with a specific name.

Note: Cookies are represented differently in requests and responses. Only the response contains the attributes beyond name and value. When updating a cookie, specify the cookie name and the fields that you want to update. When adding a cookie, the minimum fields that you must specify are cookie name and value.

You can optionally include the **Body** element to insert a body into an HTTP response. The content of the **Body** must be URL encoded. WebSEAL decodes the content when it creates the response. WebSEAL replaces any existing body in the HTTP response with the new content that is provided in this **Body** element. This element does not require an action.

Note: It is not possible to replace the body content in requests.

The authorization object name can be customized based on an incoming request. But the object name can be changed only once. A subsequent change of the object name within the same request does not generate a new authorization decision. The **ObjectName** element value can be either a relative value or an absolute value. For standard junctions, the relative value format is `junction-name/resource`. For virtual junctions, the relative value format is `resource`. For standard junctions, the absolute value format is `/host_name-instance_name/junction-name/resource`. For virtual junctions, the absolute value format is `/host_name-instance_name/@virtual-junction-name/resource`. An absolute object name must begin with `/`. The customized authorization object names do not undergo dynamic URL processing.

The ACL bits which are used in the authorization decision for an incoming request can be customized using the `AclBits` element. This only takes affect if the HTTP transformation rule is invoked as a result of a match on the request line of the request. It does not have any impact if a POP was used to trigger the HTTP transformation rule.

XSLT Extensions

The HTTP transformation rules XSLT parser contains extensions that can be used when you are authoring HTTP transformation rules.

Name	Description	Usage
matches	Allows regular expressions to be used to match strings.	matches (input, pattern)
replaces	Allows regular expressions to be used to replace strings.	replaces (input, pattern, format)

- To use the XSLT extensions, the `http://xsltfunctions.isam.ibm.com` namespace must be defined.

  For example, `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" xmlns:external="http://xsltfunctions.isam.ibm.com">`

- The new extension functions, when qualified by the xslt functions namespace, can then be used in the XSLT rule.

  For example,

  To perform a regular expression match on the request URI:

  ```
  <xsl:template match="//HTTPRequest/RequestLine">
      <xsl:choose>
          <xsl:when test="external:matches(URI, '/index[12].html')">
              <URI>/index.html</URI>
          </xsl:when>
      </xsl:choose>
  </xsl:template>
  ```

  To perform a replace of the URI with a regular expression:

  ```
  <xsl:template match="//HTTPRequest/RequestLine">
      <xsl:choose>
          <xsl:when test="external:matches(URI, '^/scim/Users/.*')">
              <URI><xsl:value-of select="external:replace(URI, '/scim/Users/(.*)',
              '/v1/scim/Users/$1')"/></URI>
          </xsl:when>
      </xsl:choose>
  </xsl:template>
  ```

Replacing the HTTP response

You can create an entirely new HTTP response for a specified HTTP request. To achieve this behavior, modify an HTTP Request to produce an **HTTPResponseChange** XML document with **action="replace"** in the base XML element.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange action="replace">
<Version>HTTP/1.1</Version>
<StatusCode>503</StatusCode>
<Reason>Not Implemented</Reason>
<Body>%3Ch1%3EError%3C%2Fh1%3E%0A%3Cp%3EInvalid%20cookie%20%3C%2Fp%3E</Body>
</HTTPResponseChange>
```

See "Scenario 5: Providing a response to a known HTTP request" on page 461 for an example scenario.

The `HTTPResponseChange` document can result from HTTP Request or HTTP Response modifications.

If WebSEAL receives an **HTTPResponseChange** document with **action="replace"** as a result of an HTTP Request modification then WebSEAL:

- Interrupts the normal flow of processing and does not generate a response in the usual manner.
- Constructs a new HTTP response based on the provided XML document.

You can use this functionality to intercept particular HTTP Requests and provide a predefined response.

Note: Similarly, if WebSEAL receives an **HTTPResponseChange** document with **action="replace"** as a result of an HTTP Response modification then WebSEAL:

- Discards the original response.
- Uses the response that is specified in the XML.

Reprocessing considerations

If an HTTP transformation rule modifies the URI, host header, or object name of the request, WebSEAL reprocesses the transformed request.

This reprocessing ensures that the transformation does not bypass WebSEAL authorization. This behavior also means that administrators can define HTTP transformations rules to send requests to different junctions.

WebSEAL performs reprocessing (and authorization) on the first HTTP transformation only. Transformed requests undergo HTTP transformation again if there is an appropriate POP attached to the associated object space. See “Protected Object Policy (POP)” on page 452. However, WebSEAL does not reprocess the new requests that result from these subsequent transformations.

Configuration

For efficiency, it is important that transactions undergo the transformation process only when necessary. You can use the **[http-transformations]** stanza and associated POPs or request pattern matching rules to configure the objects that require HTTP transformation processing.

You can attach a POP to wanted objects in the object space. The POP must contain the name of a resource as an extended attribute. This resource name is matched against the name of an entry in the **[http-transformations]** configuration stanza. The value of this matching **[http-transformations]** stanza entry specifies the name of the file that contains the applicable XSLT rules.

The alternative to configuring a POP is to configure an **[http-transformations:***<resource_name>***]** stanza with a **request-match** entry. If a match with the pattern defined in the **request-match** entry is found, HTTP transformation processing is triggered.

Configuration file updates

You can use the **[http-transformations]** stanza to define HTTP transformation resources.

Use the **resource-name** configuration entry to define HTTP transformation resources. The configuration entries must be in the following format:

```
resource-name = resource-file
```

where:

resource-name
The name of the HTTP transformation resource.

resource-file
The name of the resource XSL file.

For more information, see [http-transformations] stanza.

You can use the **[http-transformations:** *<resource-name>* **]**stanza to house configuration that is specific to a particular HTTP transformation resource. The *<resource-name>* component of the stanza name must be changed to the actual name of the resource.

Use the **cred-attr-name** configuration entry in this stanza to define the credential attribute that is included in the XML input document and used when evaluating the HTTP transformation rule. The configuration entries must be in the following format:

```
 cred-attr-name = name
```

where:

name Name of the credential attribute.

Use the **request-match** configuration entry in the stanza to determine whether a request should be processed or not. The format of this configuration entry is:

```
request-match = {request|response}:<request-line>
```

Matching on resource URIs is case sensitive. To have a transformation do an insensitive match, in the **[http-transformations:** *<resource-name>* **]** stanza, set the property **match-case-insensitive** to `true`.

The first component of the configuration entry determines whether the processing is executed on the HTTP request or response. The second component contains the request line to be matched. A case-sensitive comparison is made between the configuration entry and the HTTP request line. The wildcard characters "*" and "?" can be used in the comparison. You can specify multiple entries if needed.

You also have the option of matching a request by using a host header. This option is useful when you need to selectively enable this functionality for a particular virtual host junction. To selectively match an entry based on a particular host header. prepend the *<request-line>* with the string `[<host>]`.

For more information, see [http-transformations] stanza.

Protected Object Policy (POP)

You can use a POP to enable the predefined XSLT resource for appropriate parts of the object space. This mechanism lets you specify the resources that need to undergo HTTP transformations.

The POP must have an extended attribute with the name HTTPTransformation and a value in the form:

- Request=*resource*, or
- Response=*resource*

where:

resource
Identifies one of the HTTP transformation resource names defined in the WebSEAL configuration file.

Only one HTTP transformation is performed on each request and response. If multiple **HTTPTransformation** attributes exist with Request or Response values, the first is chosen.

The following listing shows an example POP:

```
pdadmin sec_master> pop show http-transformation-pop attribute HTTPTransformation
HTTPTransformation
        Request=resource_a
        Response=resource_b
```

Example HTTP transformation scenarios

The following scenarios provide example input and output documents for the HTTP transformation process.

Scenario 1: Modifying the URI, headers, and cookies (HTTPRequest)

This scenario illustrates how to modify the RequestLine/URI element, as well as the header and cookie elements in the original HTTP request.

The following changes are made to the HTTP request in this example:

1. Append `/test` to the existing URI value.
2. Add a new header called **NAME_A** with the value **VALUE_A** if it does not exist.
3. Update the **NAME_B** header value to be **UPDATED_B**.
4. Remove the header called **NAME_C**.
5. Add a cookie called **MY_COOKIE**.
6. Update the **EXISTING_COOKIE** cookie content to be **NEW_COOKIE_VALUE**.

Input documents

The following sample input documents are used for this scenario:

HTTP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPRequest>
      <Scheme>https</Scheme>
 <RequestLine>
  <Method>GET</Method>
  <URI>/en/us/</URI>

  <Version>HTTP/1.1</Version>
 </RequestLine>
 <Headers>
  <Header name="Host">www.ibm.com</Header>
  <Header name="NAME_B">original_b</Header>
  <Header name="NAME_C">original_c</Header>
 </Headers>
 <Cookies>
  <Cookie name="EXISTING_COOKIE">2_orQUNJCbjdxqIEdDPMXj31UHMXuU3hRCU...</Cookie>
 </Cookies>
</HTTPRequest>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a request resource with an associated POP or a request line pattern match. See "Configuration" on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

 <!--Firstly, strip any space elements -->
 <xsl:strip-space elements="*" />

 <!--
  Perform a match on the root of the document. Output the required
  HTTPRequestChange elements and then process templates.
```

```
-->
<xsl:template match="/">
 <HTTPRequestChange>
  <xsl:apply-templates />

  <!-- Update the value of object name based on the URI -->
  <ObjectName>
   /combined-instances<xsl:value-of select="//HTTPRequest/RequestLine/URI"/>
  </ObjectName>

 </HTTPRequestChange>
</xsl:template>

<!--
 Do nothing to the Method.
-->
<xsl:template match="//HTTPRequest/RequestLine/Method" />

<!--
 Match on the URI. Append "test/" to the URI.
-->
<xsl:template match="//HTTPRequest/RequestLine/URI">
 <URI>
  <xsl:value-of select="node()" />
   test/
 </URI>
</xsl:template>

<!--
 Do nothing to the Version
-->
<xsl:template match="//HTTPRequest/RequestLine/Version" />

<!--
 Match on the Headers. Add a new header called NAME_A if
 it does not exist.
-->
<xsl:template match="//HTTPRequest/Headers">
 <xsl:choose>
  <xsl:when test="Header/@name='NAME_A'" />
  <xsl:otherwise>
   <Header action="add" name="NAME_A">
    VALUE_A
   </Header>
  </xsl:otherwise>
 </xsl:choose>
 <xsl:apply-templates select="//HTTPRequest/Headers/Header" />
</xsl:template>

<!-- Process the header elements -->
<xsl:template match="//HTTPRequest/Headers/Header">
 <xsl:choose>
  <!-- Update the value of the NAME_B header -->
  <xsl:when test="@name = 'NAME_B'">
   <Header action="update" name="NAME_B">
    UPDATED_B
   </Header>
  </xsl:when>
  <!-- Delete the NAME_C header -->
  <xsl:when test="contains(@name, 'NAME_C')">
   <Header action="remove" name="NAME_C">
    <xsl:value-of select="node()" />
   </Header>
  </xsl:when>
 </xsl:choose>
</xsl:template>
```

```
 <!--
  Match on the Cookies. Add a new cookie called MY_COOKIE if
  it does not exist.
 -->
 <xsl:template match="//HTTPRequest/Cookies">
  <xsl:choose>
   <xsl:when test="Cookie/@name='MY_COOKIE'" />
   <xsl:otherwise>
    <Cookie action="add" name="MY_COOKIE">
     MY_COOKIE_VALUE
    </Cookie>
   </xsl:otherwise>
  </xsl:choose>
  <xsl:apply-templates select="//HTTPRequest/Cookies/Cookie" />
 </xsl:template>

 <!-- Process the cookie elements -->
 <xsl:template match="//HTTPRequest/Cookies/Cookie">
  <xsl:choose>
   <!-- Update the value of the EXISTING_COOKIE cookie -->
   <xsl:when test="@name = 'EXISTING_COOKIE'">
    <Cookie action="update" name="EXISTING_COOKIE">
     NEW_COOKIE_VALUE
    </Cookie>
   </xsl:when>
  </xsl:choose>
 </xsl:template>
 <xsl:template match="//HTTPRequest/Credential" />
</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document outlines changes for WebSEAL to perform on the original HTTP request.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPRequestChange>
 <URI>/en/us/test/</URI>
 <Header action="add" name="NAME_A">VALUE_A</Header>
 <Header action="update" name="NAME_B">UPDATED_B</Header>
 <Header action="remove" name="NAME_C">ORIGINAL_C</Header>
 <Cookie action="add" name="MY_COOKIE">MY_COOKIE_VALUE</Cookie>
 <Cookie action="update" name="EXISTING_COOKIE">NEW_COOKIE_VALUE</Cookie>
 <ObjectName>/combined-instances/jct/resourceB</ObjectName>
</HTTPRequestChange>
```

Scenario 2: Modifying the headers only (HTTPResponse)

This scenario illustrates how to modify the headers in an HTTP Response. The XSLT in this example adds a new header called **RESPONSE_A** with the value **VALUE_A** if it does not exist.

Input documents

The following sample input documents are used for this scenario:

HTTP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponse>
 <ResponseLine>
  <Version>HTTP/1.1</Version>
  <StatusCode>200</StatusCode>
  <Reason>OK</Reason>
 </ResponseLine>
 <Headers>
  <Header name="Date">Thu%2C%2016%20Sep%202010%2010
   %3A57%3A52%20GMT</Header>
  <Header name="Server">IBM_HTTP_Server</Header>
  <Header name="Content-Type">text%2Fhtml%3Bcharset%3DUTF-8</Header>
  <Header name="Content-Language">en-US</Header>
 </Headers>
 <Cookies>
  <Cookie name="PD-S-SESSION-ID">
   <Content>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Content>
   <Path>/</Path>
   <Domain>domainA.com</Domain>
   <Expires>Wed, 09 Jun 2021 10:18:14 GMT</Expires>
   <Secure>1</Secure>
   <HTTPOnly>0</HTTPOnly>
  </Cookie>
 </Cookies>
</HTTPResponse>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a response resource with an associated POP. See "Configuration" on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

 <!--Firstly, strip any space elements -->
 <xsl:strip-space elements="*" />

 <!--
  Perform a match on the root of the document. Output the required
  HTTPResponseChange elements and then process templates.
 -->
  <xsl:template match="/">
   <HTTPResponseChange>
    <xsl:apply-templates />
   </HTTPResponseChange>
  </xsl:template>

 <!--
  Do nothing to the Version
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/Version" />

 <!--
  Do nothing to the StatusCode
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/StatusCode" />


 <!--
  Do nothing to the Reason
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/Reason" />

 <!--
  Match on the Headers. Add a new header called RESPONSE_A
  if it does not exist.
 -->
 <xsl:template match="//HTTPResponse/Headers">
  <xsl:choose>
   <xsl:when test="Header/@name='RESPONSE_A'"/>
   <xsl:otherwise>
    <Header action="add" name="RESPONSE_A">
     VALUE_A
    </Header>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

<!--
  Do nothing to the Cookies
 -->
 <xsl:template match="//HTTPResponse/Cookies" />

</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document outlines changes for WebSEAL to perform on the original HTTP response.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange>
 <Header action="add" name="RESPONSE_A">VALUE_A</Header>
</HTTPResponseChange>
```

Scenario 3: Modifying the ResponseLine/StatusCode only (HTTPResponse)

This scenario illustrates how to modify the StatusCode and Reason elements in an HTTP Response. The XSLT in this example makes the following updates:

- 503 status codes are changed to be 501.
- When a status code update occurs, the reason is also updated to reflect the change. The reason for the 501 status is “Not Implemented".

Input documents

The following sample input documents are used for this scenario:

HTTP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponse>
 <ResponseLine>
  <Version>HTTP/1.1</Version>
  <StatusCode>503</StatusCode>
  <Reason>Service Unavailable</Reason>
 </ResponseLine>
 <Headers>
  <Header name="Date">Thu%2C%2016%20Sep%202010%2010
   %3A57%3A52%20GMT</Header>
  <Header name="Server">IBM_HTTP_Server</Header>
  <Header name="Content-Type">text%2Fhtml%3Bcharset%3DUTF-8</Header>
  <Header name="Content-Language">en-US</Header>
 </Headers>
 <Cookies>
  <Cookie name="PD-S-SESSION-ID">
   <Content>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Content>
   <Path>/</Path>
   <Domain>domainA.com</Domain>
   <Expires>Wed, 09 Jun 2021 10:18:14 GMT</Expires>
   <Secure>1</Secure>
   <HTTPOnly>0</HTTPOnly>
  </Cookie>
 </Cookies>
</HTTPResponse>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a response resource with an associated POP. See “Configuration” on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

 <!--Firstly, strip any space elements -->
 <xsl:strip-space elements="*" />

 <!--
  Perform a match on the root of the document. Output the required
  HTTPResponseChange elements and then process templates.
 -->
  <xsl:template match="/">
   <HTTPResponseChange>
    <xsl:apply-templates />
   </HTTPResponseChange>
  </xsl:template>

 <!--
  Do nothing to the Version
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/Version" />

 <!--
  If the original StatusCode is 503 then update the
  StatusCode to 501.
 -->
 <xsl:template match="//HTTPResponse/ResponseLine">
        <xsl:choose>
            <xsl:when test="StatusCode='503'">
                <StatusCode>501</StatusCode>
                <Reason>Not Implemented</Reason>
            </xsl:when>
        </xsl:choose>
   </xsl:template>

   <xsl:template match="//HTTPResponse/Credential" />

 <!--
  Do nothing to the Headers.
 -->
 <xsl:template match="//HTTPResponse/Headers" />

       <!--
  Do nothing to the Cookies.
 -->
 <xsl:template match="//HTTPResponse/Cookies" />

</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document outlines changes for WebSEAL to perform on the original HTTP response.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange>
 <StatusCode>501</StatusCode>
 <Reason>Not Implemented</Reason>
</HTTPResponseChange>
```

Scenario 4: Modifying cookies only (HTTPResponse)

This scenario illustrates how to add, modify, and remove cookies in an HTTP Response. The XSLT in this example makes the following updates:

- Adds a cookie called **NEW_COOKIE**.
- Updates the **EXISTING_COOKIE** cookie domain to be **domainB.com**.
- Removes the cookie called **OLD_COOKIE**.

Input documents

The following sample input documents are used for this scenario:

HTTP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponse>
 <ResponseLine>
  <Version>HTTP/1.1</Version>
  <StatusCode>503</StatusCode>
  <Reason>Service Unavailable</Reason>
 </ResponseLine>
 <Headers>
  <Header name="Date">Thu%2C%2016%20Sep%202010%2010
   %3A57%3A52%20GMT</Header>
  <Header name="Server">IBM_HTTP_Server</Header>
  <Header name="Content-Type">text%2Fhtml%3Bcharset%3DUTF-8</Header>
  <Header name="Content-Language">en-US</Header>
 </Headers>
 <Cookies>
  <Cookie name="EXISTING_COOKIE">
   <Value>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Value>
   <Path>/</Path>
   <Domain>domainA.com</Domain>
   <Expires>Wed, 09 Jun 2021 10:18:14 GMT</Expires>
   <Secure>1</Secure>
   <HTTPOnly>0</HTTPOnly>
  </Cookie>
  <Cookie name="OLD_COOKIE">
   <Value>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Value>   <Path>/</Path>
   <Domain>domainA.com</Domain>
   <Expires>Mon, 07 Jun 2021 11:18:21 GMT</Expires>
   <Secure>1</Secure>
   <HTTPOnly>0</HTTPOnly>
  </Cookie>
 </Cookies>
</HTTPResponse>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a response resource with an associated POP. See "Configuration" on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

 <!--Firstly, strip any space elements -->
 <xsl:strip-space elements="*" />

 <!--
  Perform a match on the root of the document. Output the required
  HTTPResponseChange elements and then process templates.
 -->
  <xsl:template match="/">
```

```
   <HTTPResponseChange>
    <xsl:apply-templates />
   </HTTPResponseChange>
  </xsl:template>

 <!--
  Do nothing to the Version
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/Version" />

 <!--
  Do nothing to the StatusCode
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/StatusCode" />

 <!--
  Do nothing to the Reason
 -->
 <xsl:template match="//HTTPResponse/ResponseLine/Reason" />

 <!--
  Do nothing to the Headers.
 -->
 <xsl:template match="//HTTPResponse/Headers" />

 <!--
  Match on the Cookies. Add a new cookie called NEW_COOKIE if
  it does not exist.
 -->
 <xsl:template match="//HTTPResponse/Cookies">

 <xsl:choose>
  <xsl:when test="Cookie/@name='NEW_COOKIE'" />
  <xsl:otherwise>
   <Cookie action="add" name="NEW_COOKIE">
    <Content>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Content>
    <Path>/</Path>
    <Domain>domainA.com</Domain>
    <Expires>Mon, 07 Jun 2021 10:12:14 GMT</Expires>
    <Secure>1</Secure>
    <HTTPOnly>0</HTTPOnly>
   </Cookie>
  </xsl:otherwise>
 </xsl:choose>

 <!-- Update the value of the EXISTING_COOKIE cookie -->
 <xsl:if test="Cookie/@name='EXISTING_COOKIE'">
  <Cookie action="update" name="EXISTING_COOKIE">
   <Domain>domainB.com</Domain>
  </Cookie>
 </xsl:if>

 <!-- Delete the OLD_COOKIE cookie -->
 <xsl:if test="Cookie/@name='OLD_COOKIE'">
  <Cookie action="remove" name="OLD_COOKIE" />
 </xsl:if>

 </xsl:template>
</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document defines the changes for WebSEAL to perform on the original HTTP response.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange>
 <Cookie action="add" name="NEW_COOKIE">
  <Content>2_orQUNJCbjdxqIEdDPMXj31UiHMXuU3hRCUtpN7xe6J1xZhxt0</Content>
  <Path>/</Path>
  <Domain>domainA.com</Domain>
  <Expires>Mon, 07 Jun 2021 10:12:14 GMT</Expires>
  <Secure>1</Secure>
  <HTTPOnly>0</HTTPOnly>
 </Cookie>
 <Cookie action="update" name="EXISTING_COOKIE">
  <Domain>domainB.com</Domain>
 </Cookie>
 <Cookie action="remove" name="OLD_COOKIE"></Cookie>
</HTTPResponseChange>
```

Scenario 5: Providing a response to a known HTTP request

This scenario illustrates how a **HTTPResponseChange** document can be used to generate a response directly from a request. In this scenario, if a cookie with name '**invalid-cookie**' exists in the HTTP Request then the XSL transformation produces an HTTP Response that indicates an invalid cookie was detected.

Input documents

The following sample input documents are used for this scenario:

HTTP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPRequest>
<Scheme>https</Scheme>
 <RequestLine>
  <Method>GET</Method>
  <URI>/en/us/</URI>
  <Version>HTTP/1.1</Version>
 </RequestLine>
 <Headers>
  <Header name="User-Agent">curl%2F7.18.2%20(i486-pc-linux-gnu)%20libcurl
  %2F7.18.2%20OpenSSL%2F0.9.8g%20zlib%2F1.2.3.3%20libidn%2F1.8</Header>
  <Header name="Host">www.ibm.com</Header>
  <Header name="Accept">*%2F*</Header>
 </Headers>
 <Cookies>
  <Cookie name="invalid-cookie">0</Cookie>
 </Cookies>
</HTTPRequest>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a request resource with an associated POP. See “Configuration” on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">

 <!-- Firstly, strip any space elements -->
 <xsl:strip-space elements="*" />

 <!--
  Perform a match on the root of the document. Output the required
  HTTPRequestChange elements and then process templates.
 -->
 <xsl:template match="/">
  <HTTPRequestChange />
  <xsl:apply-templates />
 </xsl:template>

 <!--
  Do nothing with Method
 -->
 <xsl:template match="//HTTPRequest/RequestLine/Method" />

 <!--
 Do nothing with URI
 -->
 <xsl:template match="//HTTPRequest/RequestLine/URI"/>

 <!--
 Do nothing with Version
 -->
 <xsl:template match="//HTTPRequest/RequestLine/Version" />

 <!--
 Do nothing with Headers
 -->
 <xsl:template match="//HTTPRequest/Headers" />

 <!--
 Check for the presence of a cookie name 'invalid-cookie'
 -->
 <xsl:template match="//HTTPRequest/Cookies/Cookie">
  <xsl:choose>
   <xsl:when test="@name = 'invalid-cookie'">
    <HTTPResponseChange action="replace">
     <Version>HTTP/1.1</Version>
     <StatusCode>503</StatusCode>
     <Reason>Not Implemented</Reason>
     <Header name="Date" action="add">Thu%2C%2016%20Sep%202010%2010</Header>
     <Header name="Server" action="add">IBM_HTTP_Server</Header>
     <Header name="Content-Type" action="add">text%2Fhtml%3Bcharset%3DUTF-8</Header>
     <Header name="Content-Language" action="add">en-US</Header>
     <Body>%3Ch1%3EError%3C%2Fh1%3E%0A%3Cp%3EInvalid%20cookie%20%3C%2Fp%3E</Body>
    </HTTPResponseChange>
   </xsl:when>
  </xsl:choose>
 </xsl:template>
       <xsl:template match="//HTTPRequest/Credential" />

</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document defines the response that WebSEAL provides to the original HTTP request.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange action="replace">
 <Version>HTTP/1.1</Version>
 <StatusCode>503</StatusCode>
 <Reason>Not Implemented<Reason>
 <Header name="Date" action="add">Thu%2C%2016%20Sep%202010%2010</Header>
 <Header name="Server" action="add"></Header>
 <Header name="Content-Type" action="add">text%2Fhtml%3Bcharset%3DUTF-8</Header>
 <Header name="Content-Language" action="add">en-US</Header>
 <Body>%3Ch1%3EError%3C%2Fh1%3E%0A%3Cp%3EInvalid%20cookie%20%3C%2Fp%3E</Body>
</HTTPResponseChange>
```

Scenario 6: Adding a credential attribute value to the response

This scenario illustrates how a **HTTPResponseChange** document can be used to add a credential attribute value to the response.

Input documents

The following sample input document is used for this scenario:

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a response resource with an associated POP. See “Configuration” on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">

  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <HTTPResponseChange>
      <xsl:apply-templates />
    </HTTPResponseChange>
  </xsl:template>

  <xsl:template match="//HTTPResponse/ResponseLine/StatusCode"/>
  <xsl:template match="//HTTPResponse/ResponseLine/Reason"/>
  <xsl:template match="//HTTPResponse/ResponseLine/Version"/>

  <xsl:template match="//HTTPResponse/Credential/Attributes/Attribute">
    <xsl:choose>
      <xsl:when test="contains(@name, 'AZN_CRED_PRINCIPAL_NAME')">
        <Header action="add" name="hdr-2"><xsl:value-of select="node()"/></Header>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is output from the XSL transformation. This document defines the response that WebSEAL provides to the original HTTP request.

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange action="replace">
 <Version>HTTP/1.1</Version>
 <StatusCode>503</StatusCode>
 <Reason>Not Implemented<Reason>
 <Header name="principal-name" action="add">testuser</Header>
</HTTPResponseChange>
```

Scenario 7: Adding response headers based upon values of the request headers

This scenario illustrates how to add headers to the HTTP Response if certain conditions are met by the value of the HTTP Request headers. An example of the usefulness of this is adding CORS headers.

The example shows how to add the CORS header "Access-Control-Allow-Origin" if the value of the request header Origin contains the value "test.com". Perform the following steps to add the response headers:

1. Check the HTTP Request header "Origin" to see if it contains "test.com".
2. If it does then add a new HTTP Response header `Access-Control-Allow-Origin = <Value of the HTTP Request Origin header>`.
3. If it does not contain "test.com", do nothing.

Input documents

The following sample input documents are used for this scenario:

HTTP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponse>
   <Scheme>https</Scheme>
   <ResponseLine>
       <Version>HTTP/1.1</Version>
       <StatusCode>200</StatusCode>
       <Reason>OK</Reason>
    </ResponseLine>
    <Headers>
        <Header name="Server">IBM_HTTP_Server</Header>
    </Headers>
    <HTTPRequest>
        <Scheme>https</Scheme>
        <RequestLine>
            <Method>GET</Method>
            <URI>/en/us/</URI>
            <Version>HTTP/1.1</Version>
         </RequestLine>
         <Headers>
             <Header name="Origin">myserver.test.com</Header>
         </Headers>
    </HTTPRequest>
</HTTPResponse>
```

XSLT Rules

Note: These rules must be stored in an XSL document that is defined as a response resource with an associated POP. See "Configuration" on page 451.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
       <!-- Firstly, strip any space elements -->
        <xsl:strip-space elements="*" />
       <!--
               Perform a match on the root of the document. Output the required
               HTTPResponseChange elements and then process templates.
       -->
                  <xsl:template match="/">
                   <HTTPResponseChange>
                    <xsl:apply-templates />
                   </HTTPResponseChange>
                  </xsl:template>
       <!--
               Do nothing to the version
       -->
       <xsl:template match="//HTTPResponse/ResponseLine/Version" />
       <!--
               Do nothing to the status code
       -->
       <xsl:template match="//HTTPResponse/ResponseLine/StatusCode" />
       <!--
               Do nothing to the reason
       -->
```

```
        <xsl:template match="//HTTPResponse/ResponseLine/Reason" />
        <!--
                Do nothing to the response headers
        -->
        <xsl:template match="//HTTPResponse/Headers" />
        <!--
                Do nothing to the cookies
        -->
        <xsl:template match="//HTTPResponse/Cookies" />
        <!--
                Find the Origin header from the Request. Add CORS header
              Access-Control-Allow-Origin with the value of the Origin header
              But only if it has the value test.com
        -->
        <xsl:template match="//HTTPResponse/HTTPRequest/Headers/Header">
             <xsl:choose>
                 <xsl:when test="@name='origin' and contains(text(),'test.com')">
                 <Header action="add" name="Access-Control-Allow-Origin">
                     <xsl:value-of select="node()"/>
                 </Header>
                 </xsl:when>
             </xsl:choose>
         </xsl:template>
</xsl:stylesheet>
```

Output XML document

In this scenario, the following XML document is the output from the XSL transformation. This document outlines changes for WebSEAL to perform on the original HTTP response

```
<?xml version="1.0" encoding="UTF-8"?>
<HTTPResponseChange>
      < Header action="add" name="Access-Control-Allow-Origin">myserver.test.com</Header>
</HTTPResponseChange>
```

.

Transformation errors

Most errors are printed in the server log or returned to the browser as an error page.

Note: For more detailed output, you can use the **pdweb.http.transformation** component to trace the HTTP transformation processing. This component traces the header information in the request, which might contain sensitive information. For example, a BA header.

Invalid rules file

If an invalid rules file is supplied to WebSEAL as part of the configuration then:

- The server will fail to start.
- An appropriate error will be logged.

Undefined resource

If you specify a resource in a POP that is not defined in the WebSEAL configuration file, a warning message will be printed in the server log.

HTTP transformation error

If an error occurs during HTTP transformation processing, a 500 Internal Server Error is returned to the browser.

WebSEAL error responses

WebSEAL error responses (not related to HTTP transformation processing) will be transformed if there is an appropriate POP attached to the junction or object that caused the error.

Microsoft RPC over HTTP

RPC over HTTP is a Microsoft protocol that allows Microsoft Outlook clients to access Microsoft Exchange servers over HTTP. The RPC over HTTP protocol uses one HTTP connection for request data and one HTTP connection for response data. These HTTP connections are long lived. The protocol tunnels multiple requests/responses in a single HTTP request.

A typical usage scenario is an Outlook user outside the corporate network who wants to connect to an internal Exchange server. This connection is achieved by accessing a reverse proxy (such as WebSEAL) using HTTP. The HTTP connection is terminated inside the corporate network and a configured IIS relays the RPC commands to the Exchange server.

RPC over HTTP support in WebSEAL

WebSEAL can act as the reverse proxy if you are using the RPC over HTTP version 2 protocol. See “Microsoft RPC over HTTP support” on page 38.

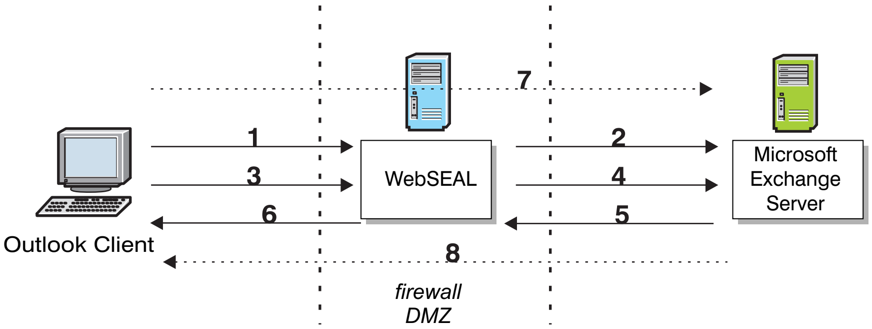


Figure 38. WebSEAL RPC over HTTP

Figure 38 illustrates a typical scenario using the RPC over HTTP protocol.

1. Microsoft Outlook makes an **RPC_IN_DATA** request to WebSEAL. The request contains the user name and password in a BA Header.
2. WebSEAL authenticates and authorizes the request. WebSEAL passes the request (including the BA header) to Microsoft Exchange.
3. Outlook makes an **RPC_OUT_DATA** request to WebSEAL. The request contains the user name and password in a BA Header.
4. WebSEAL authenticates and authorizes the request. WebSEAL passes the request (including the BA header) to Exchange.
5. Exchange sends a **200 OK** response back to WebSEAL.

6. WebSEAL forwards the response back to Outlook.
7. Outlook sends RPC data to Exchange via WebSEAL on the **RPC_IN_DATA** connection.
8. Exchange sends RPC data to Outlook via WebSEAL on the **RPC_OUT_DATA** connection.

Junction configuration

To use WebSEAL as a reverse proxy for RPC over HTTP requests between Outlook and Exchange, you must use a transparent path junction or a virtual host junction. When issuing an RPC over HTTP request, the Outlook client tries to access the URI `/rpc/rpcproxy.dll` on the junctioned IIS server that is configured to communicate with the Exchange server.

To authenticate the user to WebSEAL and the Exchange server, you must use the **-b ignore** parameter when creating the junction. This parameter ensures that the BA header used by WebSEAL for authentication is also used to authenticate to the IIS server that communicates with the Exchange server. For more details, see "Authentication limitations" on page 468.

You must use an SSL junction for this configuration; Outlook does not support HTTP when using BA authentication.

Transparent path junctions

The following command illustrates how to create a transparent path junction:

```
server task instance_name-webseald-host_name create -t ssl
-h exchange_host -p exchange_port -b ignore -x /rpc
```

where:

instance_name-**webseald-***host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the server list command.

exchange_host
: Specifies the DNS host name or IP address of the Exchange server.

exchange_port
: Specifies the TCP port of the Exchange server. The default value is 80 for TCP junctions and 443 for SSL junctions.

Virtual host junctions

The following command illustrates how to create a virtual host junction:

```
server task instance_name-webseald-host_name virtualhost create -t ssl
-h exchange_host -p exchange_port -v virtual_host -b ignore exchange
```

where:

instance_name-**webseald-***host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the server list command.

exchange_host
: Specifies the DNS host name or IP address of the Exchange server.

exchange_port
Specifies the TCP port of the Exchange server. The default value is 80 for TCP junctions and 443 for SSL junctions.

virtual_host
Specifies the value of the Host header of the request sent to the Exchange server.

POP configuration

You must configure a POP to control request and response streaming on the junction used for communication between Outlook and Exchange:

1. Create a POP that sets the **response-buffer-control** and **request-buffer-control** attributes to **bypass**. For example:

   ```
   pdadmin> pop create streaming
   pdadmin> pop modify streaming set attribute response-buffer-control bypass
   pdadmin> pop modify streaming set attribute request-buffer-control bypass
   ```

2. Attach this POP to the junction that you are using for the RPC over HTTP communication.

   Transparent Path Junction example:

   ```
   pdadmin> pop attach /WebSEAL/webseal-server/rpc streaming
   ```

   Virtual Host Junction example:

   ```
   pdadmin> pop attach /WebSEAL/webseal-server/@exchange streaming
   ```

For details, see "Bypassing buffering on a per-resource basis" on page 417.

Authentication limitations

The credentials presented to WebSEAL by Outlook are used in the RPC data and the authentication to Exchange. The WebSEAL user credentials must match the AD credentials for Exchange.

You must configure the WebSEAL server to connect to Exchange using a BA header over HTTPS. The value for **ba-auth** in the **[ba]** stanza must be set to `https`.

```
ba-auth = https
```

The BA credentials for the user must be passed through the junction to the Exchange server. Specify the **-b ignore** option when creating the junction

Note:

1. Microsoft NT LAN Manager (NTLM) authentication is not supported.
2. The **/RpcWithCert** endpoint is not supported. It expects client certificate authentication. WebSEAL cannot authenticate using a client certificate for the configured junction.

Timeout considerations

When you configure WebSEAL as a reverse proxy for RPC over HTTP requests, set the following values in the **[server]** stanza of the WebSEAL configuration file:

- `client-connect-timeout`
- `intra-connection-timeout`

These timeout configuration entries affect the Outlook client connection. To keep the connection open longer for data streaming, use large timeout values. For example:

```
[server]
client-connect-timeout = 36000
intra-connection-timeout = 36000
```

When these timeout limits are reached, the Outlook client renegotiates a connection to the Exchange server.

WebSEAL server log errors

Due to the nature of RPC over HTTP, socket errors might be printed in the WebSEAL server log.

Note: The **server-log-cfg** configuration entry in the **[logging]** stanza specifies the logging agent configuration.

For example:

```
2010-10-26-05:45:25.836+10:00I----- 0x38AD5424 webseald ERROR wiv socket
WsSslListener.cpp 1737 0xafb3fb90
DPWIV1060E   Could not read from socket (406)
2010-10-26-05:45:25.838+10:00I----- 0x38AD5425 webseald ERROR wiv socket
WsSslListener.cpp 1658 0xafb3fb90
DPWIV1061E   Could not write to socket (406)
```

These error messages are expected when request/response streaming is active and the connection is closed before the full content-length has been received.

You can turn off these error messages by setting the **suppress-client-ssl-errors** in the **[ssl]** stanza of the WebSEAL configuration file to `true`. See "Specifying the WebSEAL host name" on page 25 for details about the location of this configuration file.

Worker thread consideration

Microsoft Outlook creates multiple HTTP connections when communicating with Microsoft Exchange using RPC over HTTP. Each Outlook client connection uses multiple worker threads in WebSEAL. The number of worker threads that each Outlook client uses in WebSEAL can exceed 10. These threads are held for the length of time that the Outlook client remains connected to the Exchange server. The use of numerous worker threads for every RPC over HTTP client connection via WebSEAL can lead to worker thread starvation.

WebSEAL has a limited number of worker threads available. Limited threads limits the number of clients that can access WebSEAL at any point in time. When deploying WebSEAL for use with RPC over HTTP in an Outlook and Exchange environment, consider the number of worker threads being used.

Command option summary: standard junctions

The **pdadmin** utility provides an interactive command-line prompt from which you can perform WebSEAL junction tasks. This section describes the **pdadmin server task** command for creating standard WebSEAL junctions. See the Web command reference topics for complete syntax information for the **pdadmin** utility.

Using pdadmin server task to create junctions

Before you begin

Before using **pdadmin**, you must login to a secure domain as a user with administration authorization, such as **sec_master**.

For example:

```
pdadmin> login
Enter User ID: sec_master
Enter Password:
pdadmin>
```

Procedure

To create WebSEAL junctions, you use the **pdadmin server task create** command:

```
pdadmin> server task instance_name-webseald-host_name create options
```

For example, if the configured name of a single WebSEAL instance is **web1** , installed on a host named **www.pubs.com**, the complete server name would be expressed as follows:

```
web1-webseald-www.pubs.com
```

Use the **pdadmin server list** command to display the correct format of the complete server name:

```
pdadmin> server list
web1-webseald-www.pubs.com
```

For more information, see the reference page for **pdadmin server task create** in Chapter 13, "Command reference," on page 627 or the Web command reference topics.

Server task commands for junctions

The following junction commands are available with **pdadmin server task**:

Command	Description
add	Add an additional server to an existing junction point. **Syntax:** `add -h host-name options junction-point` See "Addition of server to an existing junction" on page 478.
create	Create a new junction for an initial server. **Syntax:** `create -t type -h host-name options junction-point` See "Creation of a junction for an initial server" on page 471.
delete	Remove the specified junction point. **Syntax:** `delete junction-point`

Command	Description
jmt load **jmt clear**	The **jmt load** command provides WebSEAL with junction mapping table data (`jmt.conf`) to handle processing of dynamically generated server-relative URLs. The **jmt clear** command removes junction mapping table data from WebSEAL.
list	List all configured junction points on this server. **Syntax:** `list`
offline	Places the server located at this junction in an offline operational state. No additional requests are sent to the specified server. If a server is not specified, then all servers located at this junction are placed in an offline operational state. **Syntax:** `offline [-i` *server_uuid*`]` *junction_point*
online	Places the server located at this junction in an online operational state. The server now resumes normal operation. If a server is not specified, then all servers located at this junction are placed in an online operational state. **Syntax:** `online [-i` *server_uuid*`]` *junction_point*
remove	Remove the specified server from a junction point. **Syntax:** `remove –i` *server-id junction-point* Use the **show** command to determine the ID of a particular server.
show	Display the details of a junction. **Syntax:** `show` *junction-point*
throttle	Places the server located at this junction in a throttled operational state. While in this state, only requests from pre-established user sessions are processed. If a server is not specified, then all servers located at this junction are placed in a throttled operational state. **Syntax:** `throttle [-i` *server_uuid*`]` *junction_point*

Creation of a junction for an initial server

You can create a new junction with the create command.

Operation: Creates a new junction point and junctions an initial server.

Syntax:

```
create -t type -h host-name options junction-point
```

–t type

Type of junction. One of: **tcp**, **ssl**, **tcpproxy**,**sslproxy**, **local**, **mutual**. This parameter is required.

Default port for –t tcp is **80**. Default port for –t ssl is **443**.

–h host-name

The DNS host name or IP address of the target back-end server. This parameter is required.

options

See Table 59.

junction-point

The name of the junction point. This parameter is required.

See "Standard WebSEAL junction configuration" on page 377.

Table 59. Options on the create command for creating junctions

Junction type	Parameter	Description
Standard junction types	–a *address*	Specifies the local IP address that WebSEAL uses when communicating with the target back-end server. If this option is not provided, WebSEAL uses the default address as determined by the operating system. If you supply an address for a particular junction, WebSEAL will be modified to bind to this local address for all communication with the junctioned server.
Standard junction types	-E description	A description of the junction.
Standard junction types	–f	Forces the replacement of an existing junction. See "Forcing a new junction" on page 401.
Standard junction types	–i	WebSEAL server treats URLs as case insensitive. See "Support for URLs as not case-sensitive" on page 413.
Standard junction types	–q *location*	Provides WebSEAL with the correct name of the **query_contents** program file and where to find the file. By default, the Windows file is called `query_contents.exe` and the UNIX file is called `query_contents.sh`. By default, WebSEAL looks for the file in the `cgi_bin` directory of the back-end Web server. Required for back-end Windows and UNIX Web servers. See "Installing and configuring query_contents on Windows-based Web servers" on page 391.
Standard junction types	–T *resource/ resource-group*	Name of GSO resource or resource group. Required for and used only with **–b gso** option. See "Configuring a GSO-enabled WebSEAL junction" on page 527.
Standard junction types	–w	Windows filesystem support. See "Junctions to Windows file systems" on page 414.

Table 59. Options on the create command for creating junctions (continued)

Junction type	**Parameter**	**Description**
TCP and SSL junction types	–p *port*	TCP port of the back-end third-party server. Default is **80** for TCP junctions; **443** for SSL junctions. See “Creating TCP type standard junctions” on page 378 and “Creating SSL type standard junctions” on page 378.
Stateful junctions See “Stateful junctions” on page 397.	–s	Specifies that the junction should support stateful applications. By default, junctions are *not* stateful.
Stateful junctions See “Stateful junctions” on page 397.	–u *UUID*	Specifies the UUID of a back-end server connected to WebSEAL using a stateful junction (–s).
Mutual junctions See “Stateful junctions” on page 397.	–p *HTTP port*	HTTP port of the back-end third-party server. See “Creating mutual junctions” on page 379.
Mutual junctions See “Stateful junctions” on page 397.	–P *HTTPS port*	HTTPS port of the back-end third-party server. See “Creating mutual junctions” on page 379.
Mutual authentication over Basic Authentication and SSL certificates See “Mutually authenticated SSL junctions” on page 393.	–B	WebSEAL uses BA header information to authenticate to back-end server. Requires –U, and –W options.
Mutual authentication over Basic Authentication and SSL certificates See “Mutually authenticated SSL junctions” on page 393.	–D "*DN*"	Specifies the distinguished name of back-end server certificate. This value, matched with actual certificate DN enhances authentication.

Table 59. Options on the create command for creating junctions (continued)

Junction type	Parameter	Description
Mutual authentication over Basic Authentication and SSL certificates See “Mutually authenticated SSL junctions” on page 393.	–K "*key-label*"	Key label of WebSEAL's client-side certificate, used to authenticate to back-end server.
Mutual authentication over Basic Authentication and SSL certificates See “Mutually authenticated SSL junctions” on page 393.	–U "*username*"	WebSEAL user name. Use with –B to send BA header information to back-end server.
Mutual authentication over Basic Authentication and SSL certificates See “Mutually authenticated SSL junctions” on page 393.	–W "*password*"	WebSEAL password. Use with –B to send BA header information to back-end server.
Proxy junction (requires –t tcpproxy or –t sslproxy) See “TCP and SSL proxy junctions” on page 395.	–H *host-name*	The DNS host name or IP address of the proxy server.
Proxy junction (requires –t tcpproxy or –t sslproxy) See “TCP and SSL proxy junctions” on page 395.	–P *port*	The TCP port of the proxy server.

Table 59. Options on the create command for creating junctions (continued)

Junction type	Parameter	Description
Supply identity information in HTTP headers	–b *BA-value*	Defines how the WebSEAL server passes client identity information in HTTP basic authentication (BA) headers to the back-end server. One of: **filter** (default), **ignore**, **supply**, **gso** See "Client identity in HTTP BA headers" on page 516.
Supply identity information in HTTP headers	–c *header-types*	Inserts client identity information specific to Security Access Manager in HTTP headers across the junction. The *header-types* argument can include any combination of the following Access Manager HTTP header types: **iv-user**, **iv-user-l**, **iv-groups**, **iv-creds**, **all**. See "Client identity in HTTP headers (–c)" on page 520.
Supply identity information in HTTP headers	–e *encoding-type*	Specifies the encoding to use when generating HTTP headers for junctions. This encoding applies to headers that are generated with both the **–c** junction option and tag-value. Possible values for encoding are: • utf8_bin • utf8_uri • lcp_bin • lcp_uri See "UTF-8 encoding for HTTP header data" on page 416.
Supply identity information in HTTP headers	–I	Cookie handling: -I ensures unique Set-Cookie header name attribute. See "Cookie handling: -I ensures unique Set-Cookie name attribute" on page 444.
Supply identity information in HTTP headers	–j	Supplies junction identification in a cookie to handle script generated server-relative URLs. See "Modification of server-relative URLs with junction cookies" on page 435.
Supply identity information in HTTP headers	–J {trailer,inhead, onfocus,xhtml10}	Controls the junction cookie JavaScript block. Use **–J trailer** to append (rather than prepend) the junction cookie JavaScript to HTML page returned from back-end server. Use **–J inhead** to insert the JavaScript block between <head> </head> tags for HTML 4.01 compliance. Use **–J onfocus** to use the onfocus event handler in the JavaScript to ensure the correct junction cookie is used in a multiple-junction/multiple-browser-window scenario. Use **–J xhtml10** to insert a JavaScript block that is HTML 4.01 and XHTML 1.0 compliant. For complete details on this option, see "Control on the junction cookie JavaScript block" on page 436.
Supply identity information in HTTP headers	–k	Sends session cookie to back-end portal server. See "Passing of session cookies to junctioned portal servers" on page 412.

Table 59. Options on the create command for creating junctions (continued)

Junction type	Parameter	Description
Supply identity information in HTTP headers	–n	Specifies that no modification of the names of non-domain cookies are to be made. Use when client-side scripts depend on the names of cookies. By default, if a junction is listed in the JMT or if the **-j** junction option is used, WebSEAL prepends the names of non-domain cookies that are returned from the junction to with: `AMWEBJCT_`*`junction_point`*`_` See "Preservation of cookie names" on page 443.
Supply identity information in HTTP headers	–r	Inserts incoming IP address in HTTP header across the junction. See "Client IP addresses in HTTP headers (–r)" on page 523.
Junction fairness See "Per-junction allocation of worker threads for junctions" on page 44.	–l *percent-value*	Defines the soft limit for consumption of worker threads.
Junction fairness See "Per-junction allocation of worker threads for junctions" on page 44.	–L *percent-value*	Defines the hard limit for consumption of worker threads.
WebSphere single signon (LTPA) junctions See "LTPA overview" on page 529.	–A	Enables junctions to support LTPA cookies (tokens). LTPA version 1 cookies (LtpaToken) and LTPA version 2 cookies (LtpaToken2) are both supported. LTPA version 1 cookies are specified by default. LTPA version 2 cookies must be specified with the additional **-2** option. Also requires –F, and –Z options.
WebSphere single signon (LTPA) junctions See "LTPA overview" on page 529.	–2	Used with the **-A** option, this option specifies that LTPA version 2 cookies (LtpaToken2) are used. The **-A** option without the **-2** option specifies that LTPA version 1 cookies (LtpaToken) are used.

Table 59. Options on the create command for creating junctions (continued)

Junction type	**Parameter**	**Description**
WebSphere single signon (LTPA) junctions See "LTPA overview" on page 529.	–F "*keyfile*"	Name of the key file used to encrypt LTPA cookie data. Only valid with **-A** option.
WebSphere single signon (LTPA) junctions See "LTPA overview" on page 529.	–Z "*keyfile-password*"	Password for the key file used to encrypt LTPA cookie data. Only valid with **-A** option.
Federation Runtime junctions "Single sign-on with the Security Token Service" on page 513	-Y	Enables the Federation Runtime for the junction. **NOTE:** Before using this option, you must first configure the WebSEAL configuration files to support the Federation Runtime single sign-on over junctions.
WebSEAL-to-WebSEAL SSL junctions See "WebSEAL-to-WebSEAL junctions over SSL" on page 396.	–C	Mutual authentication between a front-end WebSEAL server and a back-end WebSEAL server over SSL. Requires **–t ssl** or **–t sslproxy** type.
Forms single signon See "Forms single sign-on concepts" on page 532.	–S *file_name*	Name of the forms single signon configuration file.
Virtual hosts See "Standard junctions to virtual hosts" on page 415.	–v *virtual-host-name*[:*HTTP-port*]	Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. For mutual junctions this value corresponds to the virtual host which is used for HTTP requests. You use **–V** when the back-end junction server expects a **Host** header because you are junctioning to one virtual instance of that server. The default HTTP header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.

Table 59. Options on the create command for creating junctions (continued)

Junction type	Parameter	Description
Virtual hosts See "Standard junctions to virtual hosts" on page 415.	–V *virtual-host-name*[:*HTTPS-port*]	Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. The value corresponds to the virtual host which is used for HTTPS requests. Only used for mutual junctions. You use **–V** when the back-end junction server expects a **Host** header because you are junctioning to one virtual instance of that server. The default HTTPS header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.
Transparent junctions See "Transparent path junctions" on page 381.	–x	Creates a transparent path junction.

Addition of server to an existing junction

Operation: Adds an additional server to an existing junction point.

Syntax:

```
add -h host-name options junction-point
```

–h *host-name*
The DNS host name or IP address of the target back-end server to add. This parameter is required.

options
See Table 60.

junction-point
The name of the junction point. This parameter is required.

See "Standard WebSEAL junction configuration" on page 377.

Table 60. Options on the add command for adding a server to a junction

Junction type	Parameter	Description
Standard junction types	–a *address*	Specifies the local IP address that WebSEAL uses when communicating with the target back-end server. If this option is not provided, WebSEAL uses the default address as determined by the operating system. If you supply an address for a particular junction, WebSEAL binds to this local address for all communication with the junctioned server.
Standard junction types	–i	WebSEAL server treats URLs as case insensitive. See "Support for URLs as not case-sensitive" on page 413.

Table 60. Options on the add command for adding a server to a junction (continued)

Junction type	Parameter	Description
Standard junction types	–q *url*	Relative path for **query_contents** script. By default, WebSEAL looks for **query_contents** in /cgi_bin/. If this directory is different or the **query_contents** file name is different, use this option to indicate to WebSEAL the new URL to the file. Required for back-end Windows servers. See "Installing and configuring query_contents on Windows-based Web servers" on page 391.
Standard junction types	–w	Windows filesystem support. See "Junctions to Windows file systems" on page 414.
TCP and SSL junction types	–p *port*	TCP port of the back-end third-party server. Default is **80** for TCP junctions; **443** for SSL junctions. See "Creating TCP type standard junctions" on page 378 and "Creating SSL type standard junctions" on page 378.
Mutual junction types	–p *HTTP-port*	HTTP port of the back-end third-party server. See "Creating mutual junctions" on page 379.
Mutual junction types	–P *HTTPS-port*	HTTPS port of the back-end third-party server. See "Creating mutual junctions" on page 379.
Stateful junctions See "Stateful junctions" on page 397.	–u *UUID*	Specifies the UUID of a back-end server connected to WebSEAL via a stateful junction (–s).
Mutual authentication over SSL See "Mutually authenticated SSL junctions" on page 393.	–D "*DN*"	Specifies distinguished name of back-end server certificate. This value, matched with an actual certificate DN, enhances authentication.
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–H *host-name*	DNS host name or IP address of the proxy server.
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–P *port*	The TCP port of the proxy server.

Table 60. Options on the add command for adding a server to a junction (continued)

Junction type	Parameter	Description
Virtual hosts See “Standard junctions to virtual hosts” on page 415.	–v *virt-host-name*	Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. For mutual junctions this value corresponds to the virtual host which is used for HTTP requests. You use **–V** when the back-end junction server expects a host name header because you are junctioning to one virtual instance of that server. The default HTTP header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.
Virtual hosts See “Standard junctions to virtual hosts” on page 415.	–V *virt-host-name*	Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. The value corresponds to the virtual host which is used for HTTPS requests. Only used for mutual junctions. You use **–V** when the back-end junction server expects a host name header because you are junctioning to one virtual instance of that server. The default HTTPS header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.

Chapter 8. Virtual Hosting

Use the virtual hosting feature to maintain more than one server on one machine. You can use virtual hosting to run multiple web services, each with a different host name and URL.

Standard WebSEAL junctions

WebSEAL junctions create a scalable website. When demands on a website grow, you can add more servers to expand the capabilities of the site.

Security Access Manager is a product for authenticating and authorizing requests to protected web application servers. The protection of web servers is achieved by blocking direct access to the servers, and then routing the requests to WebSEAL. WebSEAL authorizes each request and, if allowed, passes on the request to the protected web server. WebSEAL returns any response back to the client.

WebSEAL acts as a single host web server. WebSEAL merges all of the server document spaces into a single document space. By merging document spaces, WebSEAL can protect many web servers and still act as a single host server. The term *junctioning* describes the action of merging a web server's document space into WebSEAL's single unified virtual document space.

For successful communication across junctions, WebSEAL must filter absolute and relative server URLs in HTML response documents that are returned from the protected web servers. Filtering correctly displays the URLs when they are viewed as part of WebSEAL's single host document space. The junction feature of WebSEAL changes the server and path information that must be used to access resources on junctioned systems. A link to a resource on a junctioned server can succeed only if the URL contains the identity of the junction.

Challenges of URL filtering

As an administrator, you must understand the challenges when URL filtering is used so that you can configure WebSEAL to parse HTML content correctly.

WebSEAL supports several solutions for filtering and processing URLs that are returned in responses from back-end junctioned application servers. All the solutions require WebSEAL to search the HTML for URLs, which can be complex.

The ability of embedded JavaScript to avoid filtering by dynamically generating URLs on the client-side is also complex. See “Modification of URLs to junctioned resources” on page 419 to understand how WebSEAL handles URL filtering over standard junctions.

Virtual hosting

Virtual hosting is a web server that appears as more than one host on the Internet; the apparent host names distinguishes one host from another one. Using virtual hosting you can run multiple web services, each with a different host name and URL, that appear to be separate sites.

For example, the following two virtual hosts can be on the same computer:

- **www.exampleA.com**
- **www.exampleB.com**

The virtual hosts are internally configured to use unique sections of the host server's document space. A user must reference the virtual hosts only by their individual domain names. Users do not need to know any extra path information.

Access to resources that use virtual hosting is possible because the HTTP/1.1 specification requires client browsers to include in any request the HTTP **Host** header. The **Host** header contains the host name of the server where the requested resource is located.

Virtual host junction solution

WebSEAL supports virtual hosting. Through virtual host junctions, it can eliminate the limitations of URL filtering.

WebSEAL can use virtual host junctions to communicate with local or remote virtual hosts. WebSEAL uses the HTTP **Host** header in client requests to direct those requests to the appropriate document spaces on junctioned servers or on the local computer.

A user can access resources directly with the host name of the junctioned server (`http://`*`protected-server/resource`*). The use can do this, rather than indirectly with the host name of the WebSEAL server and a potentially modified resource path (`http://`*`webseal/junction/resource`*). Direct access to the resource by using the host name of the junctioned server does not require URL filtering.

Virtual host junctions preserve the content of response pages in the same form as the original content on the junctioned web servers. Clients can use the unmodified absolute and server relative URL links on these response pages to successfully locate the resources.

Configuration for virtual host junctions requires that the external DNS maps all virtual host names to the IP address (or addresses) of the WebSEAL server. When the user makes a request to the host name of the junctioned server, the request is routed to WebSEAL.

The HTTP/1.1 specification requires that requests contain an HTTP **Host** header. WebSEAL uses the value of the **Host** header, rather than the URL of the request, to select the appropriate virtual host junction for dispatching the request.

WebSEAL chooses the appropriate virtual host junction as follows:

- If the **Host** header is in the request and its value matches the host name of a configured virtual host junction, WebSEAL uses the virtual host junction.

  You must include the port number in the **Host** header when the virtual host uses a non-standard port for the protocol. The standard port for TCP is `80` and the standard port for SSL is `443`.

  The **Host** header that WebSEAL sends to the virtual host junction contains the value that is specified by **-v** when the virtual host junction was created.
- In all other cases, WebSEAL uses a standard junction that is based on the URL of the request. For example, WebSEAL uses a standard junction in the following situations:

  - If the value of the **Host** header does not match any virtual host junctions (**-v** *vhost_name*[:*port*]).
  - If there is no **Host** header, such as in an HTTP/1.0 request.

By default, virtual host junctions take precedence over standard junctions. You can use the **match-vhj-first** configuration entry in the **[junction]** stanza to reverse this behavior. If this configuration entry is no, WebSEAL searches for a standard junction that matches the request. If no match is found, WebSEAL checks the **Host** header to determine whether a virtual host junction can handle the request.

If you set **match-vhj-first** to no in an environment that maintains sessions, you must set the **shared-domain-cookie** configuration entry to yes. By default, WebSEAL maintains a separate session cache for each virtual host junction and a separate session cache for all standard junctions. For session affinity, use the **shared-domain-cookie** parameter so that WebSEAL can use the same session, regardless of which junction services the request. You can also use the **session-cookie-domain** stanza to specify the domains that share session cookies. For more information about these configuration entries, see the Web Reverse Proxy Stanza Reference topics..

Unlike standard junctions, WebSEAL does not define virtual host junctions as a mount point in the document space. WebSEAL accesses virtual host resources by virtual host junction designations, which are always at the root of the document space of WebSEAL. These designations are called virtual host labels.

The junctioned server with the virtual host name in the HTTP **Host** header returns its own responses, which can contain server-relative or absolute URLs. WebSEAL returns these responses, unfiltered, directly to the client. Absolute URLs in the responses from junctioned servers that reference the server itself must use the virtual host name and not the server IP address. Clients can use the unmodified absolute and server relative URL links on these response pages to successfully locate the resources.

Other features of virtual host junctions:

- Support for both HTTP and HTTPS protocols. To support both protocols between the client and WebSEAL, you must use two virtual host junctions. Use a separate virtual host junction for each protocol.

  A virtual host junction responds only to a single protocol (port). The junctioned server must support both HTTP and HTTPS so that you can create the two junctions with unique SSL and TCP ports on that server.
- WebSEAL can also provide multiple local virtual host junctions of its own to serve protected local content.
- Virtual host junctions share many configuration options with standard WebSEAL junctions. Virtual host junctions do not support several standard junction options. There are also several new options specific to virtual host junctions.
- Each WebSEAL instance can support multiple interfaces, ports, or both. With multiple interfaces and ports, you can configure an SSL certificate for each HTTPS interface and port on which listening occurs.

  If virtual host names resolve to different interfaces or ports that WebSEAL is listening on, then WebSEAL can present different certificates to the connecting clients.

Stanzas and stanza entries ignored by virtual host junctions

Virtual host junctions do not parse or filter URLs in HTML response pages from servers with junctions.

WebSEAL configuration file stanzas and stanza entries are ignored by virtual host junctions:

- **`[server]`, `preserve-base-href`**
- **`[server]`, `process-root-requests`**
- **`[process-root-filter]`**
- **`[junction]`, `jmt-map`**
- **`[filter-url]`**
- **`[filter-events]`**
- **`[filter-schemes]`**
- **`[filter-content-types]`**
- **`[script-filtering]`**
- **`[preserve-cookie-names]`**
- **`[junction]`, `allow-backend-domain-cookies`**

Virtual hosts represented in the object space

Standard WebSEAL junctions are represented in the protected object space as a mount point within the document space of WebSEAL.

Junctions are represented in the following format:

```
/WebSEAL/instance-name/junction-name/path
```

Virtual host junctions are represented in the protected object space by virtual host junction designations, which are always at the root of WebSEAL's document space. These designations are called virtual host labels. For example:

```
/WebSEAL/instance-name/@vhost-label/path
```

The following example shows the representation of the resource, `readme.html`, under the directory, `pubs`, on a virtual host junction with the label **support.ibm.com-http** . The full path in the protected object space appears as follows:

```
/WebSEAL/instance-name/@support.ibm.com-http/pubs/readme.html
```

Virtual host junctions that are created with the **-g** options also appear in the protected object space so management ACLs can be placed on them. However, the directories and resources that are protected by those junctions are not displayed. Those directories and resources are only visible under the primary junction (the **-g** option forces WebSEAL to recognize only a single object space). For example:

```
/WebSEAL/instance-name/@support.ibm.com-http/pubs/readme.html
/WebSEAL/instance-name/@support.ibm.com-https
```

Configuration of a virtual host junction

Depending on the requirements of your environment, you can configure a remote or a local type virtual host junction.

Creation of a remote type virtual host junction

You must understand how to use the available commands so that you can create a remote type virtual host junction. Some of the options include the type of junction and the host name of the target server.

You can use the **server task...virtualhost** commands of the **pdadmin** utility to configure virtual host junctions.

The following example (entered as one line) specifies the syntax for the **pdadmin server task virtualhost create** command (entered as one line):

```
pdadmin> server task instance_name-webseald-host_name virtualhost create
options vhost-label
```

The following table described the common and required **virtualhost create** options:

Table 61. Remote type virtual host junction options

Option	Description
–t *type*	Type of junction. One of: **tcp**, **ssl**, **tcpproxy**, **sslproxy**. Required for all virtual host junctions.
–h *host-name*	The DNS host name or IP address of the target back-end server. The same host name can be used for a TCP junction and an SSL junction. The port of each virtual host differentiates one from the other so that they are each considered unique. Required by **tcp**, **ssl**, **tcpproxy**, and **sslproxy** type junctions.
–v *vhost name*[:*port*]	WebSEAL selects a virtual host junction. The junction processes a request if the request's HTTP **Host** header matches the virtual host name and port number that is specified by the **-v** option. The **-v** option is also used to specify the value of the **Host** header of the request sent to the back-end server. The port number is required if the virtual host uses a non-standard port for the protocol. Standard port for TCP is 80; standard port for SSL is 443. If **-v** is not specified for the following type of junctions, then the junction is selected from the information that is contained in the **-h** hostand **-p** port option or their default value: • **tcp** • **ssl** • **tcpproxy** • **sslproxy**

Table 61. Remote type virtual host junction options (continued)

Option	Description
–g *vhost-label*	If both HTTP and HTTPS protocols need to be supported between the client and WebSEAL, then two junctions to the same virtual host (**-h**) are required. One junction for each protocol (**-t**). By default, each junction recognizes its own unique protected object space even though the junctions point to a single object space. The **-g** option causes a second junction to share the protected object space as the initial junction. You can use a single object space reference to maintain a single access control list (ACL) on each protected object. An initial virtual host junction cannot be deleted if a second virtual host junction exists that used **-g** against the first. An error message is returned at such an attempt. This option is appropriate for junction pairs only (two junctions with complementary protocols). The option does not support the association of more than two junctions. Optional.

Virtual host label:

The virtual host label (vhost-label) is a name for the virtual host junction.

- By default, virtual host junctions are mounted at the root of the WebSEAL object space.
- You can refer to a junction in the **pdadmin** utility with this label.
- The virtual host junction label must be unique within each instance of WebSEAL.
- Because the label is used to represent virtual host junctions in the protected object space, the label name must not contain the forward slash character (/).

Example TCP and SSL virtual host junctions:

See “Scenario 1: Remote virtual host junctions” on page 488.

See “Scenario 2: Virtual host junctions with interfaces” on page 492.

References:

See “Using pdadmin server task to create virtual host junctions” on page 500 for a summary of the **virtualhost** junction commands.

See the Web command reference topics in the IBM Knowledge Center for complete syntax information for the **pdadmin** utility.

Creation of a local type virtual host junction

You must understand how to use the available commands so that you can create a local type virtual host junction. Some of the options include the type of junction and the local directory to junction.

A local virtual host junction (**-t localtcp** and **-t localssl**) is a mount point for specific content that is located locally on the WebSEAL server. Like the content from junctioned remote servers, local junction content is incorporated into WebSEAL's unified protected object space view.

The following options are appropriate for local virtual host junctions:

Table 62. Local type virtual host junction options

Option	Description
–t *type*	Type of junction (**localtcp** or **localssl**). Required.
–g *vhost-label*	The **-g** option causes a second junction to share the protected object space as the initial junction. See "Creation of a remote type virtual host junction" on page 485.
–v *vhost name*[:*port*]	WebSEAL selects a virtual host junction. The junction processes a request if the request's HTTP **Host** header matches the virtual host name and port number that is specified by the **-v** option. The **-v** option is also used to specify the value of the **Host** header of the request sent to the back-end server. The port number is required if the virtual host uses a non-standard port for the protocol. Standard port for TCP is 80; standard port for SSL is 443. The **-v** option is required for **localtcp** and **localssl** type junctions. See "Creation of a remote type virtual host junction" on page 485.
–z *replica-set-name*	Optional. Specifies the replica set that sessions on the virtual host junction are managed under. The command can group or separate login sessions among multiple virtual hosts. If **-z** is not used to specify the replica set for the virtual host junction, the virtual host junction is automatically assigned to a replica set. The replica set matches its virtual host name. For example, if the virtual host name is **vhostA.example.com**, the replica set is named **vhostA.example.com**. The replica set used for the virtual host junction must be present in the **`[replica-sets]`** stanza of the WebSEAL configuration file. See "Advanced configuration for the distributed session cache" on page 338.
–f	Force the replacement of an existing junction. See "Forcing a new junction" on page 401.
–l *percent-value*	Defines the soft limit for consumption of worker threads. See "Per-junction allocation of worker threads for junctions" on page 44.
–L *percent-value*	Defines the hard limit for consumption of worker threads. See "Per-junction allocation of worker threads for junctions" on page 44.

Example local virtual host junction:

The first command (entered as one line) creates a local virtual junction that responds to HTTP requests with **Host** header value of **p.s.com**.

```
pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
-t localtcp -v p.s.com vhost-local-ps-http
```

The second command (entered as one line) creates a local virtual junction that responds to HTTPS requests with **Host** header value of **p.s.com:444**. The **-g** option pairs this SSL junction with the first TCP junction so that they share protected object space.

```
pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
-t localssl -v p.s.com:444 -g vhost-local-ps-http vhost-local-ps-https
```

Scenario 1: Remote virtual host junctions

Understand the required configuration steps to set up junction support for two remote virtual hosts that are on a single server is implemented.

The following scenario sets up junction support for two remote virtual hosts that are on a single back-end server. Refer to the accompanying diagram as you proceed through the steps.

Required architecture:

- By default, the WebSEAL configuration file is set to support all IP addresses:

  ```
  [server]
  network-interface = 0.0.0.0
  ```

  For this virtual host scenario, WebSEAL (**webseal.ibm.com**) is configured to use a specific network address:

  ```
  [server]
  network-interface = 9.0.0.3
  ```

- WebSEAL servers that are protecting two virtual hosts on one back-end junctioned server:
  - Virtual host **a.b.com** (on server **cruz1.ibm.com**)
  - Virtual host **x.y.com** (on server **cruz1.ibm.com**)
- Direct access to the protected junctioned server (**cruz1.ibm.com**) is prevented by appropriate firewall protection. The user is not aware of this blocked access. The external DNS used by the browser to look up the virtual host names are configured to point to WebSEAL at IP address 9.0.0.3.

External DNS	
a.b.com	9.0.0.3
x.y.com	9.0.0.3

- Virtual host **a.b.com** accepts HTTP access only.
- Virtual host **x.y.com** accepts secure HTTPS access.

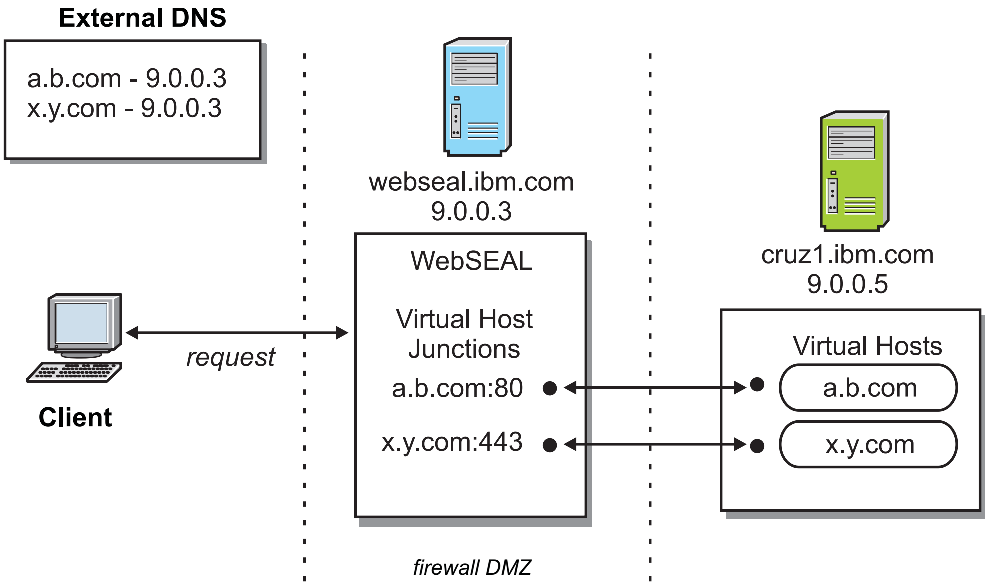


Figure 39. Virtual host junction scenario 1

Procedure:

1. The following **pdadmin** command (entered as one line) creates a virtual host junction named (labeled) **vhost-ab-http** that responds to the **Host: a.b.com** header in TCP (HTTP) requests to WebSEAL:

   ```
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t tcp -h cruz1.ibm.com -v a.b.com vhost-ab-http
   ```

2. The following command (entered as one line) creates a virtual host junction named (labeled) **vhost-xy-https** that responds to the **Host: x.y.com** header in SSL (HTTPS) requests to WebSEAL:

   ```
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t ssl -h cruz1.ibm.com -v x.y.com vhost-xy-https
   ```

3. The client user clicks the following (example) link on an HTML page:

   ```
   http://a.b.com/doc/readme.txt
   ```

   The (example) request for this resource appears as follows:

   ```
   GET /doc/readme.txt HTTP/1.1
   Host: a.b.com
   User-Agent: Mozilla 4.0 (X; I; Linux-2.0.35i586)
   Accept: image/gif, image/jpeg, */*
   ```

   DNS determines that communication to the requested server (**a.b.com**) is routed to the WebSEAL host (9.0.0.3).

   WebSEAL detects the **Host** header and routes the request across the junction for virtual host **a.b.com**, on the back-end server **cruz1.ibm.com**.

Definition of interfaces for virtual host junctions

You can configure WebSEAL to listen on multiple interfaces. You must understand the multiple interface capability because that feature is important in certificate support (SSL) for multiple virtual host junctions.

A digital certificate contains the name of the host that is being accessed. Therefore, it is necessary to have a unique certificate exchange for each virtual host that is configured for SSL. Browsers produce a warning message when there is a name mismatch between certificate and host.

Default interface specification

As an administrator, you must understand the network interface for your environment so that you can configure the settings correctly.

A network interface is defined as the combined set of values for a specific group of settings.

Included settings:
- HTTP or HTTPS port
- IP address
- Worker threads
- Certificate handling

The single default interface for a WebSEAL instance is defined by the following stanza entries in the WebSEAL configuration file:

```
[server]
http
http-port
https
https-port
worker-threads
network-interface

[ssl]
webseal-cert-keyfile-label

[certificate]
accept-client-certs
```

Defining extra interfaces

As an administrator, you can define extra interfaces so that you can configure a set of values for a specific group of settings.

About this task

To configure extra interfaces, define each custom-named interface within the **[interfaces]** stanza of the WebSEAL configuration file.

Each interface definition includes a list of properties. Most properties imitate equivalent stanza entry names that are found in the WebSEAL configuration file and that are part of the default interface specification (see “Default interface specification”).

A custom interface specification uses the following format:

```
[interfaces]
interface-name = property=value[;property=value[;...]]
```

The following table lists the available properties and values that are used to configure a custom interface:

Table 63. Valid properties and values for extra interface definitions

Property	Values	Description
http-port	• port number • `disabled` (default)	Port number to listen for HTTP requests on the specified **network-interface**. The value can also be set to `disabled`. One of either **http-port** or **https-port** must be specified when you define an interface.
https-port	• port number • `disabled` (default)	Port number to listen for HTTPS requests on the specified **network-interface**. The value can also be set to `disabled`. One of either **http-port** or **https-port** must be specified you define an interface.
worker-threads	• count • `default` (default)	Number of worker threads that are used to process requests received only on this interface. The `default` value can be used to specify use of the worker thread pool that belongs to the default interface (see "Default interface specification" on page 490).
network-interface	• IP address • `0.0.0.0` (default)	IP address to listen for requests on the specified **http-port** or **https-port**. Both IPv4 and IPv6 formats are supported.
certificate-label	• key-file-label	Label name of a certificate in the `pdsrv.kdb` key database file. Only valid when **https-port** is specified. The server-side certificate WebSEAL uses to authenticate to the client.
accept-client-certs	• `never` (default) • `required` • `optional` • `prompt_as_needed`	Specifies how WebSEAL is to handle client-side certificates. Only valid when **https-port** is specified. See "Client-side certificate authentication" on page 128.

Syntax rules for property values:

- A value that contains a semicolon (;), double quotation mark ("), or backslash (\) must be preceded by a backslash (\).
- Double quotation marks (") must be used to specify values that contain leading or trailing spaces.
- If a semicolon (;) appears inside a double-quoted value, it does not require a preceding backslash.

Example

```
[interfaces]
support = network-interface=9.0.0.8;https-port=444;certificate-label=WS6;
worker-threads=16
```

This example (entered as one line) creates an interface that is named "support" with the following properties:

- Allows WebSEAL to listen for requests at IP address 9.0.0.8, on HTTPS port 444.
- The HTTP port defaults to "disabled".
- WebSEAL authenticates to SSL clients with a server-side certificate named "WS6" stored in the WebSEAL key database file.
- The interface uses its own pool of 16 worker threads to service requests.
- The interface defaults to never requiring (prompting for) client-side certificates during authentication.

Note: When you configure WebSEAL with multiple DNS aliases that are assigned to its interfaces, each alias and interface must be assigned to a virtual host junction. WebSEAL does not support multiple DNS aliases that are assigned to its interface definitions, which are not assigned to virtual host junctions.

Scenario 2: Virtual host junctions with interfaces

Learn how to set up virtual host junctions on separate WebSEAL interfaces and two junctions that are configured to share a common protected object space.

The following scenario sets up:

- Virtual host junctions are created on separate WebSEAL interfaces.
- Two junctions are configured to share a common protected object space.

Refer to the accompanying diagram as you proceed through the steps.

Required architecture:

- WebSEAL protects three virtual hosts over the following protocols:
  - **a.b.com** (on host **cruz1.ibm.com**) over HTTP and HTTPS
  - **w.x.com** (on host **cruz2.ibm.com**) over HTTP
  - **y.z.com** (on host **cruz2.ibm.com**) over HTTPS
- Direct access to the protected junctioned servers (**cruz1.ibm.com** > **cruz2.ibm.com**) is prevented by appropriate firewall protection. The user is not aware of this blocked access. The external DNS entries that are used by the browser to look up the virtual host names are configured to point to WebSEAL at IP address 9.0.0.3 or 9.0.0.4.
- The virtual hosts are configured in the external DNS to point to the WebSEAL server:

External DNS	
a.b.com	9.0.0.3
x.y.com	9.0.0.3
y.z.com	9.0.0.4

- The WebSEAL server is known to browsers by the following host names:
  - **webseal.ibm.com** (WebSEAL's true host name)

  - **a.b.com**
  - **w.x.com**
  - **y.z.com**
- WebSEAL is configured for two interfaces (to allow serving unique server-side certificates over HTTPS for **a.b.com** > **y.z.com**):
  - 9.0.0.3
  - 9.0.0.4

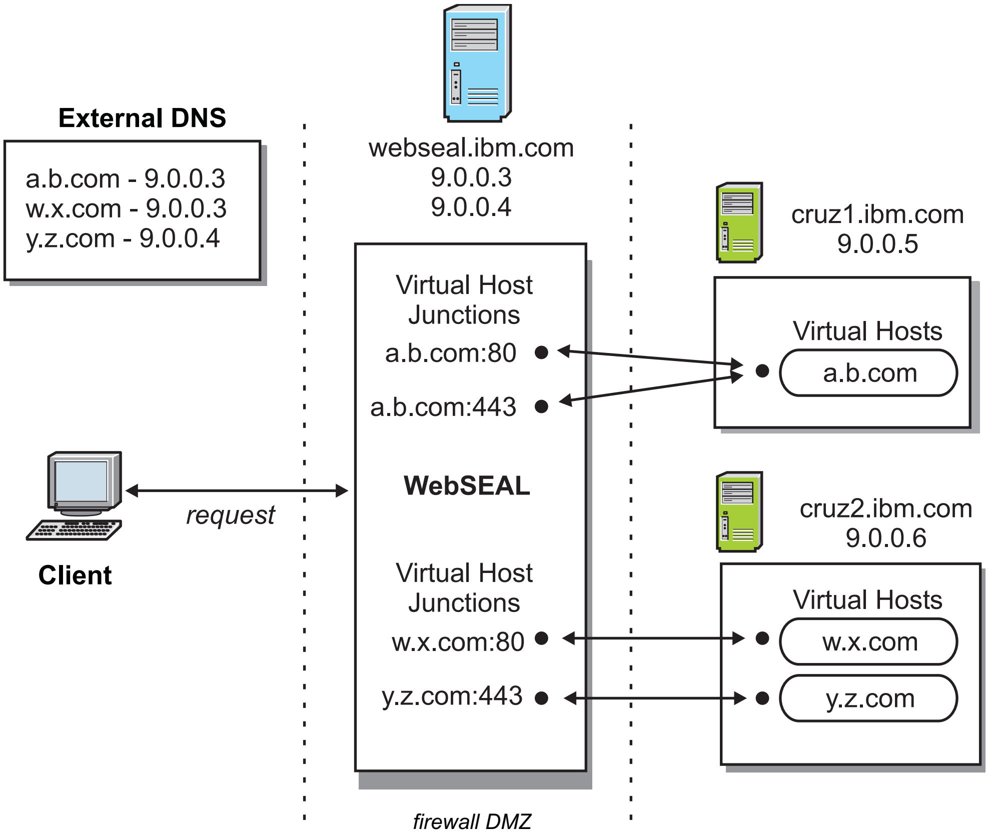


Figure 40. Virtual host junction scenario 2

Procedure - general setup:

1. Install and configure a default WebSEAL with the first of the two required interfaces (to support SSL communication with **a.b.com:443**):

   ```
   [server]
   network-interface = 9.0.0.3
   http = yes
   http-port = 80
   https = yes
   https-port = 443
   ```

2. To support SSL communication between browsers and the **a.b.com** virtual host (over port 443), install a server-side certificate (named **ab** in this example) in WebSEAL's `pdsrv.kdb` key file database. This certificate must be generated and signed by a Certificate Authority (CA). WebSEAL presents this certificate, on behalf of the interface, to authenticate to client browsers.

   ```
   [ssl]
   webseal-cert-keyfile-label = ab
   ```

Note: WebSEAL provides an option to configure a separate certificate key database for junction SSL operations. You can use a separate certificate key database rather than sharing the one used for client certificates that are specified in the **[ssl]** stanza. For more information, see "Configuration of the WebSEAL key database file" on page 367 and the description of the **jct-cert-keyfile** option in jct-cert-keyfile.

3. Configure a second interface to support SSL communication with **y.z.com:443**:

   ```
   [interfaces]
   yz-interface = network-interface=9.0.0.4; certificate-label=yz; https-port=443
   ```

4. To support SSL communication between browsers and the **y.z.com** virtual host (over port 443), install a server-side certificate (named **yz** in this example) in WebSEAL's `pdsrv.kdb` key file database. This certificate must be generated and signed by a Certificate Authority (CA). WebSEAL presents this certificate, on behalf of the interface, to authenticate to client browsers.

   Note: WebSEAL provides an option to configure a separate certificate key database for junction SSL operations. You can use a separate certificate key database rather than sharing the one used for client certificates that are specified in the **[ssl]** stanza. For more information, see "Configuration of the WebSEAL key database file" on page 367 and the description of the **jct-cert-keyfile** option in the jct-cert-keyfile.

5. Assign the appropriate name as a value to the **web-host-name** stanza entry in the WebSEAL configuration file. You must assign the appropriate name value to ensure that the primary WebSEAL host name is used when required.

   ```
   [server]
   server-name = webseal.ibm.com-default
   web-host-name = webseal.ibm.com
   ```

Procedure - create virtual host junctions:

1. Create two virtual host junctions (entered as one line) to support HTTP and HTTPS communication to **a.b.com**. Use the **-g** option to allow the two junctions to share the object space:

   ```
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t tcp -h cruz1.ibm.com -v a.b.com vhost-ab-tcp
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t ssl -h cruz1.ibm.com -v a.b.com -g vhost-ab-tcp vhost-ab-ssl
   ```

2. Create a virtual host junction (entered as one line) to support communication with **w.x.com:80**:

   ```
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t tcp -h cruz2.ibm.com -v w.x.com vhost-wx-tcp
   ```

3. Create a virtual host junction (entered as one line) to support communication with **y.z.com:443**:

   ```
   pdadmin> server task default-webseald-webseal.ibm.com virtualhost create
   -t ssl -h cruz2.ibm.com -v y.z.com vhost-yz-ssl
   ```

E-community single signon with virtual hosts

The E-community single signon feature can be used to do single signon between multiple virtual hosts that are on a single WebSEAL instance.

The virtual hosts all share the configuration file in the WebSEAL instance.

To support virtual host junctions, e-community single signon allows:

- A single virtual host to act as the Master authentication server (MAS) on a machine with more than one virtual hosts.

- Specification of per-domain single signon keys for environments with multiple virtual hosts that supporting different domains.

Configuration enhancements to e-community:

- On a machine with multiple virtual hosts, use the **master-authn-server** stanza entry to designate one of the virtual hosts as the MAS.

  ```
  [e-community-sso]
  is-master-authn-server = yes
  master-authn-server = virtual-host
  ```
- The **[e-community-domain-keys]** stanza is only appropriate for use with standard WebSEAL junctions.

  ```
  [e-community-domain-keys]
  ```
- The **[e-community-domains]** stanza lists the domains supported by virtual hosts.

  ```
  [e-community-domains]
  ```
- The **[e-community-domain-keys:*domain*]** stanzas contain the appropriate keys for each domain that is defined in the **[e-community-domains]** stanza.

  ```
  [e-community-domain-keys:domain]
  ```

Example:

- One WebSEAL instance
- Three virtual hosts (four virtual host junctions) forming an e-community:

  ```
  www.ibm.com:80
  www.ibm.com:443
  www.lotus.com:80
  www.tivoli.com:80
  ```

  Note: www.ibm.com:80 and www.ibm.com:443 are a virtual host junction protocol pair. They were created with the **-g** virtual host junction option and therefore share the object space. They server the single virtual host www.ibm.com.
- MAS is the virtual host www.ibm.com
- WebSEAL configuration file:

  ```
  [e-community-sso]
  is-master-authn-server = yes
  master-authn-server = www.ibm.com

  [e-community-domains]
  name = ibm.com
  name = tivoli.com
  name = lotus.com

  [e-community-sso-domain-keys:ibm.com]
  ibm.com = ibm.key
  tivoli.com = ibm-tivoli.key
  lotus.com = ibm-lotus.key

  [e-community-sso-domain-keys:tivoli.com]
  tivoli.com = tivoli.key
  ibm.com = ibm-tivoli.key

  [e-community-sso-domain-keys:lotus.com]
  lotus.com = lotus.key
  ibm.com = ibm-lotus.key
  ```

Cross-domain single sign-on with virtual hosts

Use cross-domain single sign-on feature to do single sign-on between multiple virtual hosts on a single WebSEAL instance.

The virtual hosts all share the configuration file on to the WebSEAL instance. However, cross-domain single sign-on with virtual hosts has certain configuration limitations.

Cross-domain single sign-on configuration does not allow specification of per-domain single sign-on keys for environments with multiple virtual hosts that support different domains. All virtual hosts that are associated with the WebSEAL instance must share the one **[cdsso-peers]** stanza that is used for key configuration. Therefore, the virtual hosts must share a common key that is used by each domain to communicate to another domain.

In the following example, two virtual hosts on a single WebSEAL instance:

- `a.a.com`
- `b.b.com`

Both domains are owned by separate entities and each of these entities has a separate CDSSO arrangement with another WebSEAL server, `c.c.com`. Ideally, `a.a.com` and `b.b.com` would have separate keys to use for single sign-on with `c.c.com`. The ideal configuration would appear as follows:

```
[cdsso-peers]
# used by a.a.com to communicate with c.c.com
c.c.com = c-a.key
# used by b.b.com to communicate with c.c.com
c.c.com = c-b.key
```

This configuration is not possible on a single WebSEAL instance (hosting both `a.a.com` and `b.b.com` virtual hosts). The **[cdsso-peers]** stanza allows only one key to be specified for a target domain.

The only configuration allowed forces both `a.a.com` and `b.b.com` to use the same key. For example:

```
[cdsso-peers]
# used by both a.a.com and b.b.com to communicate with c.c.com
c.c.com = c-ab.key
```

Each owner of the `a.a.com` and `b.b.com` domains must accept the condition that they share the key.

In addition, if the one key is compromised, both `a.a.com` and `b.b.com` are compromised.

Dynamic URLs with virtual host junctions

Understand how virtual host junctions are expressed in the configuration file so that you can identify how they are labeled and where they map to.

In the `dynurl.conf` configuration file, virtual host junctions are expressed as **`/@<vhost-label>`**.

For example, the following `dynurl.conf` configuration file entry specifies that a virtual host junction labeled **test-http** map all URLs under the directory `/mapfrom` to the protected object `/@test-http/maptoin` **`/@test-http/mapto/@test-http/mapfrom/*`**.

Standard WebSEAL junctions and virtual host junctions can co-exist in the `dynurl.conf` configuration file. For example:

```
/mapto/@test-http/mapfrom/*
/@test-http/mapto/mapfrom/*
```

Using domain session cookies for virtual host single sign-on

You can use domain cookies to support single sign-on and the sharing of a single credential across multiple virtual host junctions in the same WebSEAL instance.

About this task

WebSEAL normally uses host cookies for cookie-based session identification. A browser returns a host cookie to the originating host only. Using host cookies in a virtual host environment results in each virtual host that has its own login and credentials.

Use domain cookies if all virtual hosts are on the same WebSEAL instance and contain the same network domain name.

Alternatively, you can configure a distributed session cache environment to support single sign-on across multiple virtual hosts. In a distributed session cache environment, the session can be distributed between different WebSEAL instances.

In an environment without the distributed session cache, you must set another configuration item for WebSEAL. Set another configuration to handle single sign-on across virtual host junctions in the same WebSEAL instance. The **shared-domain-cookie** configuration item in the **[session]** stanza of the WebSEAL configuration file must be set to yes. You do not need to use this configuration item in a distributed session cache environment. In a distributed session cache environment, this item must be set to no or not defined at all.

Both standard WebSEAL junctions and virtual host junctions can support domain cookies:

- The domain that is used by the session cookie for a specific virtual host junction is determined by the closest match to an entry in the **[session-cookie-domains]** stanza.
- The domain that is used by the session cookie for a specific standard WebSEAL junction is determined by the closest match to the value of the **web-host-name** entry in the **[server]** stanza.

If there is no match, then a host type cookie is used.

Other instances of WebSEAL in the same domain also receive the same domain cookies that are configured for a particular WebSEAL instance. You can customize the names of the WebSEAL session cookies for a specific WebSEAL instance. The WebSEAL instance configuration file provides default names for both TCP and SSL cookies:

```
[session]
tcp-session-cookie-name = PD-H-SESSION-ID
ssl-session-cookie-name = PD-S-SESSION-ID
```

See, “Customization of the session cookie name” on page 314.

Procedure

To enable domain cookies, modify the WebSEAL configuration file to specify the names of the appropriate domains where domain type cookies are to be used. For example:

```
[session-cookie-domains]
domain = ibm.com
domain = cruz.tivoli.com
```

Example

Matching example:

```
[session-cookie-domains]
domain = ibm.com
domain = tivoli.com
```

Technical notes for using domain cookies with virtual hosts

When you are using domain cookies with virtual hosts, consider the following points.

- **Security warning!** It is possible for an untrusted host to exist among the collection of hosts for a specific domain.
- To use domain cookies, all virtual hosts must be in the same DNS domain.
- You can do single sign-on across virtual host junctions in the same WebSEAL instance with WebSEAL alone. Alternatively, you can configure a distributed session cache environment to do single sign-on across virtual host junctions, which can be distributed across WebSEAL instances.
- If you are using the distributed session cache to achieve single sign-on across virtual host junctions, you must not enable the `shared-domain-cookie` configuration item. Enable the configuration in the WebSEAL `[session]` stanza.
- See, "Single signon in a replica set" on page 352.

Scenario 3: Advanced virtual host configuration

Learn how to set up forms authentication, single signon using e-community, and authenticated access to specific resources.

The following scenario is based on scenario 2. The scenario additionally sets up:

- Forms authentication.
- Single signon using e-community.
- Authenticated access to specific resources.

Required architecture:

- Single signon is required between all hosts and protocols. Each virtual host has its own session credential. Standard e-community SSO logout restrictions apply.
- Forms login is required
- Authenticated-only access is required for the following resources:
  - `http(s)://a.b.com/sales/`
  - `https://w.x.com/doc/`
  - `https://y.z.com/code/`

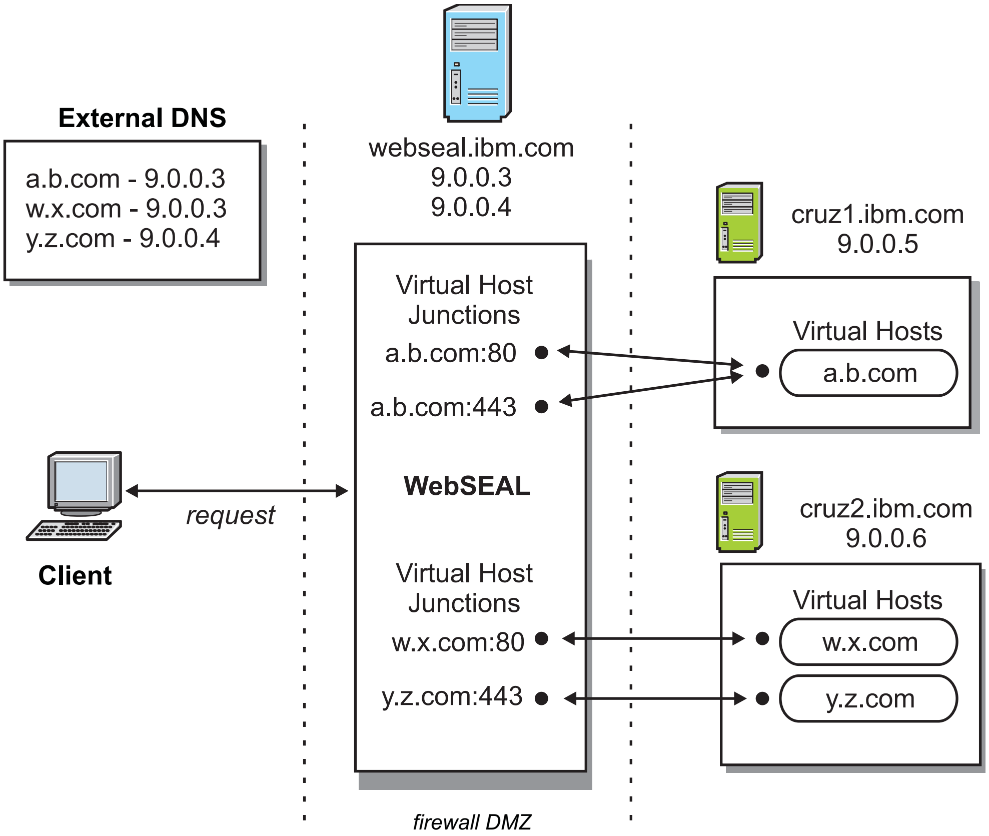


Figure 41. Virtual host junction scenario 3

Procedure - general setup:

1. Enable forms authentication for both HTTP and HTTPS:

```
[ba]
ba-auth = none

[forms]
forms-auth = both
```

2. Configure e-community single signon:

```
[e-community-sso]
e-community-sso-auth = both
e-community-name = ecomm
is-master-authn-server = yes
master-authn-server = a.b.com

[e-community-domain-keys]

[e-community-domains]
name = b.com
name = x.com
name = z.com
name = ibm.com

[e-community-domain-keys:b.com]
b.com = bb.key
x.com = bx.key
z.com = bz.key

[e-community-domain-keys:x.com]
x.com = xx.key
b.com = bx.key
```

```
[e-community-domain-keys:z.com]
z.com = zz.key
b.com = bz.key
```

3. Use the LMI to generate the following keys:
   - `bb.key`
   - `bx.key`
   - `bz.key`
   - `xx.key`
   - `zz.key`
4. Restart WebSEAL and login as **sec_master** to the **pdadmin** command.

Procedure - control access to /restricted directories:

1. Create an **open** (unrestricted) ACL for general unauthenticated access:
   ```
   pdadmin> sec_master> acl create open
   pdadmin> sec_master> acl modify open set user sec_master TcmdbsvaBRlrx
   pdadmin> sec_master> acl modify open set any-other Trx
   pdadmin> sec_master> acl modify open set unauthenticated Trx
   pdadmin> sec_master> acl modify open set group iv-admin TcmdbsvaBRrxl
   pdadmin> sec_master> acl modify open set group webseal-servers Tgmdbsrxl
   ```
2. Create a **restricted** ACL for access that requires authentication:
   ```
   pdadmin> sec_master> acl create restricted
   pdadmin> sec_master> acl modify restricted set group iv-admin TcmdbsvaBRrxl
   pdadmin> sec_master> acl modify restricted set group webseal-servers Tgmdbsrxl
   pdadmin> sec_master> acl modify restricted set user sec_master TcmdbsvaBRlrx
   pdadmin> sec_master> acl modify restricted set any-other Trx
   pdadmin> sec_master> acl modify restricted set unauthenticated T
   ```
3. Attach the **open** ACL to the default WebSEAL instance:
   ```
   pdadmin> sec_master> acl attach /WebSEAL/webseal.ibm.com-default open
   ```
4. Attach the **restricted** ACL to the `/sales` directory on **a.b.com** (entered as one line):
   ```
   pdadmin sec_master> acl attach
   /WebSEAL/webseal.ibm.com-default/@vhost-ab-tcp/sales restricted
   ```
5. Attach the **restricted** ACL to the `/doc` directory on **w.x.com** (entered as one line):
   ```
   pdadmin sec_master> acl attach
   WebSEAL/webseal.ibm.com-default/@vhost-wx-tcp/doc restricted
   ```
6. Attach the **restricted** ACL to the `/code` directory on **y.z.com** (entered as one line):
   ```
   pdadmin sec_master> acl attach
   /WebSEAL/webseal.ibm.com-default/@vhost-yz-ssl/code restricted
   ```

SSL session IDs not usable by virtual hosts

SSL session IDs are not usable for maintaining login sessions between a browser and different virtual hosts that are on the same WebSEAL instance.

Using pdadmin server task to create virtual host junctions

Learn how to use **pdadmin** commands so that you can create virtual host junctions.

Before you begin

You must log in to a secure domain as a user with administration authorization, such as **sec_master** before you use **pdadmin**.

For example:

```
pdadmin> login
Enter User ID: sec_master
Enter Password:
pdadmin>
```

About this task

The **pdadmin** utility provides an interactive command-line prompt where you can do WebSEAL virtual host junction tasks. Use the **pdadmin server task virtualhost** command to create WebSEAL virtual host junctions. See the Web command reference topics in the IBM Knowledge Center for complete syntax information for the **pdadmin** utility. Options for standard WebSEAL junctions are described in "Command option summary: standard junctions" on page 469.

Procedure

To create virtual host junctions, you use the **pdadmin server task virtualhost create** command (entered as one line):

```
pdadmin> server task instance_name-webseald-host_name virtualhost create
options vhost-label
```

For example, if the configured name of a single WebSEAL instance is **web1** installed on a host that is named **www.pubs.com**, the complete server name is:

```
web1-webseald-www.pubs.com
```

Use the **pdadmin server list** command to display the correct format of the complete server name:

```
pdadmin> server list
web1-webseald-www.pubs.com
```

For more information, see the **pdadmin server task virtualhost create** in Chapter 13, "Command reference," on page 627 or "server task virtualhost create" on page 665.

Server task commands for virtual host junctions

Learn about the available virtual junction commands so that you do actions such as add, create, and delete virtual host junctions.

The following virtual host junction commands are available with **pdadmin server task virtualhost**:

Command	Description
virtualhost add	Add more servers to an existing virtual host junction. **Syntax:** `virtualhost add` *`options vhost-label`* See "Addition of a server to a virtual host junction" on page 510.
virtualhost create	Create a new virtual host junction for an initial server. **Syntax:** `virtualhost create` *`options vhost-label`* See "Creation of a virtual host junction" on page 503.
virtualhost delete	Remove the specified virtual host junction. A virtual host junction cannot be deleted if a second virtual host junction refers to it through the **-g** option. An error message is returned at such an attempt. **Syntax:** `virtualhost delete` *`vhost-label`*
virtualhost list	List all configured virtual host junctions by label name. **Syntax:** `virtualhost list`
virtualhost offline	Places the server that is at this virtual host junction in an offline operational state. No additional requests are sent to the specified server. If a server is not specified, then all servers that are at this virtual host junction are placed in an offline operational state. **Syntax:** `virtualhost offline [-i` *`server_uuid`*`]` *`vhost_label`*
virtualhost online	Places the server that is at this virtual host junction in an online operational state. The server now resumes normal operation. If a server is not specified, then all servers that are at this virtual host junction are placed in an online operational state. **Syntax:** `virtualhost online [-i` *`server_uuid`*`]` *`vhost_label`*
virtualhost remove	Remove the specified server from a virtual host junction. **Syntax:** `virtualhost remove –i` *`server-id vhost-label`* Use the **virtualhost show** command to determine the ID of a particular server.

Command	Description
virtualhost show	Display the details of a virtual host junction with the specified label. **Syntax:** `virtualhost show vhost-label`
virtualhost throttle	Places the server that is at this virtual host junction in a throttled operational state. Only requests from users with a session with WebSEAL before the invocation of this command continues to have their requests processed by the specified server. If a server is not specified, then all servers at this virtual host junction are placed in a throttled operational state. **Syntax:** `virtualhost throttle [-i server_uuid] vhost_label`

Creation of a virtual host junction

The **virtualhost create** command creates a new virtual host junction.

Operation: Creates a new virtual host junction.

Syntax:

```
virtualhost create -t type -h host-name options vhost-label
```

-t *type*
Type of virtual host junction. Specify **tcp**, **ssl**, **tcpproxy**, **sslproxy**, **localtcp**, or **localssl**.

Default port for **–t tcp** is **80**. Default port for **–t ss**l is **443**.

This parameter is required.

-h *host-name*
The DNS host name or IP address of the target back-end server. This parameter is required for **tcp**, **ssl**, **tcpproxy**, **sslproxy** type junctions.

options
See Table 64 on page 504.

vhost-label
The name for the virtual host junction. This junction label is used to indicate the junction in the display of the protected object space. You can refer to a junction in the **pdadmin** utility by using this label.

This parameter is required.

See “Creation of a remote type virtual host junction” on page 485.

Table 64. Options on the virtualhost create command

Virtual host junction type	**Parameter**	**Description**
Virtual host option	–v *vhost-name*[:*port*]]	WebSEAL selects a virtual host junction to process a request if the request's HTTP **Host** header matches the virtual host name and port number specified by the **-v** option. The **-v** option is also used to specify the value of the **Host** header of the request sent to the back-end server. The port number is required if the virtual host uses a non-standard port for the protocol. Standard port for TCP is 80; standard port for SSL is 443. If **-v** is not specified for **tcp**, **ssl**, **tcpproxy**, and **sslproxy** type junctions, then the junction is selected from the information in the **-h** *host* and **-p** *port* options. The **-v** option is required for **localtcp** and **localssl** type junctions. See "Creation of a remote type virtual host junction" on page 485.
Virtual host option	–g *vhost-label*	The **-g** option causes a second virtual host junction to share a protected object space as the initial virtual host junction. This option is appropriate for junction pairs only (two junctions with complementary protocols). The option does not support the association of more than two junctions. Optional. See "Creation of a remote type virtual host junction" on page 485.
General option for TCP and SSL junction types	–a *address*	Specifies the local IP address that WebSEAL uses when it is communicating with the target back-end server. If this option is not provided, WebSEAL uses the default address as determined by the operating system. If you supply an address for a particular junction, WebSEAL binds to this local address for all communication with the junctioned server.
General option for TCP and SSL junction types	-E description	A description for the junction.
General option for TCP and SSL junction types	–f	Force the replacement (overwrite) of an existing virtual host junction. See "Forcing a new junction" on page 401.
General option for TCP and SSL junction types	–i	WebSEAL server treats URLs as case insensitive. See "Support for URLs as not case-sensitive" on page 413.

Table 64. Options on the virtualhost create command (continued)

Virtual host junction type	**Parameter**	**Description**
General option for TCP and SSL junction types	–p *port*	TCP port of the back-end third-party server. Default is **80** for TCP junctions. Use **443** for SSL junctions. See "Creating TCP type standard junctions" on page 378 and "Creating SSL type standard junctions" on page 378.
General option for TCP and SSL junction types	–q *path*	Provides WebSEAL with the correct name of the **query_contents** program file and where to find the file. By default, the Windows file is called `query_contents.exe` and the UNIX file is called `query_contents.sh`. By default, WebSEAL looks for the file in the `cgi_bin` directory of the back-end web server. Required for back-end Windows and UNIX web servers. See "Installing and configuring query_contents on Windows-based Web servers" on page 391.
General option for TCP and SSL junction types	–T *resource/ resource-group*	Name of GSO resource or resource group. Required for and used only with **–b gso** option. See "Configuring a GSO-enabled WebSEAL junction" on page 527.
General option for TCP and SSL junction types	–w	Windows 32-bit (Win32) file system support. See "Junctions to Windows file systems" on page 414.
Stateful junctions See "Stateful junctions" on page 397.	–s	Specifies that the virtual host junction support stateful applications. By default, junctions are not stateful.
Stateful junctions See "Stateful junctions" on page 397.	–u *UUID*	Specifies the UUID of a back-end server that is connected to WebSEAL with a stateful virtual host junction (–s).
Mutual authentication over Basic Authentication and SSL certificates See "Mutually authenticated SSL junctions" on page 393.	–B	WebSEAL uses BA header information to authenticate to back-end virtual host. Requires –U, and –W options.

Table 64. Options on the virtualhost create command (continued)

Virtual host junction type	Parameter	Description
Mutual authentication over Basic Authentication and SSL certificates See "Mutually authenticated SSL junctions" on page 393.	–D "*DN*"	Specify Distinguished Name of back-end server certificate. This value, matched with actual certificate DN enhances authentication.
Mutual authentication over Basic Authentication and SSL certificates See "Mutually authenticated SSL junctions" on page 393.	–K "*key-label*"	Key label of WebSEAL's client-side certificate, used to authenticate to back-end virtual host.
Mutual authentication over Basic Authentication and SSL certificates See "Mutually authenticated SSL junctions" on page 393.	–U "*username*"	WebSEAL user name. Use with –B to send BA header information to back-end server.
Mutual authentication over Basic Authentication and SSL certificates See "Mutually authenticated SSL junctions" on page 393.	–W "*password*"	WebSEAL password. Use with –B to send BA header information to back-end server.
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–H *host-name*	The DNS host name or IP address of the proxy server.

Table 64. Options on the virtualhost create command (continued)

Virtual host junction type	Parameter	Description
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–P *port*	The TCP port of the proxy server.
Supply identity information in HTTP headers	–b *BA-value*	Defines how the WebSEAL server passes client identity information in HTTP basic authentication (BA) headers to the back-end virtual host. One of: **filter** (default), **ignore**, **supply**, **gso** See "Client identity in HTTP BA headers" on page 516.
Supply identity information in HTTP headers	–c *header-types*	Insert client identity information specific to Security Access Manager in HTTP headers across the virtual host junction. The *header-types* argument can include any combination of the following Security Access Manager HTTP header types: **iv-user**, **iv-user-l**, **iv-groups**, **iv-creds**, **all**. See "Client identity in HTTP headers (–c)" on page 520.
Supply identity information in HTTP headers	–e *encoding-type*	Specifies the encoding to use when you generate HTTP headers for virtual host junctions. This encoding applies to headers that are generated with both the **–c** junction option and tag-value. The following list shows the possible values for encoding: • utf8_bin • utf8_uri • lcp_bin • lcp_uri See "UTF-8 encoding for HTTP header data" on page 416.
Supply identity information in HTTP headers	–I	NOT VALID. This option is not valid because cookie handling is not required over virtual host junctions.
Supply identity information in HTTP headers	–j	NOT VALID. This option is not valid because the junction cookie solution is not required over virtual host junctions.
Supply identity information in HTTP headers	–J trailer[,onfocus]	NOT VALID. This option is not valid because the junction cookie solution is not required over virtual host junctions.
Supply identity information in HTTP headers	–k	Send session cookie to back-end virtual host. See "Passing of session cookies to junctioned portal servers" on page 412.

Table 64. Options on the virtualhost create command (continued)

Virtual host junction type	Parameter	Description
Supply identity information in HTTP headers	–n	NOT VALID. This option is not valid because the junction cookie solution is not required over virtual host junctions.
Supply identity information in HTTP headers	–r	Insert incoming IP address in HTTP header across the virtual host junction. See “Client IP addresses in HTTP headers (–r)” on page 523.
Junction fairness	–l *percent-value*	Defines the soft limit for consumption of worker threads.
Junction fairness	–L *percent-value*	Defines the hard limit for consumption of worker threads.
WebSphere single signon (LTPA) junctions See “Configuration of an LTPA junction” on page 530.	–A	Enables virtual host junctions to support LTPA cookies (tokens). LTPA version 1 cookies (LtpaToken) and LTPA version 2 cookies (LtpaToken2) are both supported. LTPA version 1 cookies are specified by default. LTPA version 2 cookies must be specified with the additional **-2** option. Also requires –F, and –Z options.
WebSphere single signon (LTPA) junctions See “Configuration of an LTPA junction” on page 530.	–2	Used with the **-A** option, this option specifies that LTPA version 2 cookies (LtpaToken2) are used. The **-A** option without the **-2** option specifies that LTPA version 1 cookies (LtpaToken) are used.
WebSphere single signon (LTPA) junctions See “Configuration of an LTPA junction” on page 530.	–F "*keyfile*"	Name of the key file that is used to encrypt LTPA cookie data. Only valid with –A option.
WebSphere single signon (LTPA) junctions See “Configuration of an LTPA junction” on page 530.	–Z "*keyfile-password*"	Password for the key file that is used to encrypt LTPA cookie data. Only valid with –A option.

Table 64. Options on the virtualhost create command (continued)

Virtual host junction type	**Parameter**	**Description**
Tivoli Federated Identity Manager SSO junctions "Single sign-on with the Security Token Service" on page 513	*-Y*	Enables Tivoli Federated Identity Manager single-signon (SSO) for the junction. **Note:** Before you use this option, you must first configure the WebSEAL configuration files to support Tivoli Federated Identity Manager single-signon over junctions.
WebSEAL-to-WebSEAL SSL junctions See "WebSEAL-to-WebSEAL junctions over SSL" on page 396.	–C	Mutual authentication between a front-end WebSEAL server and a back-end WebSEAL server over SSL. Requires **–t ssl** or **–t sslproxy** type.
Forms single signon See "Forms single sign-on process flow" on page 532.	–S *path*	Name of the forms single signon configuration file.
Transparent path junctions	–x	NOT VALID.
Distributed session cache	–z *replica-set-name*	For distributed session cache environments, this parameter is optional. Specifies the replica set that sessions on the virtual host junction are managed under. It is specified so that you can group or separate log in sessions among multiple virtual hosts. If **-z** is not used to specify the replica set for the virtual host junction, the virtual host junction is automatically assigned to a replica set. The assigned replica set matches its virtual host name. For example, if the virtual host name is **vhostA.example.com**, the replica set is named **vhostA.example.com**. The replica set used for the virtual host junction must be present in the **[replica-sets]** stanza of the WebSEAL configuration file. For environments that do not use the distributed session cache, this option is not applicable. See "Advanced configuration for the distributed session cache" on page 338.

Addition of a server to a virtual host junction

The **virtualhost add** command adds another server to a virtual host junction.

Operation: Adds another server to an existing virtual host junction.

Syntax:

```
virtualhost add -h host-name options vhost-label
```

-h *host-name*
The DNS host name or IP address of the proxy server. This parameter is required. See "Standard WebSEAL junction configuration" on page 377.

options
See Table 65.

vhost-label
The name for the virtual host junction where the additional back-end server is added.

This parameter is required.

Table 65. Options on the virtualhost add command

Virtual host option	Parameter	Description
General option for TCP and SSL junction types	–a *address*	Specifies the local IP address that WebSEAL uses when communicating with the target back-end server. If this option is not provided, WebSEAL uses the default address as determined by the operating system. If you supply an address for a particular junction, WebSEAL binds to this local address for all communication with the junctioned server.
General option for TCP and SSL junction types	–i	WebSEAL server treats URLs as case insensitive. See "Support for URLs as not case-sensitive" on page 413.
General option for TCP and SSL junction types	–p *port*	TCP port of the back-end third-party server. Default is **80** for TCP junctions; **443** for SSL junctions. See "Creating TCP type standard junctions" on page 378 and "Creating SSL type standard junctions" on page 378.
General option for TCP and SSL junction types	–q *url*	Relative path for **query_contents** script. By default, WebSEAL looks for **query_contents** in `/cgi_bin/`. If this directory is different or the **query_contents** file name is different, use this option to indicate to WebSEAL the new URL to the file. Required for back-end Windows servers. See "Installing and configuring query_contents on Windows-based Web servers" on page 391.
General option for TCP and SSL junction types	–w	Windows file system support. See "Junctions to Windows file systems" on page 414.

Table 65. Options on the virtualhost add command (continued)

Virtual host option	Parameter	Description
Stateful junctions See "Stateful junctions" on page 397.	–u *UUID*	Specifies the UUID of a back-end server that is connected to WebSEAL in a stateful virtual host junction (–s).
Mutual authentication over SSL See "Mutually authenticated SSL junctions" on page 393.	–D "*DN*"	Specify Distinguished Name of back-end server certificate. This value, matched with actual certificate DN enhances authentication.
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–H *host name*	DNS host name or IP address of the proxy server.
Proxy junction (requires –t tcpproxy or –t sslproxy) See "TCP and SSL proxy junctions" on page 395.	–P *port*	The TCP port of the proxy server.

Chapter 9. Single Sign-on Solutions

Single sign-on with the Security Token Service

Understand the available configuration options so that you can use the Security Token Service to implement single sign-on.

Examples of the Security Token Service token types that can be used for single sign-on:

- LTPA tokens.
- SAML tokens.

WebSEAL can obtain tokens, which can be used for single sign-on to junctions, from the Security Token Service. The Security Token Service can generate SSO tokens with STS modules, which are available in the Security Token Service. WebSEAL retrieves the Security Token Service SSO tokens by delegating the token request to the module in the following manner:

1. The client authenticates to WebSEAL over HTTPS or HTTP and requests an object on the junctioned server. However, a Security Token Service SSO credential is required before access can be granted to the junctioned server.
2. WebSEAL sends a Simple Object Access Protocol (SOAP) request to the STS module, requesting an SSO token.
3. The Security Token Service generates the token, which is based on the requirements of the STS module.
4. The Security Token Service returns the token to WebSEAL, and then WebSEAL forwards the token to the junction.

A trust chain must be created in the Security Token Service to handle the generation of the security token. The following table highlights the configuration requirements for the trust chain.

Table 66. Configuration requirements for a Security Token Service trust chain

Trust Chain Element	Requirement
Request Type	Issue Oasis URI
Lookup Type	Use Traditional WS-Trust Elements (AppliesTo, Issuer, and TokenType)

Table 66. Configuration requirements for a Security Token Service trust chain (continued)

Trust Chain Element	Requirement
AppliesTo	**Address** Corresponds to the **applies-to** option in the [tfimsso:<*jct id*>]stanza of the WebSEAL configuration file. **Service Name** Corresponds to the **service-name** option in the [tfimsso:<*jct id*>] stanza of the WebSEAL configuration file **Note:** Set fields in this entry to either: • Asterisk (*) to match all service names, or • The second field must be set to the value defined by [tfimsso:<*jct id*>]**service-name** Refer to the Security Token Service documentation for further details on configuring Trust Chains. **Port Type** Not set.
Issuer	**Address** `amwebrte-sts-client` **Service name** Not set. **Port Type** Not set.
TokenType	One of the supported Security Token Service SSO token types
Trust Service Chain Modules	1. `com.tivoli.am.fim.trustserver.sts.modules.STSTokenIVCred: -mode = validate` 2. `com.tivoli.am.fim.trustserver.sts.modules.` *`any_STS_module`*`: -mode = issue`

To create a junction for the Security Token Service single sign-on, use the junction create command (server task create) with option **-Y**. For more information, see "Options" under "server task create" on page 635 or "server task virtualhost create" on page 665.

Note: The WebSEAL configuration file must be configured to support the specific junctions for the Security Token Service single-sign-on before you can use the junction create command with option **-Y**.

Configuration options for using the Security Token Service single sign-on approach is specified in the **[tfimsso:<jct-id>]** stanza. This stanza contains the Security Token Service single sign-on configuration information for a single junction. For standard junctions, the stanza name must be qualified with the name of the junction point, including the leading forward slash; for example: `[tfimsso:/junction_a]`. For virtual host junctions, the stanza name must be qualified with the virtual host label, for example: `[tfimsso:www.ibm.com]`.

The **tfim-cluster-name** option in the **[tfimsso:<jct-id>]** stanza defines the name of the server that is hosting the Security Token Service. Use the corresponding **[tfim-cluster:<cluster>]** stanza to specify options for the cluster.

In the **[tfim-cluster:<cluster>] server** stanza entry, specify the priority level and URL for a web server that acts as a proxy for Security Token Service. The **[tfim-cluster:<cluster>]** stanza can contain multiple**server** entries, which you can use to specify multiple server entries for failover and load balancing purposes. WebSEAL checks the status of the Security Token Service proxy web server every minute after the Security Token Service cluster is configured.

For more information about these configuration options, see the **[tfimsso:<jct-id>]** and **[tfim-cluster:<cluster>]** stanzas in the *IBM Security Web Gateway appliance: Web Reverse Proxy Stanza Reference*.

This method of single sign-on can be implemented only by using a module in the Security Token Service. For more information, refer to the Security Token Service documentation.

GSKit configuration for connections with Security Token Service

There are a number of GSKit attributes that you can use to control how GSKit creates SSL connections.

You can configure WebSEAL to use particular GSKit attributes when it initializes SSL connections.

The **gsk-attr-name** configuration entry in the **[tfim-cluster:<cluster>]** stanza controls the GSKit attributes that WebSEAL uses when initializing a connection with Security Token Service. You can specify this configuration entry multiple times. Include each desired GSKit attribute as a new entry.

```
[tfim-cluster:<cluster>]
gsk-attr-name = {enum | string | number}:id:value
```

Note: Similar configuration entries exist in the **[ssl]** stanza for connections with clients and junctioned web servers.

For further details about these configuration entries, see the Web Reverse Proxy Stanza Reference topics.

Single signon (SSO) concepts

Understand single signon concepts so that you can implement a solution that allows users to access resources with one initial login.

A client that requests for access to a resource can be required to do multiple logins. Multiple logins might be required when a protected resource is on a back-end web application server. One login for the WebSEAL server and one for the back-end server might be required. Each login likely requires different login identities.

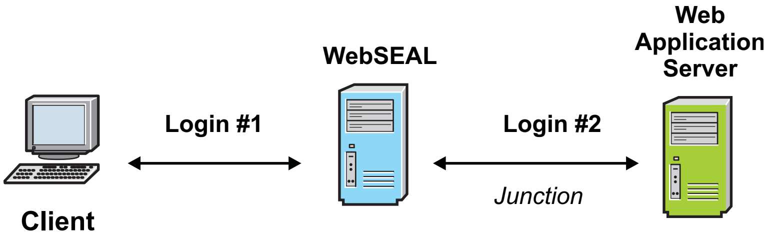


Figure 42. Multiple logins

The problem of administering and maintaining multiple login identities can often be solved with a single signon (SSO) solution. A single signon solution allows the user to access a resource, regardless of the resource's location, with only one initial login. Any further login requirements from back-end servers are handled transparently to the user.

Client identity in HTTP BA headers

You can configure WebSEAL junctions to supply the back-end server with original or modified client identity information. Understand the options available so that you can specify the required information in the HTTP basic authentication headers.

Use the **–b** options to supply specific client identity information in HTTP Basic Authentication (BA) headers.

As the administrator, you must analyze your network architecture and security requirements, and determine answers to the following questions:

1. Is authentication information required by the back-end server?

   (WebSEAL uses the HTTP Basic Authentication header to convey authentication information.)
2. If authentication information is required by the back-end server, where does this information come from?

   (What information does WebSEAL place in the HTTP header?)
3. Does the connection between WebSEAL and the back-end server need to be secure?

   (TCP or SSL junction?)

After the initial authentication between the client and WebSEAL, WebSEAL can build a new Basic Authentication header. The request uses this new header as it continues across the junction to the back-end server. You use the **–b** options to dictate what specific authentication information is supplied in this new header.

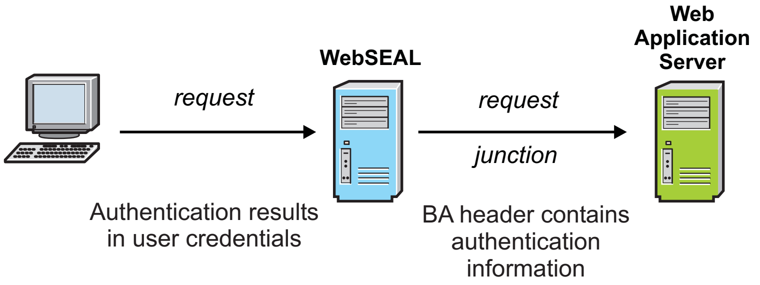

Web Application Server
WebSEAL
request
request
junction
Authentication results in user credentials
BA header contains authentication information

Figure 43. Supplying authentication information to back-end application servers

Client identity and generic password

As an administrator, you must know how client identity and passwords are handled so that you understand how WebSEAL manages authentication in some scenarios.

The **–b supply** option instructs WebSEAL to supply the authenticated Security Access Manager user name (client's original identity) with a static, generic (dummy) password. The original client password is not used in this scenario.

A generic password eliminates password administration and supports the application on a per-user basis. The dummy password is set in the **basicauth-dummy-passwd** stanza entry of the WebSEAL configuration file:

```
[junction]
basicauth-dummy-passwd = password
```

This scenario assumes that the back-end server requires authentication from a Security Access Manager identity. By mapping a client user to a known Security Access Manager user, WebSEAL manages authentication for the back-end server and provides a simple domain-wide single signon solution.

The following conditions exist for this solution:

- WebSEAL is configured to supply the back-end server with the user name contained in the original client request plus a generic dummy password.
- The dummy password is configured in the WebSEAL configuration file.
- The back-end server registry must recognize the Security Access Manager identity that is supplied in the HTTP BA header.
- Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. Therefore, an SSL junction is appropriate.

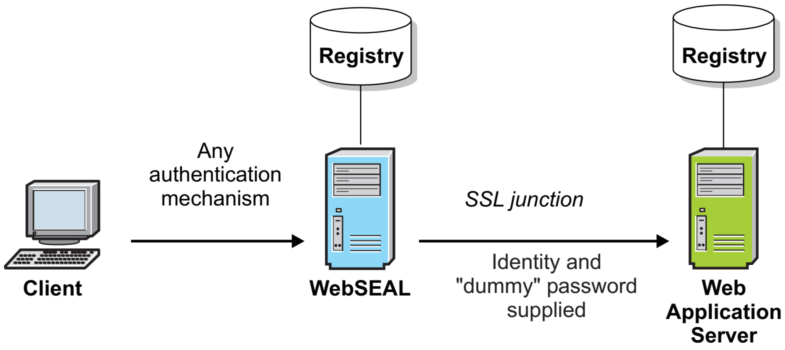

Registry
Registry
Any authentication mechanism
SSL junction
Client
WebSEAL
Identity and "dummy" password supplied
Web Application Server

Figure 44. BA Header contains identity and dummy password

Limitations of the -b supply option

You must understand the limitations of the **-b** supply option so that you can secure servers properly.

The same Security Access Manager dummy password is used for all requests; all users have the same password in the back-end server registry. The common dummy password offers no basis for the application server to prove the legitimacy of the client that is logging in with that user name.

If clients always go through WebSEAL to access the back-end server, this solution does not present any security problems. However, it is important to physically secure the back-end server from other possible means of access.

Because this scenario has no password-level security, the back-end server must implicitly trust WebSEAL to verify the legitimacy of the client.

The back-end server registry must also recognize the Security Access Manager identity in order to accept it.

Forwarding of original client BA header information

Understand how original client basic authentication information is sent to the back-end server without interference and the conditions that are required for this implementation.

The **–b ignore** option instructs WebSEAL to pass the original client basic authentication (BA) header straight to the back-end server without interference. WebSEAL can be configured to authenticate this BA client information. WebSEAL can also be configured to ignore the BA header that is supplied by the client and forward the header without modification to the back-end server.

Note: This implementation is not a true single signon mechanism, but rather a direct login to the third-party server, not apparent to WebSEAL.

The following conditions exist for this solution:

- The back-end server requires client identity information through BA.

  The back-end server sends a Basic Authentication challenge back to the client. The client responds with user name and password information that the WebSEAL server passes through without modification.
- The back-end server maintains its own client-supplied passwords.
- WebSEAL is configured to supply the back-end server with the user name and password that is contained in the original client request.
- Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. An SSL junction is most appropriate.

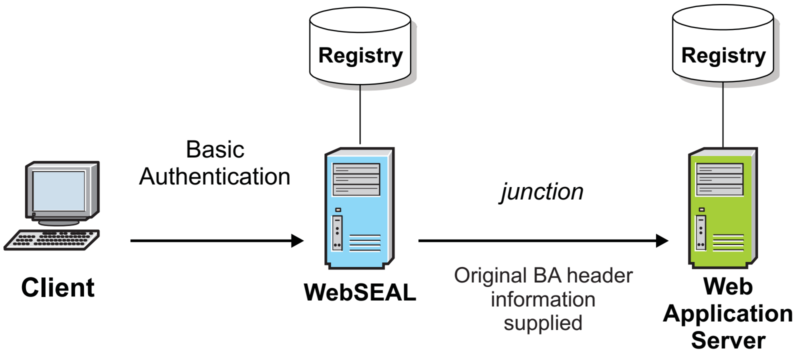


Figure 45. WebSEAL forwards original client identity information

Removal of client BA header information

Understand how basic authentication header information is removed from client requests and the conditions that are required for this implementation.

The **–b filter** option instructs WebSEAL to remove all basic authentication header information from any client requests before requests are forwarded to the back-end server. In this scenario, WebSEAL becomes the single security provider.

The following conditions exist for this solution:

- Basic authentication is configured between the client and WebSEAL.
- The back-end server does not require basic authentication.
- The back-end server can be accessed only through WebSEAL.
- WebSEAL handles authentication on behalf of the back-end server.

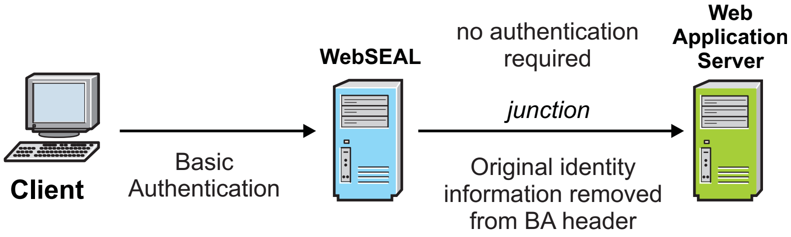

Client
Basic Authentication
WebSEAL
no authentication required
junction
Original identity information removed from BA header
Web Application Server

Figure 46. Removing client BA header information

If you need to supply the back-end server with some client information, you can combine this option with the **–c** option to insert Security Access Manager client identity information into HTTP header fields. See "Client identity in HTTP headers (–c)" on page 520.

User names and passwords from GSO

Understand how authentication information is obtained from a server that handles global signon and the conditions for this implementation.

The **–b gso** option instructs WebSEAL to supply the back-end server with authentication information (user name and password) obtained from a server that handles global signon (GSO).

The following conditions exist for this solution:

- The back-end server applications require different user names and passwords that are not contained in the WebSEAL registry.
- Security is important for both WebSEAL and the back-end server.

Because sensitive authentication information (user name and password) is passed across the junction, the security of the junction is important. An SSL junction is appropriate.

Client identity information across junctions

A junction can be set up to specify client identity information in BA headers. You must know the available options so that you can use the correct combination of options.

The **–b** option allows four possible arguments: filter, supply, ignore, global signon.

The **–b** option has an impact on the junction settings for mutual authentication and you must consider the correct combination of options.

-b supply

- WebSEAL authentication with a BA header is not allowed with this option. This option uses the BA header for the original client user name and a dummy password.
- WebSEAL authentication with a client certificate is allowed with this option.

-b ignore

- WebSEAL authentication with a BA header is not allowed with this option. This option uses the BA header for the original client user name and password.
- WebSEAL authentication with a client certificate is allowed with this option.

-b gso

- WebSEAL authentication with a BA header is not allowed with this option. This option uses the BA header for user name and password information that is supplied by the GSO server.
- WebSEAL authentication with a client certificate is allowed with this option.

-b filter

- Internally, the **–b filter** option is used when WebSEAL authentication is set to use BA header information.

  The WebSEAL BA header is used for all subsequent HTTP transactions. To the back-end server, WebSEAL appears logged on always.
- WebSEAL authentication with a client certificate is allowed with this option.
- If the back-end server requires actual client identity (from the browser), the CGI variables HTTP_IV_USER, HTTP_IV_GROUP, and HTTP_IV_CREDS can be used. For scripts and servlets, use the corresponding Security Access Manager HTTP headers: **iv-user**, **iv-groups**, **iv-creds**.

Client identity in HTTP headers (–c)

Use the **–c** junction option to insert client identity, group membership, and credential information specific to Security Access Manager. You can insert the information into the HTTP headers of requests that are destined for junctioned third-party application servers.

This HTTP header information enables applications on junctioned third-party servers to do user-specific actions (such as single signon) based on the client's Security Access Manager identity.

HTTP header information must be transformed to environment variable format for use by a service on the back-end server. To support CGI programming, header information is transformed into a CGI environment variable format. It is transformed by replacing all dashes (-) with underscores (_) and prepending HTTP to the beginning of the header string. The Security Access Manager HTTP header entries are available to CGI programs as the environment variables **HTTP_IV_USER**, **HTTP_IV_USER_L**, **HTTP_IV_GROUPS**, and **HTTP_IV_CREDS**.

For other application framework products, refer to the appropriate product documentation for instructions on extracting headers from HTTP requests.

HTTP Headers specific to Security Access Manager	CGI Environment Variable Headers	Description
iv-user	**HTTP_IV_USER**	The user name of the client (login ID). Defaults to "Unauthenticated" if client is unauthenticated (unknown).
iv-user-l	**HTTP_IV_USER_L**	The distinguished name (DN) of the client.
iv-groups	**HTTP_IV_GROUPS**	A list of groups to which the client belongs. Consists of comma separated quoted entries.
iv-creds	**HTTP_IV_CREDS**	Encoded opaque data structure that represents an Security Access Manager credential. Supplies credentials to remote servers so mid-tier applications can use the authorization API to call the authorization service. See Authorization C API Developer Reference.

The **–c** option to the junction **create** command (see “Command option summary: standard junctions” on page 469) specifies what Security Access Manager HTTP header data is sent across a junction to the back-end application server:

```
-c header-types
```

The **header-types** arguments:

Argument	Description
iv_user	Provides the user name (short form) to the **iv-user** HTTP header of the request.
iv_user_l	Provides the full DN of the user (long form) to the **iv-user-l** HTTP header of the request.
iv_groups	Provides the user's list of groups to the **iv-groups** HTTP header of the request.

Argument	Description
iv_creds	Provides the user's credential information to the **iv-creds** HTTP header of the request.
all	Provides identity information for **iv-user**, **iv-groups**, and **iv-creds** HTTP headers of the request.

The **–c** option is also supported on virtual host junctions.

Conditions of use -c junctions

Understand how to use the **-c junction** option so that you can use the correct combinations to insert information into HTTP headers of requests.

- Separate multiple arguments to the **-c** option with commas only. Do not enter any spaces.
- The **-c all** option passes only the **iv-user** (**HTTP_IV_USER**), **iv-groups** (**HTTP_IV_GROUPS**), and **iv-creds** (**HTTP_IV_CREDS**) headers across the junction.

  Note: The **-c all** option does not pass **iv-user-l** (**HTTP_IV_USER_L**)) headers.
- You must individually specify all four header options to pass all four header types across the junction:
  ```
  -c iv_user,iv_user_l,iv_groups,iv_creds
  ```
- The **HTTP_IV_USER** header is used for either **iv-user** and **iv-user-l** when only one of them is specified. If both **iv-user** and **iv-user-l** headers are specified, **HTTP_IV_USER_L** is used for **iv-user-l**.
- To ensure security of the **iv-creds** header value, use SSL junctions.
- The content caching mechanism does not cache responses to requests over junctions that are configured with the **–c** and **–C** options.

Note: It is possible to improve the WebSEAL performance under –c junction conditions by applying the cache configuration.

```
-c iv_user,iv_user_l
```

The **iv-user** (**HTTP_IV_USER**) and **iv-user-l** (**HTTP_IV_USER_L**) headers are passed to the back-end application server.

*****

```
-c iv_user_l
```

The **iv-user-l** (**HTTP_IV_USER**) header is passed to the back-end application server.

*****

```
-c all
```

The **iv-user** (**HTTP_IV_USER**), **iv-groups** (**HTTP_IV_GROUPS**), and **iv-creds** (**HTTP_IV_CREDS**) headers are passed to the back-end application server.

*****

```
-c iv_user,iv_user_l,iv_groups,iv_creds
```

The **iv-user** (**HTTP_IV_USER**), **iv-user-l** (**HTTP_IV_USER_L**), **iv-groups** (**HTTP_IV_GROUPS**), and **iv-creds** (**HTTP_IV_CREDS**) headers are passed to the back-end application server.

Client IP addresses in HTTP headers (–r)

The **–r** junction option allows you to insert client IP address information into the HTTP headers of requests destined for junctioned application servers. The HTTP header information enables applications on junctioned third-party servers to perform actions based on this IP address information.

HTTP header information must be transformed by the back-end server to environment variable format for use by a service on the back-end server. Header information is transformed into a CGI environment variable format by replacing all dashes (-) with underscores (_) and prepending "HTTP" to the beginning of the string. The value of the HTTP header becomes the value of the new environment variable.

HTTP Header Field specific to Security Access Manager	CGI Environment Variable Equivalent	Description
iv-remote-address	**HTTP_IV_REMOTE_ADDRESS**	The IP address of the client. This value could also represent the IP address of a proxy server or a network address translator (NAT).
iv-remote-address-ipv6	**HTTP_IV_REMOTE_ADDRESS_IPV6**	The IP address of the client in IPv6 format.

The **–r** option specifies that the IP address of the incoming request be sent to the back-end application server. The option is expressed without any arguments.

The **–r** option is also supported on virtual host junctions.

Limiting the size of WebSEAL-generated HTTP headers

You can limit the size of WebSEAL-generated HTTP headers that are inserted in requests to junctioned back-end servers so that they are not too large.

About this task

The **max-webseal-header-size** stanza entry in the **[junction]** stanza of the WebSEAL configuration file specifies the maximum size, in bytes, of WebSEAL-generated HTTP headers. A value of 0 disables this function:

```
[junction]
max-webseal-header-size = 0
```

Note: The **max-webseal-header-size** entry does not limit the maximum size of HTTP-Tag-Value headers.

This stanza entry can be useful if a back-end application server rejects WebSEAL-generated HTTP headers because they are too large. For example, an **iv-creds** header for a user that belongs in many groups might be too large.

When configured, this stanza entry causes WebSEAL-generated headers that exceed the maximum value to split across multiple headers. The following example output from a CGI application illustrates the effect of split headers:

```
HTTP_IV_CREDS_1=Version=1, BAKs3DCCBnMMADCCBm0wggZpAgIDkDCCAYUwKzA
HTTP_IV_CREDS_2=+0+8eAgI8iAICEdYCAgCkAgFUBAaSVNCJqncMOWNuPXNlY21==
HTTP_IV_CREDS_SEGMENTS=2
```

If you enable this function, you must modify the back-end application to recognize split headers, instead of standard WebSEAL-specific HTTP headers.

Global sign-on overview

Global sign-on (GSO) grants users access to the computing resources that they are authorized to use through a single login. This feature is designed for large enterprises that consist of multiple systems and applications within heterogeneous, distributed computing environments. GSO eliminates the need for users to manage multiple user names and passwords.

The integration is achieved by creating "aware" junctions between WebSEAL and back-end web servers.

The GSO data can be stored in either the Security Access Manager user registry or an external source that WebSEAL communicates with through a RESTful web service. The web service must be accessed through a junction that is local to the WebSEAL instance. This setup allows all of the advanced junctioning capabilities (for example: HA, failover, SSO) to be used on these web service requests.

Note: For security purposes, this junction must be set up to prevent external access. You can achieve such setup by changing the ACL so that no user has read permission on this web service. When WebSEAL sends the GSO RESTful web service request via the junction, it automatically bypasses the policy definition for the junction.

When WebSEAL receives a request for a resource that resides on a junctioned server that is configured to require GSO credential information, it attempts to locate the credential information from the Security Access Manager user registry or the external GSO data source.

GSO embedded storage

GSO data can be stored in the Security Access Manager user registry and provided to WebSEAL through its LDAP service.

GSO resources and GSO resource groups must first be created with the Web Portal Manager or **pdadmin** utility.

The following figure illustrates how the GSO mechanism is used to retrieve user names and passwords for back-end application resources.

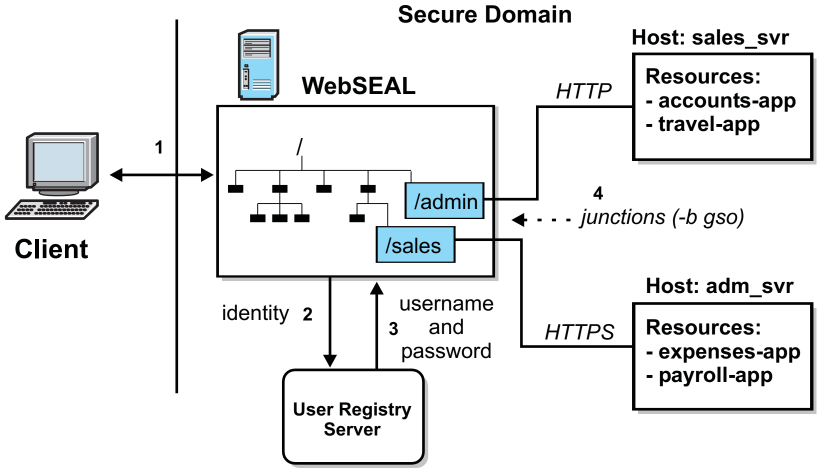


Figure 47. Global sign-on mechanism

1. The client authenticates to WebSEAL with a request for access to an application resource on a back-end server. A Security Access Manager identity is obtained.

   Note: The single signon process is independent of the initial authentication method.
2. WebSEAL passes the Security Access Manager identity to the user registry server.
3. The registry returns a user name and password appropriate for the user and the requested application resource.
4. WebSEAL inserts the user name and password information in the HTTP BA header of the request or FSSO forms data. That request is sent across the junction to the back-end server.

GSO RESTful web service

When you configure the GSO capability for forms single sign-on (FSSO) and basic authentication (BA), you are required to specify a GSO resource. Use the following syntax to specify the GSO resource if you want to use a web service to communicate GSO data from an external resource.

```
[url]{<hostname>:<port>,}<uri>
```

[url] A static string that indicates a web service is used for this GSO resource. If a [url] designator is not present, the GSO credential information is retrieved from the Security Access Manager user registry.

<*hostname*>
The host name to use if the web service is accessed over a virtual host junction.

<*port*> The port to use if the web service is accessed over a virtual host junction.

<*uri*> The URI to which GSO requests are sent. This URI is relative to the root web space of the local WebSEAL server.

Here is an example resource that communicates with WebSEAL through a web service:

```
[url]www.ibm.com:80,/gso/credentials
```

The web service can be used to retrieve or update the credential information for a particular user.

To retrieve the GSO data for a particular user, the web service format is as follows:

```
Method:   GET

URI:      <web-service-root>/<isam-user-name>

Response: A successful response should include a status of 200 OK, with the
          following JSON data included in the body:
    * gso-username
    * gso-password
          If the credential information is not found, a 404 NOT FOUND response is returned.
```

To store new credential information for a particular user, the web service format is as follows:

```
Method:   PUT

URI:      <web-service-root>/<isam-user-name>

Body:     JSON data, which includes the following fields:
    * gso-username
    * gso-password

Response: A successful response includes a 204 No-Content with an empty body.
```

To delete the credential information for a particular user, the web service format is as follows:

```
Method:   DELETE

URI:      <web-service-root>/<isam-user-name>

Response: A successful response includes a 204 No-Content with an empty body.
```

By default, the passwords that are managed by the GSO RESTful web service are not obfuscated. If you want to obfuscate the passwords, set the **gso-obfuscation-key** stanza entry in the **[junction]** stanza to contain the obfuscation key. Setting this stanza entry enables password obfuscation for the GSO RESTful web service. For more information, see gso-obfuscation-key.

GSO junction learning

You can configure WebSEAL to learn your user name and password information so that future requests to the same junctioned resource will not prompt you for authentication.

Use the **gso-credential-learning** stanza entry to enable the GSO junction learning function. For more information, see gso-credential-learning.

If the GSO junction learning function is enabled, WebSEAL learns your user name and password information for a particular junctioned resource after you manually enter it for the first time. In future requests to the same junctioned resource, you will not be prompted for authentication as WebSEAL automatically provides such info. By default, the learning function is disabled.

Authentication information mapping

The following example illustrates how the user registry provides authentication information to WebSEAL.

If user Michael wants to run the **travel-app** application resource, WebSEAL asks the user registry server for Michael's authentication information. See the “Global sign-on overview” on page 524 section for details.

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 and password combination known as a **resource credential**. Resource credentials can be created only for registered users.

The registry contains a database for Michael that maps the resource **travel-app** to a specific resource credential.

The following table illustrates the structure of the GSO resource credential database:

Michael	**Paul**

resource: travel-app
username=mike
password=123
	resource: travel-app
	username=bundy
	password=abc

resource: payroll-app
username=powell
password=456
	resource: payroll-app
	username=jensen
	password=xyz

In this example, the registry returns user name "mike" and password "123" to WebSEAL. WebSEAL uses this information when it constructs the Basic Authentication header in the request sent across the junction to the back-end server.

Configuring a GSO-enabled WebSEAL junction

Use the **create** command with the **–b gso** option to create a junction that enables GSO.

About this task

Support for GSO is configured at the junction between WebSEAL and a back-end server.

The following example illustrates the syntax for the **create** command:

```
create -t tcp -h host-name -b gso -T resource jct-point
```

Options for setting up GSO junctions:

Options	Description
–b gso	Specifies that GSO must provide authentication information for all requests that cross this junction.
–T resource/resource-group	Specifies the GSO resource or resource group. The resource name that is used as the argument to this option must exactly match the resource name as listed in the GSO database. Required for GSO junctions.

A junction that is used in a WebSEAL or GSO solution can be made secure through SSL by applying the **–t ssl** option.

Always use SSL junctions with GSO to ensure encryption of credentials and all data.

Example

Junction the application resource **travel-app** on host **sales_svr** to junction point `/sales`:

```
create -t tcp -b gso -T travel-app -h sales_svr /sales
```

Junction the application resource **payroll-app** on host **adm_svr** to junction point `/admin` and make the junction secure with SSL:

```
create -t ssl -b gso -T payroll-app -h adm_svr /admin
```

Note: In this example, the **–t ssl** option dictates a default port of 443.

Configuration of the GSO cache

Use the global signon (GSO) cache function to improve the performance of GSO junctions in a high load environment.

By default, the GSO cache is disabled. Without the enhancement of the cache, a call to the user registry server is required for each retrieval of GSO target information.

Stanza entries for configuring the GSO cache are in the **[gso-cache]** stanza of the WebSEAL configuration file. You must first enable the cache. The remaining stanza entries configure the cache size and the timeout values for cache entries. Larger lifetime and inactivity timeout values improve performance, but increase the risk of information that is exposed in the WebSEAL memory. Do not enable the GSO cache if GSO junctions are not used in your network solution.

Stanza Entries	Description
gso-cache-enabled	Enable and disable the GSO cache function. Values are `yes` or `no`. Default is `no`.
gso-cache-size	Sets the maximum number of entries that are allowed in the cache hash table. Set this value to approximate the peak number of concurrent user sessions that access an application across a GSO junction. A high value uses more memory but results in faster information access. Each cache entry consumes approximately 50 bytes.

Stanza Entries	Description
gso-cache-entry-lifetime	Maximum time (in seconds) any cache entry can remain in the cache, regardless of activity. After a cache entry expires, the next request by that same user requires a new call to the user registry server. Default value is 900 seconds.
gso-cache-entry-idle-timeout	Maximum time (in seconds) an inactive cache entry that can remain in the cache. Default value is 120 seconds.

LTPA overview

WebSEAL can provide authentication and authorization services and protection to an IBM WebSphere environment. WebSphere provides support for the cookie-based lightweight third-party authentication mechanism (LTPA).

When WebSEAL is positioned as a protective front-end to WebSphere, users are faced with two potential login points. To achieve a single signon solution to one or more IBM WebSphere servers across WebSEAL junctions, you can configure WebSEAL junctions to support LTPA.

When a user makes a request for a WebSphere resource, the user must first authenticate to WebSEAL. After successful authentication, WebSEAL generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token for WebSphere, contains the user identity, key and token data, buffer length, and expiration information. This information is encrypted with a password-protected secret key that is shared between WebSEAL and the WebSphere server.

WebSEAL inserts the cookie in the HTTP header of the request that is sent across the junction to WebSphere. The back-end WebSphere server receives the request, decrypts the cookie, and authenticates the user. The user is authenticated based on the identity information that is supplied in the cookie.

To improve performance, WebSEAL can store the LTPA cookie in a cache and use the cached LTPA cookie for subsequent requests during the same user session. You can configure lifetime timeout and idle (inactivity) timeout values for the cached cookie.

WebSEAL supports both LTPA version 1 (LtpaToken) and LTPA version 2 (LtpaToken2) cookies. LTPA version 2 cookies are suggested for cases where the WebSphere server supports LtpaToken2.

- LtpaToken

  The LtpaToken is used for interoperating with previous releases of WebSphere Application Server. This token contains the authentication identity attribute only.

  LtpaToken is generated for releases before WebSphere Application Server Version 5.1.0.2 (for z/OS) or version 5.1.1 (for distributed).
- LtpaToken2

  LtpaToken2 contains stronger encryption and you can add multiple attributes to the token. This token contains the authentication identity and additional information such as the attributes. The attributes are used for contacting the original login server and the unique cache key. It is also used for looking up the Subject when you consider more than just the identity in determining uniqueness.

LtpaToken2 is generated for WebSphere Application Server Version 5.1.0.2 (for z/OS) and for version 5.1.1 (for distributed) and beyond.

For more information about using LTPA single signon in peer server environments, see "LTPA single signon" on page 560.

Configuration of an LTPA junction

Understand the configuration requirements and the commands you need to set up an LTPA junction.

Single signon to WebSphere with an LTPA cookie requires the following configuration tasks:

1. Enable the LTPA mechanism.
2. Provide the name of the key file that is used to encrypt the identity information.
3. Provide the password to this key file.
4. Ensure the LTPA cookie name for the WebSEAL junction matches the WebSphere LTPA cookie name.

   The name of the WebSEAL cookie that contains the LTPA token must match the configured name of the LTPA cookie in the WebSphere application. You can configure the **jct-ltpa-cookie-name** configuration item on a global or per junction basis. If you do not configure this cookie name, WebSEAL uses the same default values as WebSphere. See "Specifying the cookie name for junctions" on page 159.

The first three configuration requirements are specified in the following options to the standard junction and virtual host junction **create** commands.

- The **–A** option enables LTPA cookies.

  LTPA version 1 cookies (LtpaToken) and LTPA version 2 cookies (LtpaToken2) are both supported. LTPA version 1 cookies are specified by default. LTPA version 2 cookies must be specified with the additional **-2** option.

  Also requires –F, and –Z options.
- The **–2** option specifies that LTPA version 2 cookies (LtpaToken2) are used.

  The **-A** option without the **-2** option specifies that LTPA version 1 cookies (LtpaToken) are used.
- The **–F "keyfile"** option and argument specifies the name of the key file. The key file is used to encrypt the identity information in the cookie. The shared key is originally created on the WebSphere server and copied securely to the WebSEAL server. See the appropriate WebSphere documentation for specific details for this task.
- The **–Z " keyfile-password"** specifies the password that is required to open the key file.

  The password appears as encrypted text in the junction XML file.

Use these options in addition to other required junction options when you create the junction between WebSEAL and the back-end WebSphere server. For example, entered as one line:

```
pdadmin> server task default-webseald-webseal.ibm.com create ...
-A -F "/abc/xyz/key.file" -Z "abcdefg" ...
```

Configuration of the LTPA cache

The LTPA cache helps to improve the performance of LTPA junctions in a high load environment. Without the enhancement of the cache, a new LTPA cookie is created and encrypted for each subsequent user request.

The creation, encryption, and decryption of LTPA cookies introduces processing overhead. By default, the LTPA cache is enabled.

Stanza entries for configuring the LTPA cache are in the **[ltpa-cache]** stanza of the WebSEAL configuration file. Stanza entries specify the cache size and the timeout values for cache entries. Larger lifetime and inactivity timeout values improve performance, but increase the risk of information that is exposed in the WebSEAL memory.

Stanza Entries	Description
ltpa-cache-enabled	Enable and disable the LTPA cache function. Values include "yes" and "no". Default value is "yes".
ltpa-cache-size	Sets the maximum number of entries that are allowed in the cache hash table. Set this value to approximate the peak number of concurrent user sessions that access an application across an LTPA junction. A high value uses more memory but results in faster information access. Each cache entry consumes approximately 50 bytes. Default value is 4096 entries.
ltpa-cache-entry-lifetime	Maximum time (in seconds) any cache entry can remain in the cache, regardless of activity. After a cache entry expires, the next request by that same user requires the creation of a new LTPA cookie. Default value is 3600 seconds
ltpa-cache-entry-idle-timeout	Maximum time (in seconds) an inactive cache entry can remain in the cache. Default value is 600 seconds.

Technical notes for LTPA single sign-on

Understand the technical notes that you must consider when you implement LTPA single sign-on.

The following technical notes apply to LTPA single signon:

- The key file contains information about a specific WebSphere server. An LTPA junction is specific to one WebSphere server. If you add more than one server to the same junction point, all servers share the key file.
- For single signon to succeed, WebSEAL and the WebSphere server must share the registry information.
- The WebSphere server is responsible for setting up LTPA and the creation of the shared secret key. The WebSEAL participation involves the junction and cache configurations.
- WebSphere version 5.1.1 and later support the new LTPA version 2 cookie (LtpaToken2). In these environments, use the **-2** option to specify LtpaToken2 support.
- WebSEAL does not use WebSphere LTPA Security Attribute Propagation to pass more attributes to the WebSphere server in the LTPA cookie.

Forms single sign-on concepts

Forms single sign-on authentication supports existing applications that use HTML forms for authentication. It cannot be modified to directly trust the authentication that is done by WebSEAL.

Enabling forms single sign-on authentication produces the following results:

- WebSEAL interrupts the authentication process that is initiated by the back-end application
- WebSEAL supplies data that is required by the login form and submits the login form on behalf of the user.
- WebSEAL saves and restores all cookies and headers
- The user is unaware that a second login is taking place.
- The back-end application is unaware that the login form is not coming directly from the user.
- If the credential learning function is enabled, WebSEAL can learn the user name and password information so that future requests to the same junctioned resource does not prompt the user for authentication.

Configure WebSEAL:

- To recognize and intercept the login form
- To complete the appropriate authentication data

The administrator enables forms single signon by:

- Creating a configuration file to specify how the login form is to be recognized, completed, and processed
- Enable forms single signon by configuring the appropriate junction with the **–S** option (which specifies the location of the configuration file)

Forms single sign-on process flow

Learn about the single sign-on process so that you understand how a client browser accesses a resource.

The following scenario assumes that the user is authenticated in WebSEAL and that the credential learning function is disabled. When the credential learning function is enabled, the flow is discussed in “Forms single sign-on learning flow” on page 534.

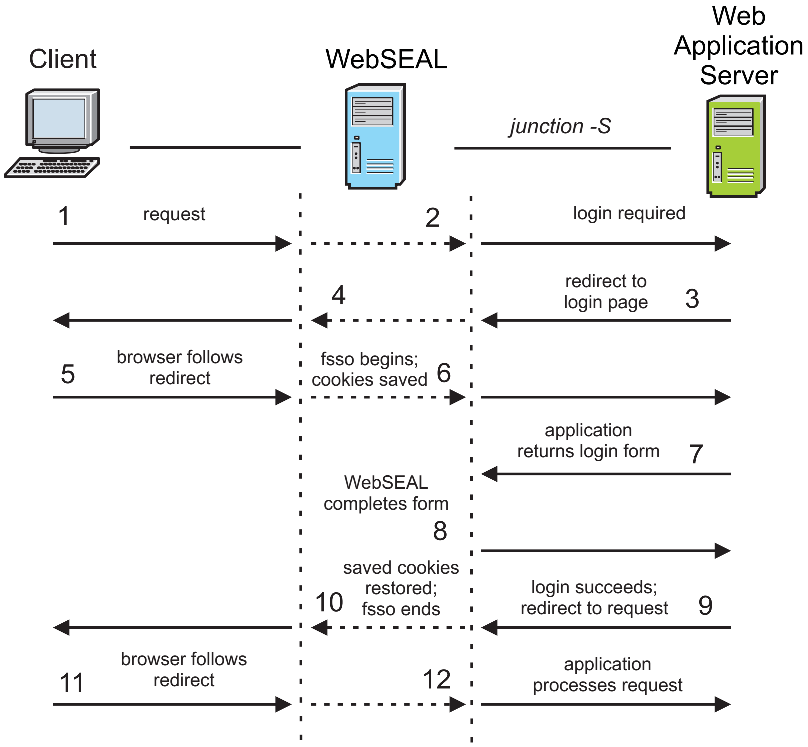

Client
WebSEAL
Web Application Server
junction -S
1 request
2
login required
4
redirect to login page 3
5 browser follows redirect
fsso begins; cookies saved 6
application returns login form 7
WebSEAL completes form
8
saved cookies restored; fsso ends
10
login succeeds; redirect to request 9
11 browser follows redirect
12
application processes request

Figure 48. Forms single sign-on process flow

1. Client browser requests the page:

   `https://webseal/formsso/content.html`
2. WebSEAL passes the request to the junction.
3. Because the back-end application requires the user to authenticate, a redirect to the application's login page (`login.html`) is sent back across the junction.
4. WebSEAL passes the redirect to the browser.
5. The browser follows the redirect and requests:

   `https://webseal/formsso/login.html`

   Note: Everything to this point in the process flow is standard WebSEAL function.
6. WebSEAL is configured for forms single sign-on (**–S** option on the junction). WebSEAL recognizes the request as a request for a login page, which is based on information in the forms SSO configuration file. The request is passed to the junction. WebSEAL saves all cookies that are sent by the browser for use in step 8.
7. The application returns the login page and application-specific cookies.

   WebSEAL parses the HTML returned to identify the login form. When WebSEAL finds an HTML form, it compares the action URI in the form to the value of the **`login-form-action`** stanza entry in the custom configuration file. If there is a match, WebSEAL uses the form that it found. Otherwise, WebSEAL keeps searching for other forms. If no form in the page matches the action URI pattern from the configuration file, then WebSEAL stops forms single sign-on processing. WebSEAL then returns an error to the browser. WebSEAL parses the page to identify the request method, the action URI, and any other input fields in the form. WebSEAL then them for use in step 8.

8. WebSEAL generates the authentication request (completes the login form) and sends it to the back-end application.
9. The application authenticates the user with the authentication data that is supplied by WebSEAL in the form. The application returns a redirect to `content.html`.
10. WebSEAL combines any cookies that are saved from the responses at step 7 and step 9, and returns these cookies with the redirect to the browser. If configured, **`login-success-pattern`** is used to inspect the response (even if credential learning is disabled). If the login is determined to be unsuccessful, the login page requested in step 5 is re-requested and returned to the user.

    Note: This process completes the forms SSO-specific function.
11. The browser follows the redirect and requests:

    ```
    https://webseal/formsso/content.html
    ```
12. WebSEAL passes the request to the back-end application across the junction.

During this process, the browser makes three requests to WebSEAL. From the user's perspective, only a single request for `https://webseal/formsso/content.html` is made. The other requests occur automatically through HTTP redirects.

Forms single sign-on learning flow

You can configure WebSEAL to learn your user name and password information so that future requests to the same junctioned resource will not prompt you for authentication.

Use the **`login-credential-learning`** stanza entry to enable the forms single sign-on learning function. Define what is a successful authentication with the **`login-success-pattern`** stanza entry.

A credential learning flow

In a successful learning flow, WebSEAL learns your user name and password information for a particular junctioned resource after you manually enter it for the first time. In future requests to the same junctioned resource, you will not be prompted for authentication as WebSEAL automatically provides such info.

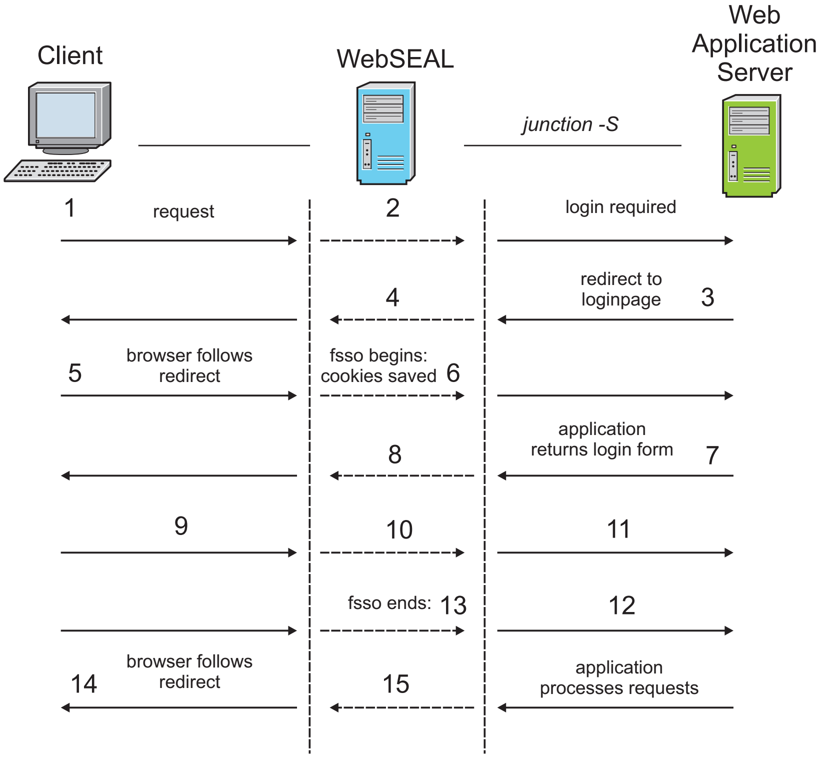


Figure 49. Forms single sign-on credential learning flow

1. Client browser requests the page:

   `https://webseal/formsso/content.html`
2. WebSEAL passes the request to the junction.
3. Because the back-end application requires the user to authenticate, a redirect to the application's login page (`login.html`) is sent back across the junction.
4. WebSEAL passes the redirect to the browser.
5. The browser follows the redirect and requests:

   `https://webseal/formsso/login.html`
6. WebSEAL is configured for forms single sign-on (**–S** option on the junction) with credential learning enabled. WebSEAL recognizes the request as a request for a login page, which is based on information in the forms SSO configuration file. The request is passed to the junction. WebSEAL saves all cookies that are sent by the browser for use in step 8.
7. The application returns the login page and application-specific cookies.
8. WebSEAL parses the HTML returned to identify the login form. WebSEAL detects the GSO resources associated with this junction. If WebSEAL cannot find a credential for the current user, it returns the response to the user.
9. The user populates this form and submits it.
10. WebSEAL inspects the request and extracts the credential information from the POST data.
11. The request is then sent to the backing application.
12. The application authenticates the user with the authentication data that is supplied by WebSEAL in the form.
13. The response is examined by WebSEAL. If the response is successful, the extracted credentials are persisted to the GSO vault. WebSEAL combines any

cookies that are saved from the responses and returns these cookies with the redirect to the browser. If the response is unsuccessful, WebSEAL disregards the credential and returns the response to the user.

Note: This process completes the forms single sign-on credential learning function.

14. In the case of a successful authentication, the browser follows the redirect and requests:

    `https://webseal/formsso/content.html`
15. WebSEAL passes the request to the back-end application across the junction.

A credential re-learning flow

There are several cases where the learning feature might need to re-learn a set of credentials. The primary cases are:

- The password is reset by some out-of-band method. For example, the user is emailed a new password or password reset link.
- The user changes the password through a form. For example, the user is required by the system to change the password upon authentication.

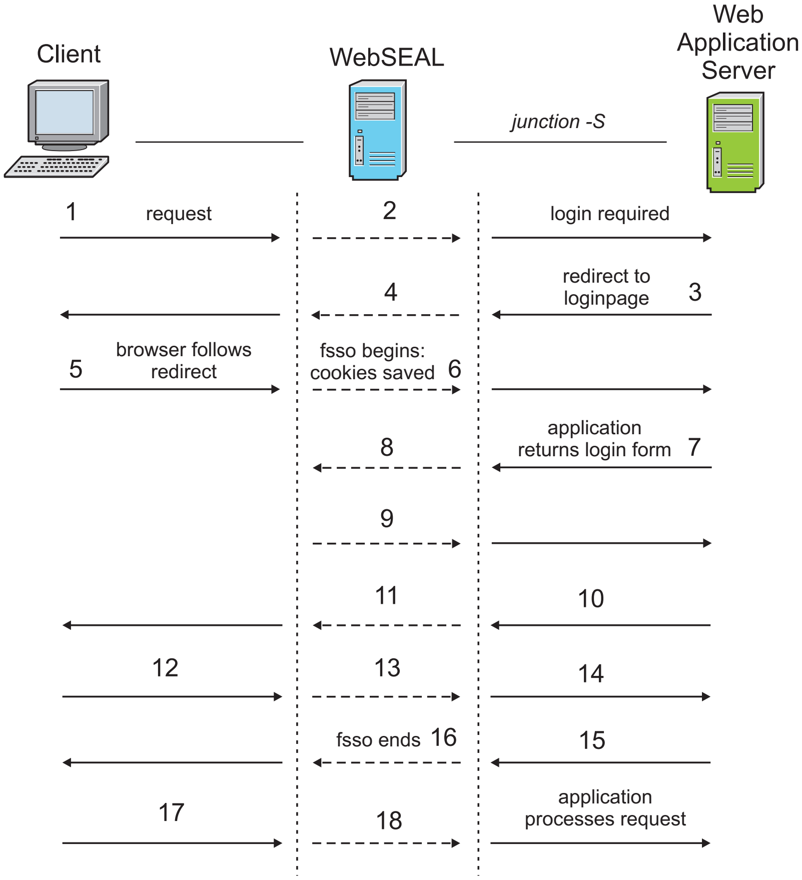


Figure 50. Forms single sign-on credential relearning flow

1. Client browser requests the page:

   `https://webseal/formsso/content.html`
2. WebSEAL passes the request to the junction.

3. Because the back-end application requires the user to authenticate, a redirect to the application's login page (`login.html`) is sent back across the junction.
4. WebSEAL passes the redirect to the browser.
5. The browser follows the redirect and requests:

   ```
   https://webseal/formsso/login.html
   ```
6. WebSEAL is configured for forms single sign-on (**–S** option on the junction) with credential learning enabled. WebSEAL recognizes the request as a request for a login page, which is based on information in the forms SSO configuration file. The request is passed to the junction. WebSEAL saves all cookies that are sent by the browser for use in step 8.
7. The application returns the login page and application-specific cookies.
8. WebSEAL parses the HTML returned to identify the login form.
9. WebSEAL detects the GSO resources associated with this junction and finds a credential for the current user. WebSEAL generates the authentication request (completes the login form) and sends it to the back-end application.
10. The authentication fails due to incorrect credentials.
11. WebSEAL determines that the response does not satisfy any of the login success rules that are configured with the **`login-success-pattern`** stanza entry. WebSEAL returns the login page to the client.
12. The user populates this login form and submits it.
13. WebSEAL inspects the request and extracts the credential information from the POST data.
14. The request is then sent to the back-end application.
15. The application authenticates the user with the authentication data that is supplied by WebSEAL in the form. The authentication succeeds. The application returns a redirect to `content.html`.
16. WebSEAL examines the response and detects that the response is successful. It updates the user's credential in the GSO vault with the extracted credential information. It also combines any cookies that were contained in the responses and returns these cookies with the redirect to the browser.

    Note: This process completes the forms single sign-on credential learning function.
17. The browser follows the redirect and requests:

    ```
    https://webseal/formsso/content.html
    ```
18. WebSEAL passes the request to the back-end application across the junction.

Example configuration

```
[forms-sso-login-pages]
login-page-stanza = myApp
# If this is a learning junction.
login-credential-learning = true

[myApp]
login-page = /login.jsp
#Successful login is a 302, failure is a 200
login-success-pattern = -200 +302 [location:*/landingPage]
login-form-action = /j_security_check
gso-resource = myGso
argument-stanza = myApp-arguments

[myApp-arguments]
j_username=gso:username
j_password=gso:password
```

Requirements for application support

Single signon for forms authentication is supported on applications that meet the specific requirements.

Applications must meet these requirements:

- The login page or pages for the application must be uniquely identifiable with a single regular expression or several regular expressions.
- The login page can include more than one HTML form. However, the login form must be identified by applying a regular expression to the action URIs of each of the login forms. Otherwise, the login form must be the first form in the login page.

  Note: If you use the action attribute to identify the login form, know that the action attribute did not pass through WebSEAL's HTML filtering. The regular expression must match the action URI before it is filtered.
- Client-side scripting can be used to validate input data. However, it must not modify the input data, such as using JavaScript to set cookies in the user browser.
- Login data is submitted at only one point in the authentication process.
- The junction where the authentication request is directed must be the same junction where the login page is returned.

Creation of the configuration file for forms single signon

The forms single signon configuration file is custom-created by the administrator and saved in any location.

The **–S** option on the junction enables the forms single signon functionality and specifies the name of the configuration file. See "How to enable forms single signon" on page 543. A sample configuration file (containing commented instructions) is available through the LMI. Go to **Secure Web Settings** > **Global Settings** > **Forms Based Single Sign-On** to create an FSSO configuration file.

The configuration file must begin with the **[forms-sso-login-pages]** stanza and has the following format

```
[forms-sso-login-pages]
login-page-stanza = xxxxx
#login-page-stanza = aaaaa
#login-page-stanza = bbbbb

[xxxxx]
login-page = regular-expression-page-match
login-form-action = regular-expression-form-match
gso-resource = gso-target
argument-stanza = yyyyy

[yyyyy]
name = method:value
```

The [forms-sso-login-pages] stanza

The forms single signon configuration file must always begin with the **[forms-sso-login-pages]** stanza.

The stanza contains one or more **login-page-stanza** entries that point to other custom-named stanzas. That stanza contains configuration information for the login pages that are found on the back-end application server.

The ability to support multiple login pages on a single junction is important. It is important because a single back-end server might host several applications that each use a different authentication method.

For example:

```
[forms-sso-login-pages]
login-page-stanza = loginpage1
login-page-stanza = loginpage2
```

If you want to enable the credential learning function on this junction, add a **login-credential-learning** stanza entry with a value of `true/yes/on` to the **[forms-sso-login-pages]** stanza. The **login-credential-learning** stanza entry specifies whether the login credential learning function is enabled for this junction. If no value is provided for this stanza entry, the value defaults to `false/no/off`.

The custom login page stanza

Each custom login page stanza is used to intercept a particular URL pattern. Understand the various stanza entries so that you know how to use them in various scenarios.

The stanza can contain the following stanza entries:

Stanza Entries	Description
login-page	This stanza entry specifies a pattern with a regular expression, that uniquely identifies requests for an application's login page. WebSEAL intercepts the pages and begins the forms single signon process. The regular expression is compared against the request URI and is relative to (and not including) the junction point where the server is mounted.
login-form-action	This stanza entry specifies a pattern, with a regular expression, that identifies which form contained in the intercepted page is the application's login form. The regular expression must match the "action" attribute of the login form. **Note:** Do not use `*` as the value of this stanza entry if the credential learning function is enabled. The credential learning function uses the value of **login-form-action** to detect a POST containing credentials that might need to be learned. The `*` value matches on every POST to the junction. So if this entry is set to `*`, WebSEAL inspects every POST to extract matching values and potentially persist the values in the GSO store, even though the request might not be related to forms single sign-on at all.
login-success-pattern	This stanza entry defines the rules to detect whether a login attempt is successful.
gso-resource	This stanza entry specifies the GSO resource to use when the GSO user name and password from a GSO database is received. Leave this stanza entry blank if GSO is not used to store a GSO user name and password. If you use an external GSO data source that WebSEAL communicates with through the GSO RESTful web service, use the syntax described in "GSO RESTful web service" on page 525.
argument-stanza	This stanza entry points to another custom stanza that lists the fields and data that is required for completing the login form.

Stanza Entries	Description
default-login-form-action	By default, the forms single sign-on engine tries to determine the login form action by parsing the form. In certain situations, parsing the form is not possible (for example, the action is constructed from JavaScript). This configuration entry can be used to define the action URI to be used if the action URI cannot be obtained from the form. If the learning function is configured and the **login-form-action** does not match on a login request, the **default-login-form-action** will also be checked.

For example:

```
[loginpage1]
login-page = /cgi-bin/getloginpage*
login-form-action = *
gso-resource =
argument-stanza = form1-data
```

About the login-page stanza entry:

The value of the **login-page** stanza entry is a regular expression. WebSEAL uses it to determine if an incoming request is a request for a login page. If so, WebSEAL intercepts this request and begins the forms single signon processing.

Only one **login-page** stanza entry is allowed in each custom login page stanza. You must create another custom login page stanza for each additional **login-page** stanza entry.

The **login-page** regular expression is compared against the request URI, relative to the junction. In the following example, the URI of a request to a WebSEAL server called "websealA" for a resource on a junction called "junctionX" appears as:

```
https://websealA.ibm.com/junctionX/auth/login.html
```

The part of this URL that is compared to the **login-page** regular expression:

```
/auth/login.html
```

About the login-form-action stanza entry:

The **login-form-action** stanza entry is used to identify the login form on the intercepted page. Only one **login-form-action** stanza entry is allowed in each stanza.

The value of the **login-form-action** stanza entry 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** stanza entry must match this path. It must match as it comes from the back-end server - even if it would normally be modified by WebSEAL before it is forwarded to the client.

If multiple "action" attributes on the page match the regular expression, only the first match is accepted as the login form.

If the regular expression does not match any form on the page, an error is returned to the browser. The error reports that the form might not be found.

Note: Do not use * as the value of this stanza entry if the credential learning function is enabled. The credential learning function uses the value of **`login-form-action`** to detect a POST containing credentials that might need to be learned. The * value matches on every POST to the junction. So if this entry is set to *, WebSEAL inspects every POST to extract matching values and potentially persist the values in the GSO store, even though the request might not be related to forms single sign-on at all.

About the login-success-pattern stanza entry

This stanza entry contains a space separated list of rules. Each rule is made up of 3 parts.

- The operator, either a + or - character. A + operator means this rule is a positive rule. If it is matched, the login was successful. A match on a rule with the operator - means the authentication was unsuccessful and processing should halt.
- The return code pattern. This pattern is used to match the value of the status code of the request. Wildcard characters such as ? and * can be used. For example, 4* matches on all 4*xx* status codes.
- A list of header rules. This value must be contained in [] brackets and is a list of pairs. Each pair is made up of a header name and a header value pattern. Each pair must be separated with a **,** character. This part is optional.

  Note: Only characters 0-9 and wildcard characters ? and * can be used in return code patterns.

The format of the stanza entry is:

```
{+/-}{ReturnCodePattern}{[HeaderName1:HeaderValuePattern1,
HeaderName2:HeaderValuePattern2,...]}
```

The rules for matching are as follows:

- The matching on header name is case insensitive. But the matching on value is case sensitive.
- This matching is done in the order the rules are defined and the first match halts further processing.

  Note: Order the rule from the most to least specific (per return code). For example, the rule sequence "-200 +200[SuccessfulLogin:True]" is poor logic, as the second rule would never be evaluated.
- Trailing white space on the header name and leading white space on the header will be removed on the configured value and when the header is inspected.
- If no rule matches, a negative result is the default.

Here is an example:

```
[forms-sso-login-pages]
login-page-stanza=fsso-cfg-args
...
[fsso-cfg-args]
...
#This configures 6 rules they are in order, as:
# 1. Succeed if the response is a 302, with a header that looks like the pattern
# "location:*/successful-login"
# AND there is a set-cookie header which matches "Set-Cookie:PD-S-SESSION-ID=1_2_1*"
# 2. Fail if the response is a 302 with the header which matches "location:*/failed-login"
# 3. Succeed if a 200 is returned with a header like "Set-Cookie:PD-S-SESSION-ID=1_2_1*"
# 4. Fail on any other 200
# 5. Fail on any response which starts with a 4 (400, 405, 415, etc)
# 6. Fail on any response which starts with 50 (502, 500, etc)
login-success-pattern = +302[location:*/successful-login,Set-Cookie:PD-S-SESSION-ID=1_2_1*]
-302[location:*/failed-login] +200[Set-Cookie:PD-S-SESSION-ID=1_2_1*] -200 -4* -50?
...
```

Use of regular expressions

Know the special characters that are allowed in a regular expression so that you can use them in the configuration file.

The following table lists the special characters that are allowed in regular expressions that are used in the forms single signon configuration file.

Table 67. Special characters allowed in regular expressions in the forms single signon configuration file

Special characters allowed	Description
*	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 (lowercase letter)
[^0-9]	Matches any character not 0 - 9 (not a number)
[a-zA-Z]	Matches any character between a and z (lowercase) or A and Z (uppercase)

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 asterisk (*) at the end of the expression. Use the asterisk (*) character so that any query data at the end of the URI does not prevent the login page from being matched.

The argument stanza

Understand how you can use the argument stanza and the variables that are contained within the stanza.

The custom argument stanza contains one or more entries in the following form:

```
name = method:value
```

name

The value of the **name** variable 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 variable can also use the value of the **name** attribute of the HTML **select** or **textarea** tags.

method:value

This combination retrieves the authentication data that is required by the form. The authentication data can include:

- Literal string data

```
string:text
```

The input that is used is the text string.

- GSO user name and password

  ```
  gso:username
  gso:password
  ```

  The input is the current user's GSO user name and password (from the target that is 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 can be accessed as:

  ```
  cred:azn_cred_authzn_id
  ```

  Custom credential attributes (added through 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
```

How to enable forms single signon

Learn the command options to enable forms single signon so that you can configure the appropriate junction to support forms single signon.

You must configure the appropriate junction to support forms single signon after you update the custom forms single signon configuration file. You must locate the file in an appropriate directory. Use the –S junction option with the **pdadmin create** command:

```
-S config-file
```

The **config-file** argument specifies the location of the custom forms single signon configuration file.

The **–S** junction option enables the forms single signon function on the junction. For example:

UNIX:

```
pdadmin> server task web1-webseald-cruz create -t tcp -h websvrA
-S fsso.conf /jctX
```

The configuration file is read when the junction is created and each time WebSEAL is started. Errors in the configuration file can cause WebSEAL to fail at start-up.

Forms single sign-on example

The following help site sample invokes its own forms-based login. The example shows 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 help application
- The custom forms single signon 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:&nbsp;
<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>
```

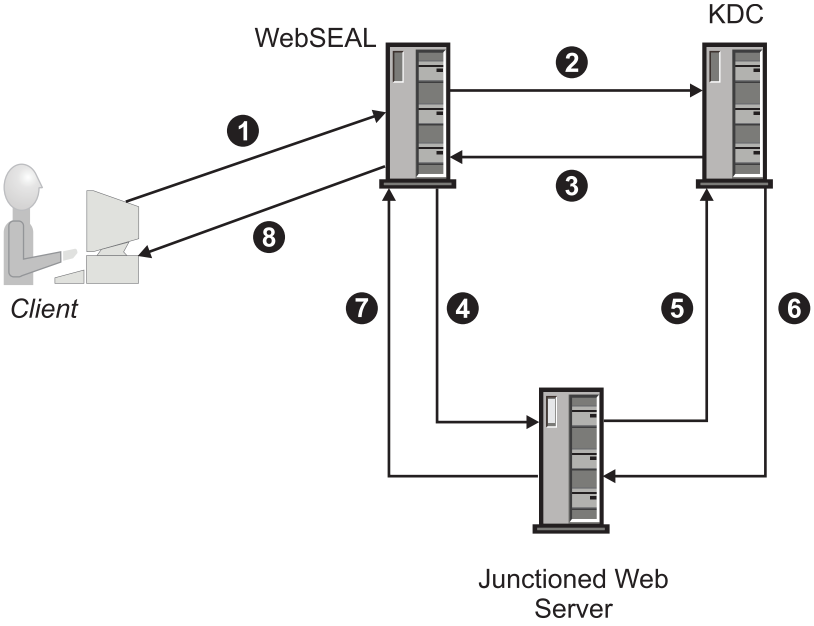
The custom configuration file used to process this form:

```
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
```

Single sign-on using Kerberos constrained delegation

You can set up constrained delegation by allowing WebSEAL to request a Windows Kerberos ticket on behalf of the client from the key distribution centre (KDC). The ticket can then be used by WebSEAL to impersonate the client to authenticate with the junctioned Web server.

Two extensions are involved in this process: **`Service-for-User-to-Self`** (**`S4U2Self`**) and **`Service-for-User-to-Proxy`** (**`S4U2Proxy`**). **`S4U2Self`** allows a service to acquire a ticket from the KDC on behalf of a client. **`S4U2Proxy`** allows a service to use the ticket obtained through **`S4U2Self`** to acquire another ticket to an external service.



The diagram above shows a sample deployment of single sign-on using Kerberos constrained delegation.

1. Client uses the standard Security Access Manager authentication process to authenticate to WebSEAL over HTTPS or HTTP and requests an object on the junctioned server. WebSEAL authorizes the request from the client, and determines that a Kerberos ticket is needed to access the junctioned application.
2. WebSEAL requests a Windows Kerberos ticket on behalf of the client from the key distribution centre (KDC).
3. KDC issues the Kerberos ticket to WebSEAL.
4. The WebSEAL server forwards the Kerberos ticket along with the client request to the junctioned Web server over either HTTP or HTTPS.
5. The junctioned Web server requests validation of the Kerberos ticket from the KDC.
6. The KDC verifies that the Kerberos ticket is valid.
7. The junctioned Web server returns an HTTP response to WebSEAL.
8. WebSEAL returns the HTTP response to the client.

To allow WebSEAL to perform Kerberos single sign-on for a junction, ensure that:

- Service users are created in Active Directory.
- The Kerberos Configuration and WebSEAL configuration file are updated on the appliance.
- The WebSEAL junction is created.

Kerberos tickets rely on embedded time stamps to decide the expiration of old tickets. For this reason, it is important to ensure that the clocks on all machines in the environment are synchronized.

Creating the WebSEAL user in Active Directory

In order for constrained delegation to operate correctly, WebSEAL and the target service must be running as Active Directory (AD) users that have an assigned Service Principal Name (SPN).

About this task

The new WebSEAL user account SPN that is created in Active Directory is used in the WebSEAL configuration as the `kerberos-principal-name`. The target service user account SPN is used as the `kerberos-service-name`. For instructions about how to configure Windows for constrained delegation, see the Developer Works article IBM Tivoli Access Manager: WebSEAL Kerberos Junctions.

To create the WebSEAL user and the target service user, complete the following steps.

Procedure

1. Create and initialize the AD WebSEAL user.
   a. On the domain controller, select **Start** > **Administrative Tools** > **Active Directory Users and Computers**.
   b. Create a new user whose password never expires. For example, `webseal`.
   c. Prepare the new user so that it can be used as the WebSEAL identity through a key table file. You can use the **ktpass** command line utility that is provided as a part of the Windows support tools to do this. After the SPN has been set for the user, change the login name to reflect the SPN. For example:
      -
        ```
        ktpass -out <Directory For Keytab> -princ HTTP/<WebSEALUser Name>@<AD DOMAIN NAME> \
        -mapUser <WebSEAL UserName> -mapOp set -pass <Password> -pType KRB5_NT_PRINCIPAL
        ```
      -
        ```
        ktpass -out C:\webseal.keytab -princ HTTP/webseal@AD_DOMAIN.COM
        -mapUser AD_DOMAIN\webseal -mapOp set -pass XXX -pType KRB5_NT_PRINCIPAL
        ```

      Note: See Ktpass for more details.
2. Create and initialize the AD target service user.
   a. On the domain controller, select **Start** > **Administrative Tools** > **Active Directory Users and Computers**.
   b. Create a new user whose password never expires. For example, `targetservice`.
   c. Prepare the new user so that it can execute as a Kerberos service. You can use the **ktpass** command line utility that is provided as a part of the Windows support tools to do this. After the SPN has been set for the user, change the login name to reflect the SPN. For example:
      -
        ```
        ktpass -princ HTTP/<Target Server Name>.<DNS domain name>@<AD DOMAIN NAME> \
        -mapuser <Target User Name> -mapOp set
        ```
      -
        ```
        ktpass -princ HTTP/target_service.ad_domain.com@AD_DOMAIN.COM
        -mapUser AD_DOMAIN\target_service
        -mapOp set -pass XXX -pType KRB5_NT_PRINCIPAL
        ```

3. By default, the security policy of the machine does not allow an AD user to execute a local service. You must change the setting so that the new AD user is allowed to execute a local service.
   a. Select **Start** > **Administrative Tools** > **Local Security Policy**.
   b. In the Local Security Policy window, select **Security Settings** > **Local Policies** > **User Rights Assignment**.
   c. In the right panel, double-click **Log on as a service**.
   d. Click **Add User or Group**.
   e. Enter the new AD user's name and then click **OK** to include the user in this policy.
4. Set the WebSEAL user to be trusted for delegation to the target service user.
   a. On the domain controller, select **Start** > **Administrative Tools** > **Active Directory Users and Computers**.
   b. Right-click the WebSEAL user.
   c. Click **Properties**.
   d. Select the **Delegation** tab.
   e. Select **Trust this user for delegation to specified services only** and **Use any authentication protocol**. For constrained delegation, **Use Kerberos only** cannot be used.
   f. Click **Add**.
   g. Click **Users and Computers**.
   h. Search for the target service user.
   i. Click **OK**.
   j. In the Add Services window, make sure the **HTTP** service is selected and then click **OK**.
   k. Click **OK** to save and exit the user properties.

   User accounts should also exist for the WebSEAL users that have permission to access the target service. These accounts do not require SPNs set-up with the `ktpass` tool.

WebSEAL Kerberos configuration

Complete the Kerberos configuration on the appliance so that single sign-on with Kerberos constrained delegation can work.

Procedure

1. From the top menu, select **Secure Web Settings** > **Global Settings** > **Kerberos Configuration**.
2. On the **Realms** tab, select **New** > **Realm**.
3. Enter the AD domain name. For example, `<DOMAIN>`.
4. Click **Save**.
5. Select the new realm.
6. Click **New** > **Property**.
7. In the Create New Property window, select **kdc**.
8. Enter the AD KDC address in the **Value** field. The AD KDC address is the name of the domain controller. For example, `<machine>.<domain>`.
9. Click **Save**.
10. On the **Defaults** tab, change the **default_realm** to be the new realm that you just created.

11. On the **Keyfiles** tab, import the key table file that was generated for the WebSEAL user.
12. Deploy the changes.
13. From the top menu, select **Manage System Settings** > **Network Settings** > **Hosts File**.
14. Add the AD domain and KDC addresses to the hosts file.

    Note: This step is only necessary if the DNS is not configured.
15. Deploy the changes.

Configuring WebSEAL to enable Kerberos single sign-on

To enable Kerberos single sign-on for a junction, set the value of the **kerberos-sso-enable** entry in the **[junction]** stanza to yes.

About this task

For more information about the **[junction]** stanza, see [junction] stanza.

Procedure

1. From the top menu, select **Secure Web Settings** > **Manage** > **Reverse Proxy**.
2. Create a new WebSEAL instance.
3. Select the instance.
4. Click **Manage** > **Configuration File**.
5. Locate the **[junction]** stanza.
6. Update the configuration items accordingly. For example:
   ```
   kerberos-sso-enable = yes
   kerberos-keytab-file = webseal.keytab
   kerberos-principal-name = HTTP/webseal@AD_DOMAIN
   kerberos-service-name = HTTP/target_service.ad_domain.com@AD_DOMAIN.COM
   ```

   Note: These SPNs are set in Active Directory in “Creating the WebSEAL user in Active Directory” on page 546. The domain names are case-sensitive and must be uppercase.

   To extend Kerberos SSO support to users on domains other than the WebSEAL service account domain, use the kerberos-user-identity stanza entry to enable and define a custom user principal name (UPN).
7. Click **Save**.
8. Deploy the changes.
9. Restart the WebSEAL instance.

Cross-domain single signon concepts

This section contains the following topics:

Cross-domain single signon overview

Cross-domain single signon (sometimes referred to as CDSSO) provides a default mechanism for transferring user credentials between unique servers and domains. CDSSO allows Web users to perform a single signon and move seamlessly between two separate secure domains when requesting a resource. The CDSSO authentication mechanism does not rely on a master authentication server (sometimes referred to as the MAS) (see “E-community single signon” on page 561).

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 signon.

When a user makes a request to a resource located in another domain, the CDSSO mechanism transfers an encrypted user identity token from the first domain to the second domain. The identity information in this token indicates to the receiving domain that the user is successfully authenticated in the first domain. The identity does not contain password information. The receiving server uses this token to build credentials in its own cache for that user. The user is not forced to perform an additional login.

Default and custom authentication tokens

Cross-domain single signon solutions employ authentication tokens that convey an encoded version of the user identity to the destination server. The construction of these tokens by the initial server is called "token creation". The decoding and use of the token by the destination server is called "token consumption". WebSEAL uses built-in token create and token consume modules to allow default CDSSO operation.

Alternatively, you can build custom token create and consume modules to meet the specific requirements of your network and Security Access Manager implementation. Complete information and API reference material for cross-domain external authentication can be found in the *IBM Security Access Manager for Web: Web Security Developer Reference*.

Extended user attributes and identity mapping

The CDSSO process is supported by the cross-domain mapping framework (sometimes referred to as CDMF) to allow the inclusion of extended attributes to further describe a user identity. CDMF is a programming interface that can handle extended user attributes during token creation and provide mapping services for the user identity during token consumption.

Built-in default CDMF operation during CDSSO returns:

- "SUCCESS" and no extended attributes to the CDSSO token creation module
- "SUCCESS" and no identity mapping to the CDSSO token consumption module.

You can use the cross-domain mapping framework C API to customize the handling of user attributes and the mapping of user identities. Complete information and API reference material for the cross-domain mapping framework can be found in the *IBM Security Access Manager for Web: Web Security Developer Reference*.

Alternatively, attributes can be specified in the WebSEAL configuration file for transfer from the source server to the destination server.

CDSSO process flow with attribute transfer and user mapping

The following process flow description is illustrated in Figure 51 on page 551.

1. Any user who wants to participate in multiple domains must have a valid user account in the initial domain and an identity that can be mapped into a valid account in each of the participating remote domains.

   A user cannot invoke the CDSSO functionality without initially authenticating to an initial secure domain (A) that contains the user's account.
2. The user makes a request to access a resource in domain B using a custom link on a Web page located on the server **websealA**.

   The link is constructed using a special CDSSO management page expression:

   `/pkmscdsso?`*`destination-URL`*

   For example:

   ```
   http://websealA/pkmscdsso?https://websealB/resource.html
   ```

   Note: The `pkmscdsso` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.
3. The request is first processed by the **websealA** server in domain A. The **websealA** server calls the **ssocreate** module to build an authentication token that contains the user's credentials, including the Security Access Manager identity (short name), the current domain ("A"), additional user information, and a time stamp.
4. The token create module can obtain extended user attribute information that can be used by domain B during the user mapping process.

   Attribute information can come from two sources. First, the **[cdsso-token-attributes]** stanza of the WebSEAL configuration file is checked for configured stanza entries. Secondly, the CDMF library is called (**cdmf_get_usr_attributes**) to obtain additional attributes. Attributes from the CDMF library override any settings in the **[cdcsso-token-attributes]** stanza.
5. WebSEAL triple-DES encrypts this token data with the symmetric key generated by the LMI. This key file is shared and specified in the **[cdsso-peers]** stanza of the WebSEAL configuration file on both domain A and domain B WebSEAL servers.
6. 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.
7. The token is placed in a redirected request to the destination server, using the URL contained in the **pkmscdsso** link. For example:

   `http://websealB/resource.html?PD-ID=`*`encoded-authn-token`*`&PD-REFERER=websealA`

   The PD-ID argument name in the query string is obtained from the **cdsso-argument** stanza entry of the WebSEAL configuration file.

   The **ssocreate** module provides the PD-REFERER argument with the name of the origin server (**websealA**) so that the destination server (**websealB**) accurately knows where the redirected request is from, without relying on the browser's Referer header information.
8. The **websealA** server sends a redirect response for the resource on **websealB** containing the encrypted token back to the browser.
9. The **websealB** server recognizes the request as one containing a CDSSO token (based on the value of the **cdsso-argument** stanza entry of its WebSEAL configuration file).

10. The **webseal B** server decodes and validates the token as coming from the referring domain. This process is performed by the "token consume" authentication module (**ssoconsume**).
11. When constructing the attribute list for the new identity, the token consume module first processes the attributes according to the settings in the **[cdsso-incoming-attributes]** stanza of the WebSEAL configuration file. Then the module calls to the CDMF library, which performs the actual user mapping (**cdmf_map_usr**).

    The CDMF library passes the user's mapped identity, and any extended attribute information, back to the token consume module. The token consume module passes the identity to the WebSEAL server, which builds a credential.
12. The **webseal B** authorization service permits or denies access to protected objects based on the user's credential and the specific ACL and POP permissions associated with the requested objects.

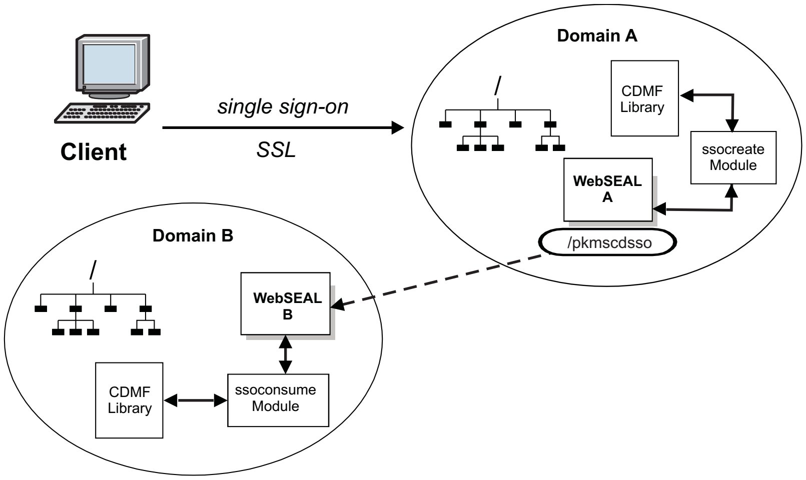


Figure 51. Cross-domain single signon process with CDMF

Configuration of cross-domain single signon

This section contains the following introductory topics:

The main configuration tasks are described in the following topics:
1. "Enabling and disabling CDSSO authentication" on page 553
2. "Encrypting the authentication token data" on page 554
3. "Configuring the token time stamp" on page 554
4. "Configuring the token label name" on page 555
5. "Creating the CDSSO HTML link" on page 555

The remaining topics provide supporting information:

CDSSO configuration summary

The following configuration steps are explained in detail in the remaining sections of this CDSSO chapter division.

Configuring CDSSO token create functionality

About this task

The following procedures are appropriate for the initial WebSEAL server:

Procedure

1. Enable WebSEAL to generate CDSSO tokens (**cdsso-create**).
2. Configure the built-in token creation module (**sso-create**).
3. Create the key file used to encode and decode the token. Copy the key file to all appropriate participating servers (**[cdsso-peers]** stanza).
4. Configure the token time stamp (**authtoken-lifetime**)
5. Configure the token label (**cdsso-argument**).
6. Create the CDSSO HTML link (**/pkmscdsso**).

Configuring CDSSO token consume functionality

About this task

The following procedures are appropriate for the destination WebSEAL server:

Procedure

1. Enable WebSEAL to consume CDSSO tokens (**cdsso-auth**) for authentication.
2. Configure the built-in token consumption module (**sso-consume**).
3. Assign the appropriate key file (**[cdsso-peers]** stanza).
4. Configure the token time stamp (**authtoken-lifetime**)
5. Configure the token label (**cdsso-argument**).

CDSSO conditions and requirements

- All WebSEAL servers participating in CDSSO must have machine times synchronized. Authentication between servers can fail when machine time differences are too great.
- For CDSSO to function successfully, each participating WebSEAL server must reveal its fully qualified host name to the other participating servers in the cross-domain environment. If any host name does not include a domain, CDSSO cannot be enabled and an error message is logged in `msg_webseald.log`. When setting up a CDSSO environment, ensure that the machine-specific networking setup for each participating server is configured to identify the server with a fully qualified host name.
- Because some WebSEAL configuration requires machine host names to be described as fully qualified host names, you must ensure that your system and network can resolve machine names into fully qualified host names. For example, using fully qualified host names allows for many host names (IP addresses) per machine, as in the case of multiple WebSEAL instances.
- Do not reuse key pairs (used to encrypt and decrypt token data) generated for a specific CDSSO environment in any other CDSSO environments. Always generate unique key pairs for each CDSSO environment.

Resolving machine names

CDSSO can be unintentionally disabled upon WebSEAL startup because the machine itself is not adequately configured to resolve machine names.

About this task

The machine on which WebSEAL is located needs to be able to fully resolve an IP address. You can use the LMI to configure the DNS.

The following general information is provided only as an example:

Goal: Configure the machine to first look to DNS before checking the local /etc/hosts file for DNS information.

Procedure

1. Make sure that /etc/resolv.conf has valid DNS server entries.
2. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:
   ```
   hosts dns files
   ```
   Alternative goal: Configure the machine to first use local DNS information (/etc/hosts) before checking DNS.

Results

1. Configure the machine to check /etc/hosts before looking to DNS. Edit /etc/nsswitch.conf so the hosts line indicates the correct order for checking DNS information:
   ```
   hosts files dns
   ```
2. Enter appropriate DNS information in /etc/hosts:
   ```
   webseal1.fully.qualified.com 1.11.111.111
   webseal2.fully.qualified.com 2.22.222.222
   ```

Enabling and disabling CDSSO authentication

About this task

Configuration for the initial WebSEAL server:

To enable the CDSSO token create module, configure the following stanza entry in the WebSEAL configuration file:

```
[cdsso]
cdsso-create = {none|http|https|both}
```

The value both specifies both HTTP and HTTPS protocols. The value none disables the token create module.

Configuration for the destination WebSEAL server:

To enable the CDSSO token consume module, configure the following stanza entry in the WebSEAL configuration file:

```
[cdsso]
cdsso-auth = {none|http|https|both}
```

The value both specifies both HTTP and HTTPS protocols. The value none disables the token consume module.

Note: You must stop and restart the WebSEAL server in order to activate changes to the WebSEAL configuration file. Complete all of the applicable configuration steps in this section and then restart WebSEAL.

Encrypting the authentication token data

About this task

WebSEAL must encrypt the authentication data placed in the token using a key generated through the LMI. You must "synchronize" this key by sharing the key file with each participating WebSEAL server in each participating domain. Each participating WebSEAL server in each domain needs to use the same key.

Note: Do not reuse key pairs (used to encrypt and decrypt token data) generated for a specific CDSSO environment in any other CDSSO environments. Always generate unique key pairs for each CDSSO environment.

The generated key is a triple DES 192 bit key. You cannot specify a life span time on this key.

Note: The distribution of key files is not a part of the Security Access Manager CDSSO process.

Specify this key file in the **[cdsso-peers]** stanza of the WebSEAL configuration file of the participating WebSEAL server in each domain. The format must include the fully qualified host name of the WebSEAL server and the name of the key file:

```
[cdsso-peers]
fully-qualified-host-name = keyfile-name
```

Configuration example for server websealA in domain A:

```
[cdsso-peers]
websealB.domainB.com = A-B.key
```

This setting specifies what key websealA uses to encrypt a token destined for websealB in domain B.

Configuration example for server websealB in domain B:

```
[cdsso-peers]
websealA.domainA.com = A-B.key
```

This setting specifies what key websealB (in domain B) uses to decrypt a token received from websealA in domain A.

In this example, the A-B.key file is generated on one machine (websealA, for example) and manually (and securely) copied to the other machine (websealB, for example).

Configuring the token time stamp

About this task

The token contains a configurable time stamp that defines the lifetime of the identity token. After the time stamp has expired, the token is considered not valid 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** stanza entry, located in the **[cdsso]** stanza of the WebSEAL configuration file, sets the token lifetime value. The value is expressed in seconds. The default value is 180:

```
[cdsso]
authtoken-lifetime = 180
```

You must take into account any clock skew among the participating domains. Clock skew means that the system times differ on the relevant servers in each domain.

When this difference approaches the value of `authtoken-lifetime`, the effective lifetime of the token is greatly reduced. When this difference exceeds the value of `authtoken-lifetime`, tokens from one domain cannot be valid for the other domain.

You should adjust `authtoken-lifetime` accordingly. However, when clock skew requires that `authtoken-lifetime` be set to a large value, the risk of replay attacks increases. In this case, you should consider synchronizing the system time on the relevant servers in each domain.

For more information, see authtoken-lifetime.

Configuring the token label name

About this task

The authentication information used for a single signon transaction is placed in the redirected request as an encrypted token query string argument to the request. This token string requires a name, or label, to identify it. The label name uniquely identifies the request to the receiving WebSEAL server as a single signon request to be handled by the CDSSO token consume mechanism (module).

You must configure this token label identically on both WebSEAL servers participating in the single signon functionality. To configure the token label, use the **cdsso-argument** stanza entry located in the **[cdsso]** stanza of the WebSEAL configuration file. For example (default):

```
[cdsso]
cdsso-argument = PD-ID
```

For more information, see the cdsso-argument.

Creating the CDSSO HTML link

About this task

The HTML link (located on the original WebSEAL server) that connects the user to a resource on the destination WebSEAL server must use a special CDSSO expression that directs the request to a CDSSO management page **pkmscdsso**:

```
/pkmscdsso?destination-URL
```

For example:

```
http://websealA/pkmscdsso?https://websealB/resource.html
```

Note: The pkmscdsso management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

The token create module creates and encodes an authentication token (containing the user's identity information) and includes this token in a redirect response to the resource using the destination URL information from the CDSSO link. For example:

```
http://websealB/resource.html?PD-ID=encoded-authn-token&PD-REFERER=websealA
```

Handling errors from CDMF during token creation

About this task

During the creation of a CDSSO token, the **ssocreate** module calls the CDMF library to acquire any extended attributes to be included in the token. Extended attributes (that further describe the user) can be required for successful identity mapping of the user on the destination server. The CDMF API uses the **cdmf_get_usr_attributes** call to acquire extended attributes.

It is possible for the **cdmf_get_usr_attributes** call to fail in obtaining the required information and return an error. In such a case, the subsequent behavior of the token creation process can be controlled through use of the **propagate-cdmf-errors** stanza entry located in the **[cdsso]** stanza. Values for this stanza entry include "yes" and "no".

A "no" value (default) allows the token creation process to proceed even when CDMF fails to obtain attributes and returns an error.

A "yes" value forces the token creation process to end when CDMF fails to obtain attributes and returns an error.

Example:

```
[cdsso]
propagate-cdmf-errors = no
```

Protection of the authentication token

While the authentication token does not contain authentication information (such as user name and password), it does contain a user identity that is trusted within the receiving domain. The token itself must therefore be protected against theft and replay.

The token is protected through the use of SSL to secure communications between the WebSEAL servers and the users. The token could conceivably be stolen from the user's browser history. The time stamp on the token should be short enough to make it unlikely that the token could be stolen and replayed during the lifetime of the token.

However, a token that has expired with respect to its time stamp is still vulnerable to cryptographic attacks. If the key used to encrypt the token is discovered or otherwise compromised, malicious users could build their own tokens.

Then, the tokens can be inserted into a pseudo-CDSSO flow. They would be indistinguishable from real authentication tokens to the WebSEAL servers participating in the CDSSO domain. For this reason, the keys used to protect the tokens must also be carefully managed and changed on a regular basis.

Use of cross-domain single signon with virtual hosts

See “Cross-domain single sign-on with virtual hosts” on page 495.

Extended attributes for CDSSO

CDSSO can handle extended attributes in two ways.

Two ways to handle extended attributes:

- The **[cdsso-token-attributes]** stanza of the WebSEAL configuration file is checked for configured stanza entries.
- The CDMF library is called (**cdmf_get_usr_attributes**) to obtain additional attributes.

Attributes from the CDMF library override any settings in the **[cdcsso-token-attributes]** stanza.

This section describes the use of the **[cdsso-token-attributes]** stanza and contains the following topics:

Extended attributes to add to token

In the WebSEAL configuration file, you can specify extended attributes from a user credential to add to the cross-domain single signon token. Extended attributes consist of information about a user identity that is added to an extended attribute list when a user credential is created. Extended attributes can be added by a number of authentication mechanisms, including custom authentication modules. The custom authentication modules can be used, for example, to obtain user information from a registry that is external to Security Access Manager.

You can use this setting to customize the contents of the cross-domain single signon token. This feature enables you to tailor the token contents to match the needs of the destination domain. When you use this feature to add an attribute to a token, you must also configure the WebSEAL configuration file for the server in the destination domain. For the destination server, the **[cdsso-incoming-attributes]** stanza is used to specify the handling (extract or ignore) of each attribute.

You can specify extended attributes by name, or you can declare a pattern that matches multiple attribute names. You can use standard Security Access Manager wildcard-matching characters. For a list of supported wildcard pattern matching characters, see “Supported wildcard pattern matching characters” on page 57.

Each entry is assigned the name of the domain for which the token is intended. You can include multiple entries specifying names or patterns for each domain.

The syntax is:

```
[cdsso-token-attributes]
domain_name = pattern1
domain_name = pattern2
...
domain_name = patternN
<default> = pattern1
<default> = pattern2
...
<default> = patternN
```

The <default> entry is optional. When WebSEAL does not find an entry that matches the domain name, WebSEAL looks for a <default> entry. If the configuration file contains a <default> entry, WebSEAL uses the assigned attribute patterns for the current domain. The string <default> is a keyword, and must be specified exactly as shown above, including the "<" and ">" characters.

Example: You are creating a cross-domain single signon solution between two domains: example1.com and example2.com. Users log in to example1.com but can get redirected to example2.com during the user session. Your deployment includes a customized external authentication C API module that inserts information into each user credential. The information includes a fixed name attribute "job_category" and a variable number of attributes, each prefixed with the characters "my_ext_attr_". This information needs to be added to the cross-domain token. The configuration file entries would be:

```
example2.com = job_category
example2.com = my_ext_attr_*
```

Extended attributes to extract from a token

In the WebSEAL configuration file, you can specify how the token consume module handles extended attributes that have been added to a cross-domain single sign-on token.

The attributes can be either extracted or ignored. In some cases, you must extract the attributes because there is no way for a server in the destination domain to generate them. In other cases, you do not want to extract the tokens, because the server in the destination domain can use an independent process to gather the same extended attributes. For example, the attribute can reflect a timestamp that needs to reflect the system time on the destination server.

In the token consume module, attributes that are extracted from the token are passed through to the cross-domain mapping framework module. The default cross-domain mapping framework module passes attributes directly through to the user credential. Customized cross-domain mapping framework modules can manipulate attributes as needed before passing them to the user credential.

The syntax for the entries is as follows:

```
[cdsso-incoming-attributes]
attribute_pattern = {preserve|refresh}
```

Typically, the names of the extended attributes (*attribute_pattern*) match the names of the attributes specified in the **[cdsso-token-attributes]** stanza of the configuration file for a WebSEAL server that generates the tokens. The value must be one of the following keywords:

- preserve

  Extract all attributes that match *attribute_pattern*.
- refresh

  Do not extract attributes that match *attribute_pattern*.

Extended attributes in the token that do not match an entry in **[cdsso-incoming-attributes]** are preserved (extracted).

The order of the entries in the stanza is important. The first entry that matches an attribute name is used. Other entries are ignored. For example, if you want to extract the attribute named **my_special_attr1** but want to ignore all other entries with the prefix my_special_attr_, the stanza entries should be:

```
[cdsso-incoming-attributes]
my_special_attr1 = preserve
my_special_attr_* = refresh
```

Using the examples shown above in "Extended attributes to add to token" on page 557, the entries in the WebSEAL configuration file for a server that operates in the example2.com domain could be:

```
[cdsso-incoming-attributes]
job_category = preserve
my_cdas_attr_1 = preserve
my_cdas_attr_* = refresh
```

In this example, the attributes **job_category** and **my_cdas_attr_1** are extracted from the token. The remainder of the attributes with the prefix my_cdas_attr_ are ignored.

Compatibility issues for CDSSO

This section contains the following topics:

UTF-8 encoding of tokens for cross domain single signon

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

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

The default value is true.

When **use-utf8** is set to "false", strings are encoded using the local code page. Use this value when implementing cross domain single signon with pre-version 5.1 WebSEAL servers. WebSEAL servers from versions prior to version 5.1 do not use UTF-8 encoding for tokens. When deploying WebSEAL in an environment that includes these older servers, configure WebSEAL to not use UTF-8 encoding. This setting is necessary for backward compatibility.

Token compatibility with earlier versions of WebSEAL

Security Access Manager, version 8.0, changed the default cipher that WebSEAL uses to encode the tokens. Tokens that are generated with this default cipher are not compatible with the tokens that are generated by earlier versions of WebSEAL.

If you integrate CDSSO with servers that include versions of Security Access Manager earlier than version 8.0, you must enable the **pre-800-compatible-tokens** stanza entry in the **[server]** stanza of the WebSEAL configuration file. For example:

```
pre-800-compatible-tokens = yes
```

The default setting for this stanza entry is no.

LTPA single signon

This chapter discusses single sign-on access to peer security servers using an LTPA cookie for authentication.

This section contains the following topics:

LTPA single signon overview

Single signon solutions across junctions describes how WebSphere provides single sign-on to junctioned servers using lightweight third-party authentication mechanism (LTPA). LTPA version 2 can also be used to provide single signon to peer servers.

When WebSEAL is positioned within an environment with other authentication enabled servers (e.g. DataPower) there are many potential login point. To achieve a single signon solution to one or more WebSphere or DataPower servers you can configure WebSEAL to accept and generate LTPA cookies.

When a user makes a request for a WebSEAL protected resource, the user must first authenticate to WebSEAL. After successful authentication, WebSEAL generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token, contains the user identity, key and token data, buffer length, and expiration information. This information is encrypted using a secret key shared between WebSEAL and the other LTPA-enabled servers.

WebSEAL inserts the cookie in the HTTP response which is sent back to the client. The LTPA enabled server receives this cookie upon the next request, decrypts the cookie, and authenticates the user based on the identity information supplied in the cookie.

WebSEAL only supports LTPA version 2 (LtpaToken2) cookies.

Configuring LTPA single signon

About this task

LTPA cookies are generated when LTPA authentication is enabled within WebSEAL. These cookies can then be used to achieve single signon to other LTPA-enabled authentication servers. For further details, see “LTPA authentication” on page 158.

Procedure

Single signon to other LTPA-enabled servers using an LTPA cookie requires the following configuration tasks:

1. Enable the LTPA mechanism.
2. Provide the name of the key file used to encrypt the identity information.
3. Provide the password to this key file.
4. Ensure the LTPA cookie name for the WebSEAL junction matches the WebSphere LTPA cookie name.

   The name of the WebSEAL cookie containing the LTPA token must match the configured name of the LTPA cookie in theWebSphere application. You can configure the **jct-ltpa-cookie-name** configuration item on a global or per junction basis. If you do not configure this cookie name, WebSEAL uses the same default values as WebSphere. See “Specifying the cookie name for junctions” on page 159.

The first three configuration requirements are specified in the options to the standard junction and virtual host junction **create** commands. Use these options in addition to other required junction options when you create the junction between WebSEAL and the back-end WebSphere server. For example (entered as one line):

```
pdadmin> server task default-webseald-webseal.ibm.com create ...
-A -F "key.file" -Z "abcdefg" ...
```

These options are further described in “LTPA authentication” on page 158.

Technical notes for LTPA single signon

The following technical notes apply to LTPA single signon.

- The key file contains information about a specific LTPA enabled authentication server. A single key file is used by WebSEAL when generating/authenticating LTPA cookies and as such all of the LTPA enabled server must share the same key file. If you add more than one server to the same junction point, all servers share the same key file.
- For single signon to succeed, WebSEAL and the LTPA enabled authentication server must share the same registry information.
- The LTPA-enabled server is responsible for setting up LTPA and the creation of the shared secret key.
- WebSEAL only supports LTPA version 2 cookies.
- WebSEAL does not use WebSphere LTPA Security Attribute Propagation to include additional attributes within the LTPA token.

E-community single signon

E-community single signon (sometimes referred to as ECSSO) allows authenticated users to access protected resources across multiple servers in multiple domains without requiring additional logins.

This chapter discusses the concepts and configuration steps required to implement an e-community single signon solution.

Topic Index:

E-community single signon concepts

This section contains the following topics:

E-community overview

E-community single signon (sometimes referred to as ECSSO) is another implementation of cross-domain authentication in a Security Access Manager environment. The goal of cross-domain authentication is to allow users to access resources across multiple servers in multiple domains without having to perform multiple logins.

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 disparate businesses with a shared relationship (for example, company headquarters, a life insurance company, and a financial management company).

In either scenario, there is always one domain that is designated the "home" or "owner" domain. In the case of participating businesses, the home domain owns the business agreements that govern the e-community.

In both scenarios, authentication information about the users who participate in the e-community (including the user names and passwords used for authentication) is maintained in the home domain. This arrangement allows a single point of reference for administration issues, such as help desk calls within the e-community that all refer to the home domain.

Alternatively, you can use the Web Portal Manager to delegate the management of this information that participating domains have responsibility for the administration of their own users.

The diagram below illustrates a sample e-community with two participating domains: domain A (dA.com) and domain B (dB.com). In this example, domain A represents the home or owner domain. Domain B is a participating, or "remote", domain.

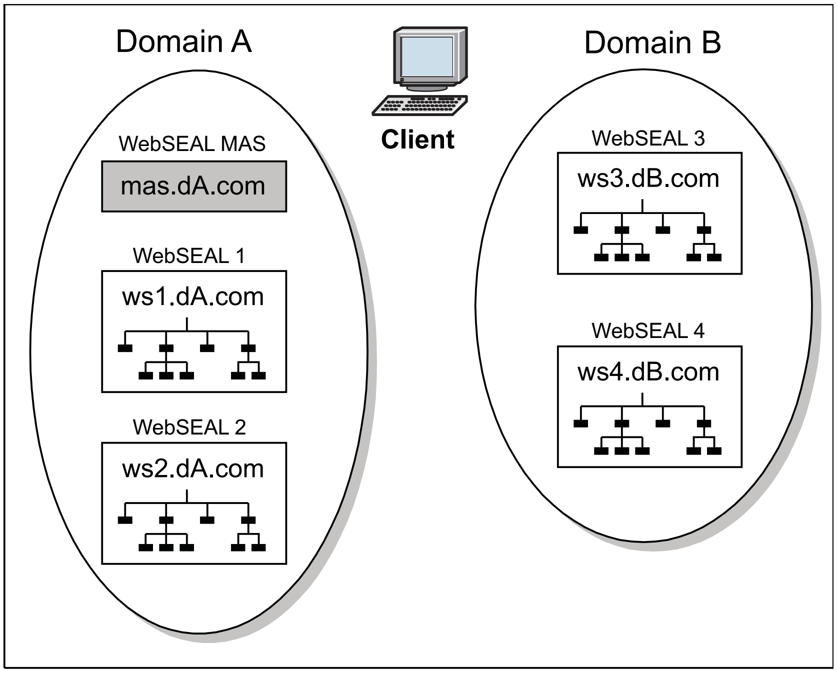


Figure 52. The e-community model

The home domain "owns" the users—that is, it controls the user's authentication information. Regardless of where a user makes a request for resources, the home domain is always where the user must authenticate.

Authentication occurs against a master authentication server (sometimes referred to as the MAS)—a server (or set of replica servers) that is located in the home domain and configured to authenticate all users. The diagram represents the MAS as mas.dA.com. The duty of the MAS should be restricted to providing authentication services. The MAS should not contain resources that are available to users.

After a user has successfully authenticated to the MAS, the MAS generates a "vouch-for" token. This token is passed back to the server where the user is making the request. The server treats this vouch-for token as proof that the user has successfully authenticated to the MAS and can participate in the e-community.

The transfer of information between e-community domains is described in detail in the section “E-community process flow.”

E-community features and requirements

- The ECSSO model supports access using direct URLs (bookmarks) to resources. This feature contrasts with the cross-domain single signon (CDSSO) model, which relies on a specially configured **pkmscdsso** link (see “Cross-domain single signon concepts” on page 548).
- All users who participate 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 if the user does not have a valid account with the MAS (for example, users who belong to domain B but do not participate in the domain A-domain B e-community, where domain A is the "home" domain).
- Unless WebSEAL is configured to handle authentication failure at 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) "vouches for" the user's authenticated identity.
- Domain-specific cookies are used to identify the server that can provide vouch-for services. Domain cookies allow servers in a remote domain to request vouch-for information locally. The encrypted contents of e-community cookies do not contain user identity or security information.
- Special tokens are used to pass encrypted "vouched for" user identity. The vouch-for token does not contain actual user authentication information. Integrity is provided by shared secret key (triple-DES). The token contains a timeout (lifetime) value to limit the duration of the token validity.
- WebSEAL provides a configuration option that, when enabled, permits only the MAS to generate "vouch-for" tokens.

E-community process flow

An e-community consists of a master authentication WebSEAL server (MAS) and additional WebSEAL servers located in the home domain and remote domains. The MAS can exist as a single instance of a WebSEAL server, or a set of WebSEAL replicas located behind a load balancer (where the load balancer is identified as the MAS).

All participating local and remote WebSEAL servers need to be configured to use the home domain MAS for initial client authentication. This configuration is a hard requirement for servers in the home domain, and a soft requirement for servers in remote domains. For example, some servers in remote domains can be configured to handle their own authentication. These servers, and the resources they protect, can operate independently of the e-community, even if they are located in a participating e-community domain.

The e-community implementation is based on a vouch-for system. Typically, when a user requests a resource from a WebSEAL server where the user has not established a valid session, WebSEAL prompts the user for authentication information. In an e-community configuration, the WebSEAL server identifies a vouch-for server and requests verification from this vouch-for server that the user has authenticated.

The vouch-for server has valid credential information for that user. For the user's first request, the vouch-for server is always the MAS. The MAS continues to serve as the vouch-for server for resources located in the home domain. As the user continues with resource requests across the e-community, an individual server in each remote domain can build its own credential for the user (based on user identity information from the MAS) and assume the role of vouch-for server for resources in its domain.

The verification requested of the vouch-for server takes the form of a vouch-for token. The vouch-for server creates the token and returns it to the requesting WebSEAL server. The user identity information in the token is encrypted. The token contains a lifetime limit.

Upon receipt of the vouch-for token, the requesting server builds credentials and a local session for that user. The user can now access the requested resource based on normal authorization controls. The user benefits from not having to log in again—a goal of the e-community model.

Refer to the following diagram as you follow the e-community process flow in the remainder of this section. The process flow describes two possible FIRST access scenarios (1 and 2). These are followed by two possible NEXT access scenarios (3 and 4) which follow immediately after 2 or 3. Scenario 5 occurs any time after the initial access.

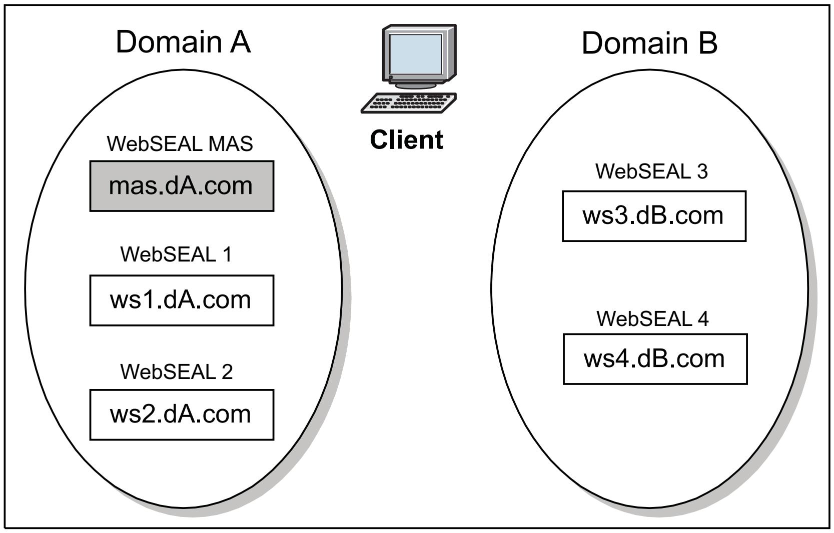


Figure 53. Example configuration for e-community process flow

"Vouch-for" Servers

- The MAS is always used to authenticate a user accessing any part of the e-community for the first time.

  The MAS should perform only as an authentication server and not as a resource provider. The MAS should not be configured to act as a master authentication server and, simultaneously, to protect resources. This recommendation addresses performance concerns and is not a security requirement.

- The MAS is always the vouch-for server for the home domain (domain A in this example).
- A domain-specific e-community cookie is used to identify the vouch-for server for all other servers within a given domain. The vouch-for server is the first server in a domain that requests a vouch-for token from the MAS. The vouch-for server provides vouch-for information for the user within the domain. Subsequent requests for vouch-for services in a given remote domain can be made locally by this server, rather than accessing the out-of-domain MAS. In the home domain, the e-community cookie identifies the MAS as the vouch-for server.

(1) FIRST e-Community Local (Domain A) Access: WebSEAL 1

1. User requests a resource protected by WebSEAL 1 (within the same domain as MAS). The browser contains no e-community cookie for this domain. WebSEAL 1 has no cached credentials for the user.
2. WebSEAL 1 configuration has e-community authentication enabled and specifies the location of the MAS. WebSEAL 1 redirects the browser to a special vouch-for URL on the MAS.
3. The MAS receives the vouch-for request and, failing to find credentials for that user, prompts the user to login.
4. After successful login, the MAS builds a credential for the user, stores it in the cache, and redirects the browser back to the originally requested URL on WebSEAL 1 with an encrypted vouch-for token. In addition, a domain A-specific e-community cookie is placed on the browser to identify the vouch-for server for this domain (in this case, the MAS).
5. The token create module can obtain extended user attribute information that can be used by the destination server during the user mapping process.

   Attribute information can come from two sources. First, the **[ecsso-token-attributes]** stanza of the WebSEAL configuration file is checked for configured stanza entries. Secondly, the CDMF library is called (**cdmf_get_usr_attributes**) to obtain additional attributes. Attributes from the CDMF library override any settings in the **[ecsso-token-attributes]** stanza.
6. If the login attempt is unsuccessful, the MAS returns a vouch-for token that indicates a failure status. This token is constructed to be indistinguishable from a success status vouch-for token. The requesting server reacts to a failure status token as if the user had failed local authentication.
7. WebSEAL 1 decrypts the token.

   Note: Identity mapping should not be required within the same domain. If identity mapping is required, WebSEAL 1 can use the cross-domain mapping framework (CDMF).
8. If an attribute list for the new identity is constructed, the token consume module first processes the attributes according to the settings in the **[ecsso-incoming-attributes]** stanza of the WebSEAL configuration file. Then the module calls to the CDMF library, which performs the actual user mapping (**cdmf_map_usr**).
9. The CDMF library passes the user's mapped identity, and any extended attribute information, back to the token consume module. The token consume module passes the identity to the WebSEAL server, which builds a credential.

10. The authorization service permits or denies the request based on the user's credential and the specific ACL and POP permissions associated with the requested resource.

(2) FIRST e-Community Remote (Domain B) Access: WebSEAL 3

1. User requests a resource protected by WebSEAL 3 (remote domain B). The browser contains no e-community cookie for this domain. WebSEAL 3 has no cached credentials for the user.
2. WebSEAL 3 configuration has e-community authentication enabled and specifies the location of the MAS. WebSEAL 3 redirects the browser to a special vouch-for URL on the MAS.
3. The MAS receives the vouch-for request and, failing to find credentials for that user, prompts the user to login.
4. After successful login, the MAS builds a credential for the user, stores it in the cache, and redirects the browser back to the originally requested URL on WebSEAL 3 with an encrypted vouch-for token. In addition, a domain A-specific e-community cookie is placed on the browser to identify the vouch-for server for this domain (in this case, the MAS).
5. The token create module can obtain extended user attribute information that can be used by the destination server during the user mapping process.

   Attribute information can come from two sources. First, the **[ecsso-token-attributes]** stanza of the WebSEAL configuration file is checked for configured stanza entries. Secondly, the CDMF library is called (**cdmf_get_usr_attributes**) to obtain additional attributes. Attributes from the CDMF library override any settings in the **[ecsso-token-attributes]** stanza.
6. If the login attempt is unsuccessful, the MAS returns a vouch-for token that indicates a failure status. This token is constructed to be indistinguishable from a success status vouch-for token. If the user fails authentication at the MAS, then the user is prompted for a local authentication at WebSEAL 3. If the user's account exists on this server, authentication then succeeds.
7. WebSEAL 3 decrypts the token.
8. If an attribute list for the new identity is constructed, the token consume module first processes the attributes according to the settings in the **[ecsso-incoming-attributes]** stanza of the WebSEAL configuration file. Then the module calls to the CDMF library, which performs the actual user mapping (**cdmf_map_usr**).
9. The CDMF library passes the user's mapped identity, and any extended attribute information, back to the token consume module. The token consume module passes the identity to the WebSEAL server, which builds a credential for the user.
10. WebSEAL 3 creates and sets a second e-community cookie (valid for domain B) on the browser, identifying WebSEAL 3 as the vouch-for server for domain B.
11. The authorization service permits or denies the request.

(3) NEXT e-Community Local (Domain A) Access: WebSEAL 2

1. User requests a resource protected by WebSEAL 2 (within the same domain as MAS). The browser contains a domain A e-community cookie identifying the MAS as the vouch-for server. WebSEAL 2 receives this cookie. WebSEAL 2 has no cached credentials for the user.
2. WebSEAL 2 configuration has e-community authentication enabled and specifies the location of the MAS. The presence of the domain A e-community

cookie overrides the WebSEAL 2 configuration for the MAS location. The cookie provides WebSEAL 2 with the identity of the vouch-for server. (If scenario 2 occurred first, there would also be a domain B cookie maintained on the browser that would not be sent to a domain A server.)

3. WebSEAL 2 redirects the browser to a special vouch-for URL on the domain A vouch-for server identified in the cookie (in this case the MAS, because WebSEAL 2 is in domain A).
4. The MAS receives the vouch-for request and finds credentials for that user in the cache (this occurred in scenario 1 and 2).
5. The MAS redirects the browser back to the originally requested URL on WebSEAL 2 with an encrypted vouch-for token. (See scenario 1 and 2 for a description of extended attribute handling.)
6. WebSEAL 2 decrypts the token and builds its own credential for the user.
7. The authorization service permits or denies the request.

(4) NEXT e-Community Remote (Domain B) Access: WebSEAL 4

1. User requests a resource protected by WebSEAL 4 (remote domain B). If scenario 2 occurred first, the browser contains a domain B e-community cookie identifying WebSEAL 3 as the vouch-for server. WebSEAL 4 has no cached credentials for the user.
2. WebSEAL 4 configuration has e-community authentication enabled and specifies the location of the MAS. The presence of a domain B e-community cookie overrides the WebSEAL 4 configuration for the MAS location. The cookie provides WebSEAL 4 with the identity of the vouch-for server. (If scenario 1 occurred first, there would only be a domain A cookie maintained on the browser that would not be sent to a domain B server. The configured MAS location would be used instead. WebSEAL 4 would then become the vouch-for server for domain B.)
3. If scenario 2 occurred first, WebSEAL 4 redirects the browser to a special vouch-for URL on the domain B vouch-for server identified in the domain B cookie (in this case WebSEAL 3).
4. WebSEAL 3 receives the vouch-for request and finds credentials for that user in the cache (this occurred in scenario 2).
5. WebSEAL 3 redirects the browser back to the originally requested URL on WebSEAL 4 with an encrypted vouch-for token. (See scenario 1 and 2 for a description of extended attribute handling.)
6. WebSEAL 4 decrypts the token and builds its own credential for the user.
7. The authorization service permits or denies the request.

(5) ANOTHER e-Community Local (Domain A) Access: WebSEAL 2

1. User connects to WebSEAL 2 (domain A) with a request. If scenario 3 occurred, WebSEAL 2 has cached credentials for the user.
2. The authorization service permits or denies the request.

Logout from the e-Community

- If the user logs out by closing the browser, all SSL sessions and all e-community cookies are cleared.
- If the user logs out using the **/pkmslogout** page, the SSL session and e-community cookie for that domain are cleared.

The e-community cookie

- The e-community cookie is a domain-specific cookie set by one WebSEAL server, is stored in the memory of the user's browser, and is transmitted to other WebSEAL servers (in the same domain) in subsequent requests.
- The domain-specific cookie contains the name of the vouch-for server, the e-community identity, a location (URL) of the vouch-for server and functionality, and a lifetime (timeout) value. The cookie contains no user or security information.
- The e-community cookie allows servers in participating domains to request vouch-for information locally. The e-community cookie for the domain where the MAS is located plays a less significant role.
- The cookie has a lifetime value that is set in the WebSEAL configuration file. This lifetime value specifies how long a remote server is able to provide vouch-for information for the user. When the cookie lifetime has expired, the user must be redirected to the MAS for authentication.
- If the value of the **disable-ec-cookie** option in the **[e-community-sso]** stanza is **yes**, then the MAS is the only server permitted to generate vouch-for tokens.
- 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.

The vouch-for request:

The vouch-for request is triggered when a user requests a resource from a target server (configured for e-community) that contains no credential information for that user.

The server sends a redirect to the vouch-for server, either the MAS or a delegated vouch-for server identified in an e-community cookie.

The vouch-for request contains the following information:

```
https://vouch-for-server/pkmsvouchfor?ecommunity-name&target-URL
```

The receiving server checks the *ecommunity-name* to validate the e-community identity. The receiving server uses the *target-URL* in the vouch-for reply to redirect the browser back to the originally requested page.

The **pkmsvouchfor** vouch-for URL is configurable.

For example:

```
https://mas.dA.com/pkmsvouchfor?companyABC&https://ws5.dB.com/index.html
```

Note: The `pkmsvouchfor` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

The vouch-for reply:

The vouch-for reply is the response from the vouch-for server to the target server.

The vouch-for server is either the MAS or a delegated vouch-for server in a domain remote that is from the MAS domain.

The vouch-for reply contains the following information:

```
https://target-URL?PD-VFHOST=vouch-for-server&PD-VF=encrypted-token
```

The PD-VFHOST label identifies the server that performed the vouch-for operation. The receiving (target) server uses this information to select the correct key required to decrypt the vouch-for token. The PD-VF label identifies the encrypted token in the vouch-for reply URL.

For example:

```
https://w5.dB.com/index.html?PD-VFHOST=mas.dA.com&PD-VF=3qhe9fjkp...ge56wgb
```

The vouch-for token

In order to achieve cross-domain single signon, some user identity information must be transmitted between servers. This sensitive information is handled using a redirect that includes the identity information encrypted as part of the URL. This encrypted data is called a **vouch-for token**.

- The token contains the vouch-for success or failure status, the user's identity (if successful), the fully qualified name of the server that created the token, the e-community identity, and a creation time value.
- The holder of a valid vouch-for token can use this token to establish a session (and set of credentials) with a server without explicitly authenticating to that server.
- The token is encrypted using a shared triple-DES secret key so that its authenticity can be verified.
- Encrypted token information is not stored on the browser.
- The token is passed only once. The receiving server uses this information to build user credentials in its own cache. The server uses these credentials for future requests by that user during the same session.
- The token has a lifetime (timeout) value that is set in the WebSEAL configuration file. This value can be very short (seconds) to reduce the risk of a replay attack.

Configuration of e-community single sign-on

This section contains the following introductory topics:

The main configuration tasks are described in the following topics:

The remaining topics provide supporting information:

E-community configuration summary

An e-community is configured under the following conditions and guidelines:

- The vouch-for server (the MAS or a delegated vouch-for server) always has the token create responsibility.
- The receiving server (where the requested resource is located) always has the token consume responsibility.
- A delegated vouch-for server (for all domains remote from the MAS domain) must have both token create and token consume capabilities.

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division:

Configuring token create functionality on the vouch-for server:

About this task

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division.

Procedure

1. Enable e-community authentication to process single signon requests by communication type (**e-community-sso-auth**).
2. Specify the unifying name of the e-community for all participating servers (**e-community-name**).
3. Configure the built-in single signon token create module (**sso-create**).
4. Create the key file used to encode and decode the vouch-for token. Copy the key file to all appropriate participating servers (**[e-community-domain-keys]** stanza).
5. Configure the token label used in the vouch-for reply (**vf-argument**).
6. Specify if this server is the MAS or not the MAS (**is-master-authn-server**).
7. Specify the vouch-for URL used in the vouch-for request (**vf-url**).
8. Configure token and e-community cookie lifetime values (**vf-token-lifetime** and **ec-cookie-lifetime**).

Configuring token consume functionality on the receiving server:

About this task

The following configuration steps are explained in detail in the remaining sections of this e-community chapter division.

Procedure

1. Enable e-community authentication to process single signon requests by communication type (**e-community-sso-auth**).
2. Specify the unifying name of the e-community for all participating servers (**e-community-name**).
3. Configure the built-in single signon token consume module (**sso-consume**).
4. Assign the appropriate key file (**[e-community-domain-keys]** stanza).
5. Configure the token label used in the vouch-for reply (**vf-argument**).
6. Specify that this server is not the MAS (**is-master-authn-server**).
7. Specify the vouch-for URL used in the vouch-for request (**vf-url**).
8. Configure token and e-community cookie lifetime values (**vf-token-lifetime** and **ec-cookie-lifetime**).

E-community conditions and requirements

- The e-community implementation requires a consistent configuration across all WebSEAL servers in all domains participating in the e-community.
- For e-community to function successfully, each participating WebSEAL server must reveal its fully qualified host name to the other participating servers in the cross-domain environment. If any host name does not include a domain, e-community cannot be enabled and an error message is logged in

msg_webseald.log. When setting up an e-community environment, ensure that the machine-specific networking setup for each participating server is configured to identify the server with a fully qualified host name.

- All WebSEAL servers participating in e-community must have machine times synchronized. Authentication between servers can fail when machine time differences are too great.
- The e-community implementation is supported on both HTTP and HTTPS.
- Individual e-community domains manage their own user identities and associated privileges. You can use the Cross-domain Mapping Function (CDMF) API to map a user from a remote domain to a valid user in the local domain.

  If the e-community domains share global user identities, those users could be distinguished by different passwords in the different domains. For example, a user "abc" can exist in both domain A and domain B, using different passwords for each domain.
- Configuration for e-community is set in the WebSEAL configuration file of each participating WebSEAL server.
- If the originally requested URL is not redirected back to the browser from the MAS (or vouch-for server), there could be a problem with page caching if the browser is Microsoft Internet Explorer. If this is the case, configure the browser to always check for newer versions of stored pages:

  ```
  Tools > Internet Options > General > Temporary Internet Files > Settings
  ```
- Do not configure the MAS server on the same interface (IP address) of another participating WebSEAL instance.
- Because some WebSEAL configuration requires machine host names to be described as fully qualified host names, you must ensure that your system and network can resolve machine names into fully qualified host names. For example, using fully qualified host names allows for many host names (IP addresses) per machine, as in the case of multiple WebSEAL instances.

Resolving machine names in an e-community environment:

E-community can be disabled upon WebSEAL startup because the machine itself is not adequately configured to resolve machine names.

About this task

The machine on which WebSEAL is located needs to be able to fully resolve an IP address. Because this functionality is very operating system-specific, it is not the role of this document to provide instructions. Always consult your system administrator if you are not sure your system has the proper capabilities.

Enabling and disabling e-community authentication

About this task

The **e-community-sso-auth** stanza entry, located in the **[e-community-sso]** stanza of the WebSEAL configuration file, enables and disables the e-community authentication method, and processes single signon requests by communication type.

Procedure

- To enable the e-community authentication method, enter "http", "https", or "both".

The values "http", "https", and "both" specify the type of communication used by e-community participants.

- To disable the e-community authentication method, enter "none".

  The value "none" disables e-community for that server. The default setting is "none".

  For example:

  ```
  [e-community-sso]
  e-community-sso-auth = https
  ```

  Note: You must stop and restart the WebSEAL server in order to activate changes to the WebSEAL configuration file. Complete all of the applicable configuration steps in this section and then restart WebSEAL.

Specifying an e-community name

About this task

The **e-community-name** stanza entry identifies the unifying name of the e-community for all participating servers in all participating domains. For example:

```
[e-community-sso]
e-community-name = companyABC
```

The **e-community-name** value must be the same for all WebSEAL servers in all domains that are participating in the e-community.

Encrypting the vouch-for token

About this task

Note: The 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 name of the key files used to secure tokens sent between servers participating in the e-community is specified in the **`[e-community-domain-keys]`** stanza.

```
[e-community-domain-keys]
domain-name = keyfile-name
domain-name = keyfile-name
```

E-community domain keys:

The key files required for encrypting and decrypting the tokens exchanged among the servers participating in the e-community is specified in the **`[e-community-domain-keys]`** stanza.

You must specify fully qualified domain name for each server and the name of the associated key file.

The following example provides the MAS (domain A) with key files for communicating with two remote domains (**dB** and **dC**) and a key for communicating with other servers in domain A:

```
[e-community-domain-keys]
dA.com = key.fileA-A
dB.com = key.fileA-B
dC.com = key.fileA-C
```

In this example, key.fileA-A identifies the key file used between all of the servers in domainA.

key.fileA-B identifies the key file used between domain A and domain B.

key.fileA-C identifies the key file used between domain A and domain C.

Each remote server needs to have a copy of the appropriate key file used by the MAS. To exchange tokens with the MAS (domain A), all servers in domain B require copies of key.fileA-B:

```
[e-community-domain-keys]
dA.com = key.fileA-B
```

To exchange tokens with the MAS (domain A), all servers in domain C require copies of key.fileA-C:

```
[e-community-domain-keys]
dA.com = key.fileA-C
```

Any servers in domain A which use authentication services provided by the MAS must have a copy of key.fileA-A:

```
[e-community-domain-keys]
dA.com = key.fileA-A
```

In this example, key.fileB-B identifies the key file used between all of the servers in domainB. Also, key.fileC-C identifies the key file used between all of the servers in domainC

```
[e-community-domain-keys]
dB.com = key.fileB-B
dC.com = key.fileC-C
```

Configuring the vouch-for token label name

About this task

The authentication information used for a single signon transaction is placed in the redirected request as an encrypted token query string argument to the request. This token string requires a name, or label, to identify it. The label name uniquely identifies the request to the receiving WebSEAL server as a single signon request to be handled by the single signon token consume module.

You must configure this token label on both WebSEAL servers participating in the single signon functionality.

For more information, see the Web Reverse Proxy Stanza Reference topics..

Procedure

To configure the token label, use the **vf-argument** stanza entry located in the **[e-community-sso]** stanza of the WebSEAL configuration file. For example (default):

```
[e-community-sso]
vf-argument = PD-VF
```

Specifying the master authentication server (MAS)

About this task

You must specify which server machine in the e-community is to function as the master authentication server (MAS). You must also specify if a server machine is not the MAS.

is-master-authn-server

Use the **is-master-authn-server** stanza entry to specify whether a server is the MAS or not. Values include "yes" or "no".

For example:

```
[e-community-sso]
is-master-authn-server = yes
```

Multiple WebSEALs can be configured to act as master authentication servers and then placed behind a load balancer. In this scenario, the load balancer is "recognized" as the MAS by all other WebSEAL servers in the e-community.

If the server you are configuring is not the MAS, use the **master-authn-server** to specify to this server the location of the MAS.

master-authn-server

If the **is-master-authn-server** stanza entry is set to "no", this stanza entry must be uncommented and specified. The stanza entry identifies the fully qualified domain name of the MAS.

For example:

```
[e-community-sso]
master-authn-server = mas.dA.com
```

Additionally, you can specify the HTTP and HTTPS listening ports used by the MAS if these port values are other than the default (port 80 for HTTP and port 443 for HTTPS).

master-http-port

If **e-community-sso-auth** enables HTTP e-community authentication and the master authentication server listens for HTTP requests on a port other than the standard HTTP port (port 80), the **master-http-port** stanza entry identifies the non-standard port. This stanza entry is ignored if this server is the master authentication server. By default, this stanza entry is disabled.

```
[e-community-sso]
master-http-port = port-number
```

master-https-port

If **e-community-sso-auth** enables HTTPS e-community authentication and the master authentication server listens for HTTPS requests on a port other than the standard HTTPS port (port 443), the **master-http-port** stanza entry identifies the non-standard port. This stanza entry is ignored if this server is the master authentication server. By default, this stanza entry is disabled.

```
[e-community-sso]
master-https-port = port-number
```

Specifying the vouch-for URL

About this task

vf-url

The **vf-url** stanza entry specifies the vouch-for URL. The value must begin with a forward-slash (/). The default value is **/pkmsvouchfor**.

For example:

```
[e-community-sso]
vf-url = /pkmsvouchfor
```

You can also express an extended URL:

```
vf-url = /ecommA/pkmsvouchfor
```

The extended URL is used when the client is communicating with a MAS that is not a WebSEAL server. This use of **vf-url** enables the client to specify access to a MAS with a specialized authentication library, such as a customized token consumption module.

Note: The `pkmsvouchfor` management page is a management command to the WebSEAL server. It is not represented in the object space and you cannot attach policies to it.

Configure token and ec-cookie lifetime values

About this task

vf-token-lifetime

The **vf-token-lifetime** stanza entry sets the lifetime timeout value (in seconds) of the vouch-for token. This value is checked against the creation time stamped on the cookie. The default value is 180 seconds. You must take into account clock skew among participating servers.

For example:

```
[e-community-sso]
vf-token-lifetime = 180
```

ec-cookie-lifetime

The **ec-cookie-lifetime** stanza entry specifies the maximum lifetime (in minutes) of the e-community domain cookie. The default value is 300 minutes.

For example:

```
[e-community-sso]
ec-cookie-lifetime = 300
```

You must take into account any clock skew among the participating domains. Clock skew means that the system times differ on the relevant servers in each domain. When this difference approaches the value of **vf-token-lifetime**, the effective lifetime of the token is greatly reduced. When this difference exceeds the value of **vf-token-lifetime**, tokens from one domain cannot be valid for the other domain. Administrators should adjust **vf-token-lifetime** accordingly. However, when clock skew requires that **vf-token-lifetime** be set to a large value, the risk of replay attacks increases. In this case, administrators should consider synchronizing the system time on the relevant servers in each domain.

For more information, see the Web Reverse Proxy Stanza Reference topics..

Handling errors from CDMF during token creation

About this task

During the creation of a e-community token, the **ssocreate** module calls the CDMF library to acquire any extended attributes to be included in the token. Extended attributes (that further describe the user) can be required for successful identity mapping of the user on the destination server. The CDMF API uses the **cdmf_get_usr_attributes** call to acquire extended attributes.

It is possible for the **cdmf_get_usr_attributes** call to fail in obtaining the required information and return an error. In such a case, the subsequent behavior of the token creation process can be controlled through use of the **propagate-cdmf-errors** stanza entry located in the **[cdsso]** stanza. Values for this stanza entry include "yes" and "no".

A "no" value (default) allows the token creation process to proceed even when CDMF fails to obtain attributes and returns an error.

A "yes" value forces the token creation process to end when CDMF fails to obtain attributes and returns an error.

Example:

```
[cdsso]
propagate-cdmf-errors = no
```

Enabling unauthenticated access

About this task

You can control whether unauthenticated users are allowed access to unprotected resources on e-community SSO participating servers. When authenticated users are allowed this access, the participating server can serve the resource without requiring that the user authenticate through the master authentication server. When this policy is configured, the participating server redirects to the master authentication server only when the client requests access to a protected resource.

This policy is set in the WebSEAL configuration file:

```
[e-community-sso]
ecsso-allow-unauth = {yes|no}
```

When **ecsso-allow-unauth** is set to "yes", unauthenticated access is enabled. The default setting is "yes".

When **ecsso-allow-unauth** is set to "no", unauthenticated access is disabled. In this case, clients must authenticate through the master authentication server when requesting access to a resource (protected or not protected) on an e-community SSO participating server.

Note: The default behavior changed for WebSEAL Version 5.1. In prior versions, unauthenticated access was disabled. To retain backwards compatible behavior with older versions of WebSEAL, set `ecsso-allow-unauth = no`.

Limiting the ability to generate vouch-for tokens

About this task

The WebSEAL configuration file includes an optional parameter that enables you to limit the ability to generate vouch-for tokens to the MAS. The **disable-ec-cookie** option in the **e-community-sso** stanza is set to no by default. Changing the value of this option to **yes** disables the use of the e-community cookie and permits only the MAS to generate vouch-for tokens. In this case, the single-signon process always uses the MAS, allowing the MAS to detect all hosts that sign on across the e-community. This option is useful for customers who want to construct a customized ECSSO signoff process. For more information on a customized single signoff process, see "Logout using pkmslogout-nomas."

Configuration of the behavior for authentication failure

When an unauthenticated user has an unsuccessful login at the MAS (for example, provides an incorrect password), the MAS generates a vouch-for token containing the error and, in the default configuration, redirects the Web browser back to the requesting host. When the requesting host encounters the error in the vouch-for token, it typically requests a local login. The **handle-auth-failure-at-mas** option in the **e-community-sso** stanza of the WebSEAL configuration file allows administrators to configure the behavior for authentication failure. If **handle-auth-failure-at-mas** is set to **yes** , the MAS handles login failures directly without redirecting the Web browser back to the requesting host; in this case, the MAS does not generate a vouch-for token until a successful authentication occurs.

Logout using pkmslogout-nomas

To help customers who wish to construct and implement a single signoff process in which all hosts are signed off at the MAS, WebSEAL can be configured to use the pkmslogout-nomas management page. When ECSSO is configured, the pkmslogout-nomas management page can be used as an alternative to the **pkmslogout** command to log out from the session on the current host. For example:

```
https://www.example.com/pkmslogout-nomas
```

The /pkmslogout management page causes clients that log out of a local system to be redirected to perform another logout on the MAS. The /pkmslogout-nomas management page operates exactly like the /pkmslogout management page with the exception that it does not redirect the Web browser to the MAS host /pkmslogout management page to log out the user's session. This makes it possible to chain logouts from the MAS. For example, a custom logout page can be placed on the MAS system that contains hidden links to the /pkmslogout pages on all the non-MAS systems, thus logging the user out of every system in the community.

Use of e-community with virtual hosts

See "E-community single signon with virtual hosts" on page 494.

Extended attributes for ECSSO

This section contains the following topics:

Extended attributes to add to token

In the WebSEAL configuration file, you can specify extended attributes from a user credential to add to the cross-domain single signon token. Extended attributes consist of information about a user identity that is added to an extended attribute list when a user credential is created. Extended attributes can be added by a number of authentication mechanisms, including external authentication C API services. The external authentication C API modules can be used, for example, to obtain user information from a registry that is external to Security Access Manager.

You can use this setting to customize the contents of the e-community single signon token. This feature enables you to tailor the token contents to match the needs of the destination domain. When you use this feature to add an attribute to a token, you must also configure the WebSEAL configuration file for the server in the destination domain. For the destination server, the stanza **[ecsso-incoming-attributes]** is used to specify the handling (extract or ignore) of each attribute.

You can specify extended attributes by name, or you can declare a pattern that matches multiple attribute names. You can use standard Security Access Manager wildcard-matching characters. For a list of supported wildcard pattern matching characters, see “Supported wildcard pattern matching characters” on page 57.

Each entry is assigned the name of the domain for which the token is intended. You can include multiple entries specifying names or patterns for each domain.

The syntax is:

```
[ecsso-token-attributes]
domain_name = pattern1
domain_name = pattern2
...
domain_name = patternN
<default> = pattern1
<default> = pattern2
...
<default> = patternN
```

The `<default>` entry is optional. When WebSEAL does not find an entry that matches the domain name, WebSEAL looks for a `<default>` entry. If the configuration file contains a `<default>` entry, WebSEAL uses the assigned attribute patterns for the current domain. The string `<default>` is a keyword, and must be specified exactly as shown above, including the < and > characters.

Example: You are creating an e-community single signon solution between two domains: example1.com and example2.com. Users log in to example1.com but can get redirected to example2.com during the user session. Your deployment includes a customized external authentication C API module that inserts information into each user credential. The information includes a fixed name attribute "`job_category`" and a variable number of attributes, each prefixed with the characters "`my_ext_attr_`". This information needs to be added to the cross-domain token. The configuration file entries would be:

```
example2.com = job_category
example2.com = my_ext_attr_*
```

Extended attributes to extract from token

In the WebSEAL configuration file, you can specify how the token consume module handles extended attributes that have been added to an e-community single sign-on token.

The attributes can either be extracted or ignored. In some cases, you must extract the attributes because there is no way for a server in the destination domain to generate them. In other cases, you do not want to extract the tokens, because the server in the destination domain can use an independent process to gather the same extended attributes. For example, the attribute could reflect a timestamp that needs to reflect the system time on the destination server.

In the built-in token consume module, attributes that are extracted from the token are passed through to the cross-domain mapping framework library. The default cross-domain mapping framework library passes attributes directly through to the user credential. Customized cross-domain mapping framework libraries can manipulate attributes as needed before passing them to the user credential.

The syntax for the entries is as follows:

```
[ecsso-incoming-attributes]
attribute_pattern = {preserve|refresh}
```

Typically, the names of the extended attributes (*attribute_pattern*) match the names of the attributes specified in the **[ecsso-token-attributes]** stanza of the configuration file for a WebSEAL server that generates the tokens. The value must be one of the following keywords:

- preserve

  Extract all attributes that match *attribute_pattern*.
- refresh

  Do not extract attributes that match *attribute_pattern*.

Extended attributes in the token that do not match an entry in **[ecsso-incoming-attributes]** are preserved (extracted).

The order of the entries in the stanza is important. The first entry that matches an attribute name is used. Other entries are ignored. For example, if you want to extract the attribute named `my_special_attr1` but want to ignore all other entries with the prefix `my_special_attr_`, the stanza entries should be:

```
[ecsso-incoming-attributes]
my_special_attr1 = preserve
my_special_attr_* = refresh
```

Using the examples shown above in “Extended attributes to add to token” on page 578, the entries in the WebSEAL configuration file for a server that operates in the example2.com domain could be:

```
[ecsso-incoming-attributes]
job_category = preserve
my_cdas_attr_1 = preserve
my_cdas_attr_* = refresh
```

In this example, the attributes `job_category` and `my_cdas_attr_1` are extracted from the token. The remainder of the attributes with the prefix `my_cdas_attr_` are ignored.

Compatibility issues for ECSSO

This section contains the following topics:

UTF-8 encoding of tokens for e-community single signon

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

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

The default value is "yes".

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

Token compatibility with earlier versions of WebSEAL

Security Access Manager, version 8.0, changed the default cipher that WebSEAL uses to encode the tokens. Tokens that are generated with this default cipher are not compatible with the tokens that are generated by earlier versions of WebSEAL.

If you integrate e-community single sign-on with servers that include versions of Security Access Manager earlier than version 8.0, you must enable the **pre-800-compatible-tokens** stanza entry in the **[server]** stanza of the WebSEAL configuration file. For example:

```
pre-800-compatible-tokens = yes
```

The default setting for this stanza entry is no.

Single sign-off

You can configure WebSEAL to initiate single sign-off from multiple protected web resources located on junctioned backend servers.

The single signoff functionality is detailed in the following sections:

Overview of the single sign-off functionality

You can configure WebSEAL to send HTTP requests to predefined applications when a session is terminated. The applications that receive these requests can then terminate any associated sessions that are located on junctioned backend servers.

When a session is ended, WebSEAL deletes the session and the session data that it manages. WebSEAL cannot control sessions created and managed by backend applications. This situation results in backend server sessions remaining active after the corresponding WebSEAL session is terminated. WebSEAL provides a mechanism to remove sessions on backend servers when a session ends in WebSEAL.

To achieve single signoff, WebSEAL sends a request to configured single signoff URIs whenever a WebSEAL session is destroyed. Using the information provided in the request, applications on the backend servers can terminate the stale sessions.

There are four different mechanisms that can terminate a WebSEAL session:

- User request by accessing **pkmslogout**.
- Session timeout.
- EAI session termination command.
- Session terminate command from the **pdadmin** tool.

Note: Using this feature in a distributed session cache environment generates a separate signoff request from each WebSEAL server containing the terminated session. Therefore, the single signoff application in a distributed session cache environment must handle multiple signoff requests for a single session - one per WebSEAL server.

Configuring single signoff

About this task

You can enable single signoff in WebSEAL by specifying the URIs that receive the single signoff request. Configure a **single-signoff-uri** entry in the **[acnt-mgt]** stanza to reference each single signoff application. This resource cannot be located on a virtual host junction, and you must provide the server relative URI. For example:

```
[acnt-mgt]
single-signoff-uri = /applications/signoff
```

Each time a WebSEAL session is terminated, WebSEAL sends a request to each of the specified URIs. Each request contains the configured headers and cookies for the junction of the specified resource. The single signoff resources are responsible for using this information to terminate any sessions on the back-end servers.

WebSEAL expects to receive a response containing an HTTP status code of **200 OK**. If the response contains any other status code, WebSEAL logs an error.

Note: You can perform single signoff on multiple junctioned servers. Configure more than one **single-sign-off-uri** entry to send a request to multiple URIs.

With `single-signoff-uri` configured, WebSEAL does not send cookies that were sent by the browser to the backend `single-signoff-uri`. The WebSEAL `single-signoff-uri` mechanism by design, does not use cookies sent by the client. The design of the mechanism is to only use cookies stored in the WebSEAL cookie jar (managed cookies). This way WebSEAL is able to perform the single sign-off even in cases like a timeout, where there is no logout request from the browser.

Specifications for single sign-off requests and responses

When you configure the single signoff functionality, the single signoff requests and responses are formatted according to these specifications.

Single sign-off requests

WebSEAL sends the request to the single signoff resource that is specified by the **single-signoff-uri** configuration entry. The single sign-off request contains:

- The HTTP GET method.
- Any cookies and headers configured for the junction point where the single signoff resource resides.

Single sign-off responses

WebSEAL expects a response with an HTTP status code **200 OK**. Any other status code results in a logged error. WebSEAL disregards the body and any other headers in the response.

Chapter 10. Deployment

Deployment planning

Before you implement a particular Security Access Manager solution, you must determine the specific security and management capabilities that are required for your network.

The first step in planning the deployment of a Security Access Manager security environment is to define the security requirements for your computing environment. Defining security requirements means determining the business policies that must apply to users, programs, and data. This definition includes:

- Objects to be secured
- Actions that are permitted on each object
- Users that are permitted to perform the actions

Enforcing a security policy requires an understanding of the flow of access requests through your network topology. In your plan, identify correct roles and locations for firewalls, routers, and subnets. Deploying a Security Access Manager security environment also requires identifying the optimal points within the network that evaluates user access requests, and grants or denies the requested access.

Implementation of a security policy requires understanding the number of users, quantity of data, and throughput that your network must accommodate. You must evaluate performance characteristics, scalability, and the need for failover capabilities.

After you have an understanding of the features that you want to deploy, you can decide which Security Access Manager systems you need in your environment.

For useful planning documentation, including actual business scenarios, see supplemental product information at the following websites:

http://www.ibm.com/redbooks/

http://www.ibm.com/software/sysmgmt/products/support/Field_Guides.html

WebSEAL instance deployment

This chapter discusses techniques for deploying one or more instances of WebSEAL.

WebSEAL instance configuration overview

This section contains the following topics:

WebSEAL instance configuration planning

A WebSEAL instance is a unique WebSEAL server process with a unique configuration file and listening port. WebSEAL deployments support multiple WebSEAL instances.

To configure a WebSEAL instance, you must collect some information about the Security Access Manager deployment.

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

- **Administrator name and password**

  The authentication details for the Security Access Manager administrative user. By default, this is the **sec_master** user. You must have administrative user permissions to configure a WebSEAL instance.
- **Domain**

  The Security Access Manager domain.
- **Host name**

  The host name that the Security Access Manager policy server uses to contact the appliance. The address that corresponds to this host name must match a management interface address of the appliance. Valid values include any valid host name or IP address.
- **Instance name**

  A unique name that identifies the WebSEAL instance. Multiple WebSEAL instances can be installed on one appliance. Each instance must have a unique name.

  Valid characters for instance names include the alphanumeric characters ([A-Z][a-z][0–9]) plus the following characters: underscore (_), hyphen (-), and period (.). No other characters are valid.

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

  The instance name also affects how the full server name is listed during a **`pdadmin server list`** command. For this command, the full server name has the following format:

  instance_name`-webseald-`*host_name*

  For example, an *instance_name* of `web1` installed on a host named `diamond` has the following full server name:

  `web1-webseald-diamond`
- **Listening port**

  This is the port through which the WebSEAL instance communicates with the Security Access Manager policy server. The default port number is 7234. This port number must be unique for every WebSEAL instance.

  The default port is typically used by the default (first) WebSEAL instance. The installation automatically increments to the next available port. You can modify the port number if necessary.

  Any port number above 1024 is valid. Select a port that is not used for any other purpose.
- **IP address for the primary interface**

  The unique IP address for the WebSEAL instance. The WebSEAL server listens on this IP address for incoming requests. You must also assign each WebSEAL instance a unique HTTP and HTTPS port.
- **HTTP protocol and HTTP port**

  Specifies whether to accept user requests across the HTTP protocol. If you enable HTTP, you must assign a port number. The default port number is 80. This port is used by the default (first) instance. If this port is not available, the installation automatically increments to the next available port.
- **HTTPS protocol and HTTPS port**

Specifies whether to accept user requests across the HTTPS protocol. If you enable HTTPS, you must assign a port number. The default port number is 443. This port is used by the default (first) instance. If this port is not available, the installation automatically increments to the next available port.

- **User registry - SSL communication**

  WebSEAL communicates with the LDAP server during authentication procedures. Use of SSL during communication with the LDAP server is optional. However, use of SSL is highly recommended for security reasons in all production deployments. Disabling of SSL usage can be considered for temporary testing or prototyping environments.

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

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

  – **Key file name**

    The file that contains the LDAP SSL certificate.

  – **Certificate label**

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

  – **Port**

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

Example WebSEAL instance configuration values

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

Setting	Value
Security Access Manager Administrator name	sec_master
Security Access Manager Administrator password	mypassw0rd
Security Access Manager domain	domainA
Host name	diamond.subnet2.ibm.com
Instance name	web1
Listening port	7235
IP address for the Primary Interface	1.2.3.5
Enable HTTP	yes
HTTP port	81
Enable HTTPS	yes
HTTPS port	444
Enable SSL	yes

Setting	Value
Key file name	`client.kdb`
Certificate label	(none)
Port	636

Unique configuration file for each WebSEAL instance

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

The format is:

```
webseald-instance_name.conf
```

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

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

WebSEAL instance configuration tasks

Tasks:

Adding a WebSEAL instance

About this task

To add a WebSEAL instance, complete each of the following steps:

Procedure

1. Plan the configuration. Complete the following worksheet. For information on determining appropriate values for each setting, see "WebSEAL instance configuration planning" on page 583.

Table 68. Worksheet for adding a WebSEAL instance

Setting	Value
Administrator name	
Administrator password	
Host name	
Instance name	
Listening port	
Enable HTTP	yes or no
HTTP port	
Enable HTTPS	yes or no
HTTPS port	
Enable SSL	yes or no
Certificate label	
Key file name	
Port	

2. Ensure that all configured WebSEAL instances are running. This prevents any possible conflicts between servers over port usage.
3. Click **New** on the Reverse Proxy management page in the LMI to create a new WebSEAL instance.
4. Provide settings for the fields displayed on the **Instance**, **IBM Security Access Manager**, **Transport** and **User Registry** tabs.
5. Click **Finish**. A message is displayed indicated that the new instance is successfully configured.
6. Verify that the new instance is running in the LMI. The Reverse Proxy management page indicates that the state of the instance is **Started**.

Removing a WebSEAL instance

About this task

To remove the configuration for a WebSEAL instance, complete the following steps:

Procedure

1. Assemble the following information:
   - Instance name
   - Administrator ID
   - Administrator ID password
2. On the Reverse Proxy management page in the LMI, select the instance that you want to delete.
3. Click **Delete**. The Delete Reverse Proxy Instance window displays.
4. Enter the Administrator authentication details for the selected instance.
5. To delete the instance, click **Delete** in the Delete Reverse Proxy Instance window. A system notification displays to indicate that the instance was successfully deleted.

Load balancing environments

This section contains the following topics:

Replicating front-end WebSEAL servers

About this task

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

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

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

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

1. Install and configure WebSEAL on both WS1 and WS2 server machines.
2. Using the **pdadmin** command, create a new object to be the root of the authorization space for both WebSEAL servers. For example:

   ```
   pdadmin> object create /WebSEAL/newroot "Description" 5 ispolicyattachable yes
   ```
3. Stop WebSEAL on WS1.
4. On WS1, change the value of the **server-name** stanza entry in the WebSEAL configuration file from "WS1" to "newroot":

   ```
   [server]
   server-name = newroot
   ```
5. Restart WebSEAL on WS1.
6. Repeat Steps 3-5 for WS2.

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

Use the following procedure when unconfiguring either WS1 or WS2:

Procedure

1. Stop the WebSEAL server.
2. Change the value of the **server-name** stanza entry back to its original value. For example, for WS1:

   ```
   [server]
   server-name = WS1
   ```
3. Proceed with normal unconfiguration procedures.

Results

Conditions:

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

Controlling the login_success response

About this task

In a network topology involving multiple WebSEAL instances controlled by a load balancing system, the request URL can be "lost" during the communication exchange that occurs for the authentication process. For example, it is possible for one WebSEAL instance to receive the request URL (for a protected resource) and present the user with a login form. When the user submits the completed login form, a non-sticky load balancer might send the POST data to a second WebSEAL instance. (Sticky load balancing is the distribution of user requests across a set of servers in such a way that requests from a given user are consistently sent to the same server.)

This second WebSEAL instance can successfully process the login POST data, but is not able to redirect to the original URL request. In this case, the second WebSEAL instance sends the `login_success.html` page that reports the message "Your login was successful."

Note: WebSeal does not make use of the Referer header, because if the initial unauthenticated request was a POST, a redirect to the original URI would result in a GET to a resource which should receive a POST.

There are several possible solutions:

- Use a sticky load balancing system and configure an adequate sticky time (for example, 20-30 seconds) that allows only one WebSEAL instance to process the overall login exchange.
- Modify the WebSEAL configuration file to enable automatic redirection after authentication. WebSEAL can then redirect the user to a specified response page that can better handle the post-login process. With automatic redirection WebSEAL no longer uses the `login_success.html` response page. Automatic redirection involves the following stanzas and stanza entries:

  ```
  [enable-redirects]
  redirect =

  [acnt-mgt]
  login-redirect-page =
  ```

  See "Automatic redirection after authentication" on page 231 for complete information.
- Modify the `login_success.html` response page so that it redirects the user's browser back in the request history. This technique allows the second WebSEAL instance to receive and process the original request URL. For example:

  ```
  <HTML>
  <HEAD>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <TITLE>Success</TITLE>
  </HEAD>
  <BODY onLoad="history.back()">
  ...
  </BODY>
  </HTML>
  ```

Application integration

This chapter contains a variety of information pertaining to the integration of third-party application servers in a WebSEAL environment.

Topic Index:

Support for back-end server-side applications

WebSEAL supports server-side applications that run on back-end junctioned servers and require client identity information as input. Examples of server-side applications include:

- Java servlets
- Cartridges for Oracle Web Listener
- Server-side plug-ins

When you create a junction to a back-end server using the **-c** option, WebSEAL inserts Security Access Manager-specific client identity information into the HTTP headers of requests destined for that server. Security Access Manager-specific headers include:

- **iv-user**
- **iv-user-l**
- **iv-groups**
- **iv-creds**
- **iv-remote-address**
- **iv-remote-address-ipv6**

The Security Access Manager-specific HTTP header information enables applications on junctioned third-party servers to perform user-specific actions based on the client's Security Access Manager identity.

For complete information on this topic, see:

- "Client identity in HTTP headers (–c)" on page 520.
- "Client IP addresses in HTTP headers (–r)" on page 523.

Best practices for standard junction usage

This section includes best practices recommendations when using standard WebSEAL junctions in application integration.

Complete Host header information with -v

Virtual host configurations and portal applications require correct IP address information for proper socket connections, and complete server name information for accurate routing.

These special back-end application services require complete server name and port designation information in any requests from browsers. The **Host** header of a request contains this information and makes it available to the application. When using WebSEAL junctions, this information is supplied to the **Host** header through the use of the **–v** junction option.

Insufficient or missing server name and port information degrades the performance of virtual hosting and portal applications. In addition, domain cookies set by these applications might not contain sufficient information.

To provide the most complete information to the **Host** header, the "best practices" recommendation is to always use both the fully qualified domain name of the junctioned server and the connection port number in the **–v** option when creating or adding the junction.

The **–v** option uses the following syntax:

```
-v fully-qualified-host-name[:port]
```

For example:

```
-v xyz.ibm.com:7001
```

Note: The port designation should be supplied only if you are using a non-standard port number.

Standard absolute URL filtering

WebSEAL, as a front-end reverse proxy, provides security services to back-end junctioned application servers. Pages returned to the client from back-end applications most often contain URL links to resources located on the back-end junctioned server.

It is important that these links include the junction name to successfully direct the requests back to the correct locations of the resources. WebSEAL uses a set of standard rules to filter static URLs and supply this junction information. Additional configuration is required to filter URLs in scripts and dynamically generated URLs. For detailed information on URL filtering, see "Modification of URLs to junctioned resources" on page 419.

WebSEAL's ability to properly filter absolute URLs from static HTML pages requires information about the server name provided in the **–h** junction option. This option provides WebSEAL with the name of the back-end junctioned server. Arguments to this option can include:

- Fully qualified domain name of the server
- Short name of the server
- IP address of the server

WebSEAL identifies absolute URLs to filter based on its knowledge of the back-end junctioned server name. Depending on your network environment, the **–h** *short-name* configuration might not provide WebSEAL with sufficient information.

In the following example, a junction is created using the following option and argument for a back-end server, located in the **ibm.com**® network, with a short name of "xyz":

```
-h xyz
```

A link on an HTML page from this server appears as follows:

```
http://xyz.ibm.com/doc/release-notes.html
```

When this page passes to the client during a request, WebSEAL might fail to filter this URL because, based on the information provided by **–h**, it could not recognize "xyz.ibm.com" as the server name. Without the junction name in the path, a request for the release notes document fails.

To support proper filtering of static absolute URLs, the "best practices" recommendation is to always use the fully qualified domain name of the junctioned server in the **–h** option when creating or adding the junction.

Custom personalization service

This section contains the following topics:

Personalization service concepts

A Web portal, or launch page, is an integrated Web site service that dynamically produces a customized list of Web resources available to a specific user. Resources can include corporate content, support services, and learning tools. The portal output represents a personalized list of resources based on the access permissions for the particular user. The launch page displays only those resources that have the correct access permissions for that user.

You can use WebSEAL configuration options and the authorization API entitlements service to build a custom portal solution in a Security Access Manager environment.

The process flow for building a custom WebSEAL portal service includes the following tasks:

1. Secure policies are formulated and attached at the appropriate points in the protected object resource.
2. Appropriate explicit ACLs are attached to each of these resource objects.
3. The WebSEAL configuration file is edited to include the URL to the portal service, the path of the object space containing the portal resources, and the permission bit required by the user for access to these resources.
4. For each user request to the portal URL, WebSEAL uses the Authorization Entitlement Service to search this object space and produce a list of resources that meet the authorization conditions for that user.
5. WebSEAL places this information in a PD_PORTAL HTTP header that is sent to the back-end (junctioned) portal server.
6. The custom portal service (such as a CGI or servlet) located on the back-end server reads the PD_PORTAL header contents and, for example, maps the contents to descriptions and URL links that are displayed to the user on a Web page. This information represents the personalized list of resources available to the user based on access control permissions.

Configuring WebSEAL for a personalization service

Procedure

1. Create a new WebSEAL junction to the personalization service. For example (entered as one line):

   ```
   pdadmin> server task server_name create -t tcp -h portalhost.abc.com /portal-jct
   ```

2. Edit the WebSEAL configuration file to add a new **[portal-map]** stanza:

   ```
   [portal-map]
   ```

3. The entry in this stanza identifies the server-relative URL of the portal service program and the region of the object space that is searched for available protected portal resources, followed by the permission required for access. This is the list that is placed in the PD_PORTAL header.

   ```
   [portal-map]
   URL = object_space_region:permission
   ```

4. After adding the stanza and the appropriate mapping entries, WebSEAL (**webseald**) must be re-started.

Personalization service example

- Create a junction to the portal server:

  ```
  pdadmin> server task web1-webseald-cruz -t ssl -h PORTAL1 /portal
  ```

- Define the region of the WebSEAL protected object space that contains resources available to the personalization service:

  ```
  pdadmin> objectspace create /Resources "Portal Object Hierarchy" 10
  pdadmin> object create /Resources/Content "" 10 ispolicyattachable yes
  pdadmin> object create /Resources/Support "" 10 ispolicyattachable yes
  pdadmin> object create /Resources/Content/CGI "" 11 ispolicyattachable yes
  pdadmin> object create /Resources/Support/Servlet "" 11 ispolicyattachable yes
  ```

Note: The “ispolicyattachable” argument must be set to “yes” for each resource. The search mechanism selects only qualified resource objects with explicitly set ACLs.

- WebSEAL configuration file:

```
[portal-map]
/portal/servlet/PortalServlet = /Resources:r
```

- Portal URL used by the user:

```
https://WS1/portal/servlet/PortalServlet
```

User session management for back-end servers

WebSEAL can maintain session state with clients. You can configure WebSEAL to extend this session information to back-end junctioned application servers. Back-end applications can use session information to maintain session state with clients and terminate sessions.

This section contains the following topics:

User session management concepts

A client/server session is a series of related communications between a single client and a server that take place over a period of time. With an established session, the server can identify the client associated with each request, and has the ability to remember—over numerous requests—a specific client.

Without an established session, the communication between a client and a server might be renegotiated for each subsequent request. Session state information improves performance by eliminating repeated closing and re-opening of client/server sessions. The client can log in once and make numerous requests without performing a separate login for each request.

The WebSEAL server has the ability to maintain session state with clients and to additionally extend this session information to junctioned back-end application servers.

WebSEAL uses a session identification key, called the WebSEAL session ID, to maintain session state between the client and WebSEAL. The WebSEAL session ID serves as an index to the client's session data stored in the WebSEAL session cache. See “WebSEAL session cache structure” on page 281 and “Session cache configuration overview” on page 285.

A separate session identification key, called the user session ID, can be used to maintain session state between the client and a junctioned back-end application server. The user session ID uniquely identifies a specific session for an authenticated user and is stored as part of the user's credential information.

Back-end applications can use user session IDs to track user sessions and terminate sessions. See “Enabling user session ID management” on page 594.

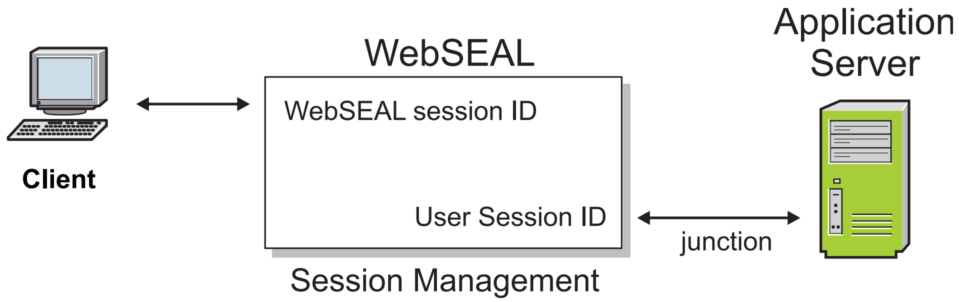


Figure 54. Session management

A single user that logs in multiple times (for example, from different machines) has multiple WebSEAL session IDs and a credential for each session. The user session ID is based on the WebSEAL session ID (there exists a one-to-one mapping between the two keys). Therefore, a user session ID exists for each WebSEAL session ID.

There are two configuration steps required to enable session management with the user session ID:

- Configure WebSEAL to store a unique user session ID for each authenticated client as an extended attribute in the credential of each client.
- Configure an extended attribute on a junction that can provide the value of this credential extended attribute (the user session ID) to a back-end application server in an HTTP header.

Enabling user session ID management

About this task

The **user-session-ids** stanza entry in the **[session]** stanza of the WebSEAL configuration file allows you to enable and disable the creation of a unique user session ID as an extended attribute in the credential of each client making a request. The default value is "no" (disabled):

```
[session]
user-session-ids = no
```

Procedure

To enable the creation of unique user session IDs, set:

```
[session]
user-session-ids = yes
```

The unique user session ID is stored in a user's credential as an extended attribute with a name and value:

```
(credential attribute)
credential attribute
credential attributetagvalue_user_session_id = user-session-id-string
```

This extended attribute name always appears with a “**tagvalue_**” prefix to prevent any conflicts with other existing information in the credential.

Inserting user session data into HTTP headers

To provide a client's user session ID to the back-end application server, configure the **HTTP-Tag-Value** extended attribute on the junction.

Setting an extended attribute on a junction:

About this task

In general, an attribute enables the junction to perform some type of operation. You use the **pdadmin object modify set attribute** command to set an attribute on a junction object in the WebSEAL protected object space:

```
pdadmin> object modify object_name set attribute attr_name attr_value
```

The HTTP-Tag-Value extended attribute for junctions:

The **HTTP-Tag-Value** extended attribute specifically instructs WebSEAL to extract a value from an extended attribute in a user credential and send that value in an HTTP header across the junction to the back-end application server with each request.

The **HTTP-Tag-Value** attribute:

- Identifies a specific extended attribute in the credential.
- Specifies the custom name of the HTTP header that will contain the value of this attribute.

The **HTTP-Tag-Value** attribute uses the following format:

```
credential_extended_attribute_name=http_header_name
```

Attribute components:

- *credential_extended_attribute_name*

  The name of the credential extended attribute that contains the user session ID is **user_session_id**. (The format used is the extended attribute name without the "tagvalue_" prefix.)

  The credential extended attribute name is not case-sensitive.
- *http_header_name*

  The *http_header_name* value specifies the custom name of the HTTP header used to deliver the user session ID across the junction.

  The example in this section uses a header called **TAM-USER-SESSION-ID**.

Setting the HTTP-Tag-Value junction attribute:

About this task

An example **pdadmin** command that sets an **HTTP-Tag-Value** junction attribute appears as follows (entered as one line):

```
pdadmin> object modify /WebSEAL/junctionA set attribute
HTTP-Tag-Value user_session_id=TAM-USER-SESSION-ID
```

Processing the HTTP-Tag-Value junction attribute:

About this task

When WebSEAL processes a client request to a back-end application server, it looks for any attributes configured on the junction object.

In this example, WebSEAL:

Procedure

1. Detects the **HTTP-Tag-Value** attribute.
2. Looks at the credential of the user making the request.
3. Extracts the user session ID value from the **tagvalue_user_session_id** extended attribute in the credential.
4. Places the value in the **TAM-USER-SESSION-ID** HTTP header as:
   ```
   TAM-USER-SESSION-ID:user-session-id-string
   ```

Results

In summary:

- Name and value of the user session ID as it appears in the user credential:
  ```
  tagvalue_user_session_id:user-session-id-string
  ```
- Example value of **HTTP-Tag-Value** attribute set on the junction object:
  ```
  user_session_id=TAM-USER-SESSION-ID
  ```
- Resulting example HTTP header name and value:
  ```
  TAM-USER-SESSION-ID:user-session-id-string
  ```

Terminating user sessions

Users can terminate their current session by using the **pkmslogout** command. Administrators, or back-end applications, can terminate user sessions using the information provided by the user session ID string. This section contains the following topics:

User session ID string format:

The format of the user session ID string consists of a MIME64-encoded WebSEAL server name and the user session ID for the client:

```
encoded-webseal-server-name_user-session-id
```

The WebSEAL server name is the value specified by the **server-name** stanza entry in the **[server]** stanza of the WebSEAL configuration file.

The *user-session-id* value is able to be directly used to provide the value of the user session ID in a **pdadmin server task terminate session** command. See “Termination of single user sessions” on page 597.

Compatibility with older user session ID format:

Beginning with WebSEAL verion 6.0, the format of the user session ID value additionally includes the name of the replica set.

The **user-session-ids-include-replica-set** stanza entry in the **[session]** stanza of the WebSEAL configuration file controls this operation. The default value is "yes". For example:

```
[session]
user-session-ids-include-replica-set = yes
```

A value of "yes" instructs WebSEAL to include the replica set name in the user session ID value. This setting allows you to use the **pdadmin server task terminate session** to terminate user sessions on virtual host junctions.

If you require user session ID compatibility with versions of WebSEAL prior to version 6.0, you can disable this function. For example:

```
[session]
user-session-ids-include-replica-set = no
```

A value of "no" instructs WebSEAL to not include the replica set name in the user session ID value.

If `user-session-ids-include-replica-set = no`, WebSEAL assumes that the **pdadmin server task terminate session** command applies to members of the replica set defined by the **[session]**, **standard-junction-replica-set** stanza entry.

The **pdadmin server task terminate session** command does not succeed for sessions initiated on virtual host junctions.

To assign standard junctions to a replica set, see “Assigning standard junctions to a replica set” on page 341.

To assign virtual hosts to a replica set, see “Virtual hosts assigned to a replica set” on page 342.

Termination of single user sessions:

An administrator or a back-end application can use the Security Access Manager administration API to terminate a specific user session based on the user session ID.

See “User session ID string format” on page 596 to review the structure of the user session ID string.

Note: The **pdadmin server task terminate session** command is supported in a distributed session cache environment.

The *user_session_id* portion of the user session ID string can be passed to the **ivadmin_server_performtask()** function. This function takes an input command string from the standard **pdadmin server task terminate session** command. For example:

```
pdadmin> server task instance-webseald-host terminate session user_session_id
```

The WebSEAL instance name can be obtained from the HTTP **iv_server_name** header passed in every request.

Note: Although you can manually perform this **pdadmin** operation, the long value of the *user_session_id* can make this task cumbersome.

WebSEAL verifies that the back-end server initiating the terminate operation has appropriate permissions before terminating the user's session. WebSEAL then removes the corresponding session cache entry so that the session is terminated.

It is important to consider the conditions under which this command might be used. If the intent is to make sure that a user is removed from the secure domain entirely, the termination of a single user is only effective when, in addition, the account for that user is also made not valid (removed).

Certain authentication methods—such as basic authentication, client-side certificate, LTPA cookies and failover cookies—return cached authentication information automatically with no user intervention. The **pdadmin server task terminate session** action would not prevent return logins for a user using any of those authentication methods. You must additionally invalidate the appropriate user account in the registry.

Refer to the Administration C API Developer Reference for further information and for **ivadmin_server_performtask()** syntax.

When a user is logged out unexpectedly because of session termination, the original session cookie remaining on the user's browser becomes an old, or "stale" cookie that no longer maps to an existing entry in the WebSEAL session cache. When the user makes a subsequent request for a protected object, WebSEAL requires authentication and returns a login form. You can customize the login response to contain additional information that explains the reason for the new login requirement. For further information on this feature, see "Customized responses for old session cookies" on page 315.

Termination of all user sessions:

An administrator or a back-end application can use the Security Access Manager administration API to call the **pdadmin** command that terminates all sessions for a specific user based on the user's login ID.

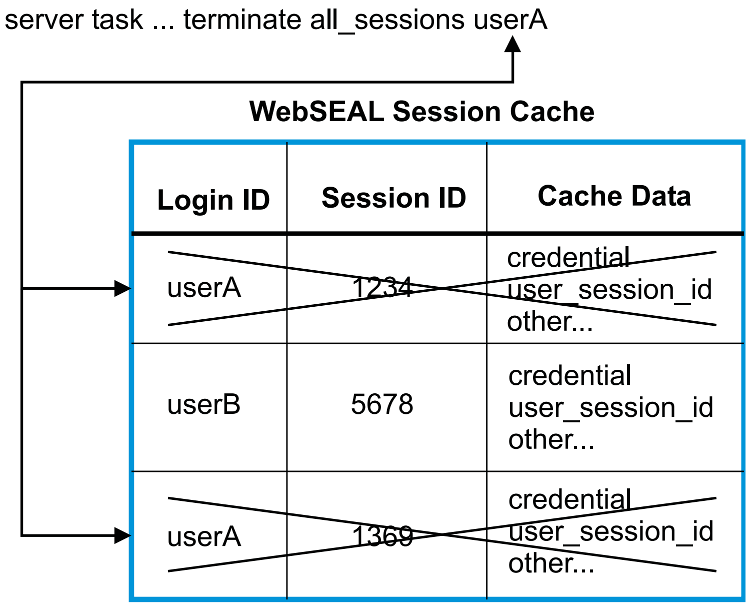
For example:

```
pdadmin> server task instance-webseald-host terminate all_sessions login_id
```

Note: The **pdadmin server task terminate all_sessions** command is not supported in a distributed session cache environment. Use the **dscadmin session terminate** command instead.

The user's login ID (*login_id*) can be passed to the junctioned back-end server in the Security Access Manager **iv-user** header. To accomplish this task, you must initially create the junction using the **-c iv_user** option and argument. See "Client identity in HTTP headers (–c)" on page 520.

The WebSEAL session cache is organized to cross-reference the user's login ID, the WebSEAL session ID, and other cache entry information. A user always has the same login ID across multiple sessions. Each WebSEAL session ID, however, is unique. The **pdadmin server task terminate all_sessions** command removes all cache entries belonging to a specific user's login ID.



Login ID	Session ID	Cache Data
~~userA~~	~~1234~~	~~credential~~ ~~user_session_id~~ ~~other...~~
userB	5678	credential user_session_id other...
~~userA~~	~~1369~~	~~credential~~ ~~user_session_id~~ ~~other...~~

Figure 55. Terminate all userA sessions

WebSEAL checks for appropriate permissions on the initiator of the **pdadmin** command before terminating user sessions.

It is important to consider the conditions under which this command might be used. If the intent is to make sure a certain group of users are removed from the secure domain entirely, the **pdadmin server task terminate all_sessions** command is only effective when, in addition, the accounts for those users are made not valid (removed).

Certain authentication methods—such as basic authentication, client-side certificate, LTPA cookies and failover cookies—return cached authentication information automatically with no user intervention. The **pdadmin server task terminate all_sessions** command would not prevent return logins for users using any of those authentication methods. You must additionally invalidate the appropriate user accounts in the registry.

When a user is logged out unexpectedly because of session termination, the original session cookie remaining on the user's browser becomes an old, or "stale" cookie that no longer maps to an existing entry in the WebSEAL session cache. When the user makes a subsequent request for a protected object, WebSEAL requires authentication and returns a login form. You can customize the login response to contain additional information that explains the reason for the new login requirement. For further information on this feature, see "Customized responses for old session cookies" on page 315.

User event correlation for back-end servers

Sometimes you must correlate individual events between WebSEAL and junctioned Web servers. A WebSEAL event can be uniquely identified by a combination of the session identifier and the transaction identifier.

Inserting event correlation data into HTTP headers:

About this task

For information on how to insert the session identifier into the HTTP stream, see "User session management for back-end servers" on page 593.

You can insert the transaction identifier into the HTTP stream using the WebSEAL HTTP-Tag-Value functionality. Use this function to insert elements from the user session into the HTTP stream. An extended attribute on a junction object in the WebSEAL protected object space defines the session elements that are included in the forwarded request. The **pdadmin object modify set attribute** command sets an attribute on an object:

```
<pdadmin> object modify <object_name> set attribute <attr-name> <attr-value>
```

The `<attr-name>` value should always be set to "HTTP-Tag-Value". The format of the `<attr-value>` should be:

```
<session_data_name>=<http_header_name>
```

To include the transaction identifier in the HTTP stream, set the `<session_data_name>` value to `session:tid`. The `<http_header_name>` value specifies the custom name of the HTTP header that delivers the transaction identifier across the junction.

The following example shows the pdadmin command that sets an HTTP-Tag-Value junction attribute. Enter the command on one line.

```
<pdadmin> object modify /WebSEAL/junctionA
set attribute HTTP-Tag-Value session:tid=TAM-TRANSACTION-ID
```

Inserting event correlation data into the WebSEAL request log:

You can use the **request-log-format** configuration entry to customize the contents of the WebSEAL request log to include event correlation data.

About this task

Format specifiers define the contents of the log text. The following format specifiers add the event correlation data to the request log:

Data	Format Specifier
transaction identifier	%d
session identifier	%{tagvalue_user_session_id}C

For more information, see request-log-format.

Dynamic URLs

This chapter discusses providing protection for dynamic URLs.

Topic Index:

Access control for dynamic URLs

The current Web environment gives users immediate access to rapidly changing information. Many Web applications dynamically generate Uniform Resource Locators (URLs) in response to each user request. These *dynamic* URLs usually exist only for a short time. Despite their temporary nature, dynamic URLs still need strong protection from unwanted use or access.

Dynamic URL components

Some Web application tools use dynamic URLs and hidden form elements to communicate requested operations (and values) to application servers. A dynamic URL augments the standard URL address with information about the specific operation and its values.

The dynamic data (operations, parameters, and values) provided to the Web application interface are located in the query string portion of the request URL.

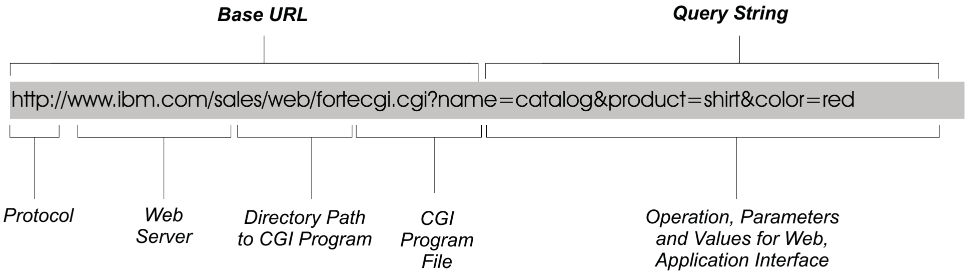

Base URL
Query String
http://www.ibm.com/sales/web/fortecgi.cgi?name=catalog&product=shirt&color=red
Protocol
Web Server
Directory Path to CGI Program
CGI Program File
Operation, Parameters and Values for Web, Application Interface

Figure 56. Passing data in the query string of a request URL

Access control for dynamic URLs: dynurl.conf

WebSEAL uses the protected object space model, access control lists (ACL), and protected object policies (POP) to secure dynamically generated URLs, such as those generated by database requests.

Each request to WebSEAL is resolved to a specific object as the first step in the authorization process. An ACL or POP applied to the object dictates the required protection on any dynamic URL mapped to that object.

Because dynamic URLs exist only temporarily, it is not possible to have entries for them in a pre-configured authorization policy database. Security Access Manager solves this problem by providing a mechanism where many dynamic URLs can be mapped to a single static protected object.

Mappings from objects to patterns are kept in a plain text configuration file called `dynurl.conf`.

You can use the LMI to manage the dynamic URL configuration file. Go to **Secure Web Settings** > **Global Settings** > **URL mapping**.

The name of this file is defined by the **dynurl-map** stanza entry in the **[server]** stanza of the WebSEAL configuration file:

```
[server]
dynurl-map = dynurl.conf
```

You must create this file; the file does not exist by default.

The existence of this file (with entries) during WebSEAL startup enables the dynamic URL capability.

Conversion of POST body dynamic data to query string format

When you map a URL regular expression to an object space entry, the query string format as produced from the GET request method must be used—regardless of the request method (POST or GET).

In a GET request, the dynamic data (operations, parameters, and values) provided to the Web application interface are located in the query string portion of the request URL.

In a POST request, the dynamic data (operations, parameters, and values) provided to the Web application interface are located in the request body.

To maintain the required query string structure required for access control evaluation, the POST request body should be in the query string format, as is done with the default POST media type of **`application/x-www-form-urlencoded`**. For example, a POST request has the following request URL:

```
http://server/login.form
```

Dynamic data (login information provided by the user) is located in the body of this POST request:

```
name=ibm&password=secure
```

To achieve the query string structure required for dynamic URL evaluation, WebSEAL appends the POST body information to the request URL:

```
http://server/login.form?name=ibm&password=secure
```

If the request body uses a different media type, such as **`multipart/form-data`**, and might include unencoded binary or reserved URL characters, the **`[server] suppress-dynurl-parsing-of-posts`** parameter should be set to **`'yes'`**.

Mapping ACL and POP objects to dynamic URLs

About this task

To specify access control of dynamic URLs, create the `dynurl.conf` configuration file and edit the file to map resource objects to patterns. Entries in the file are of the format:

```
object template
```

Security Access Manager uses a subset of UNIX shell pattern matching (including wildcards) to define the set of parameters that constitute one object in the object space. Any dynamic URL that matches those parameters is mapped to that object. For a list of supported wildcard pattern matching characters, see "Supported wildcard pattern matching characters" on page 57.

The following example illustrates the form of a dynamic URL (from a GET request) that performs credit balance lookup:

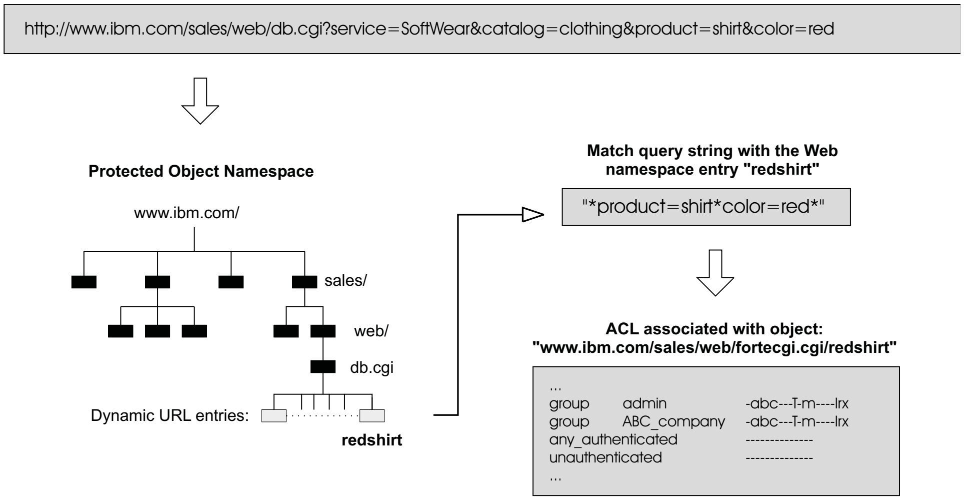
```
http://server-name/home-bank/owa/acct.bal?acc=account-number
```

The object that represents this dynamic URL would appear as follows:

```
http://server-name/home-bank/owa/acct.bal?acc=*
```

Careful examination of the dynamic URL in this example shows that it describes a specific account number. The object for account balances at **home-bank** shows that the ACL and POP permissions apply to *any* account, because the last portion of the entry (**acc=*)** uses the asterisk wildcard which matches all characters.

The following figure illustrates a complete scenario of a specific dynamic URL mapped to a specific protected object:


http://www.ibm.com/sales/web/db.cgi?service=SoftWear&catalog=clothing&product=shirt&color=red
Protected Object Namespace
www.ibm.com/
sales/
web/
db.cgi
Dynamic URL entries:
redshirt
Match query string with the Web namespace entry "redshirt"
"*product=shirt*color=red*"
ACL associated with object: "www.ibm.com/sales/web/fortecgi.cgi/redshirt"

```
...
group     admin          -abc---T-m----lrx
group     ABC_company    -abc---T-m----lrx
any_authenticated        -------------
unauthenticated          -------------
...
```

Figure 57. Authorization on a dynamic URL

Character encoding and query string validation

When dynamic URL is enabled, WebSEAL maps the dynamic data in the query strings of requests to objects requiring protection (access control). If WebSEAL receives dynamic data (in a POST body or query string) using characters that are neither UTF-8 nor from the character set in which WebSEAL runs, WebSEAL rejects the request and returns an error.

To securely map query strings to objects, the strings need to use the same character set known to WebSEAL and the back-end application server. Otherwise, dynamic URL access control could be circumvented by a request that uses a character accepted by the back-end application, but not accepted by WebSEAL.

The dynamic URL feature is affected by the value of the **decode-query** stanza entry in the **[server]** stanza of the WebSEAL configuration file. If WebSEAL is configured to not validate query strings in requests (`decode-query=no`), then dynamic URL mapping for authorization checking, if enabled, must be disabled. WebSEAL will not start if this condition is not met.

If WebSEAL (with dynamic URL enabled and `decode-query=yes`) is running in a non-UTF-8 environment, and request POST bodies (or query strings) contain UTF-8 characters, you can use the **utf8-form-support-enabled** stanza entry in the **[server]** stanza of the WebSEAL configuration file to allow WebSEAL to decode the UTF-8 coding in these requests.

Updating WebSEAL for dynamic URLs

Procedure

Use the **dynurl update** command to update the WebSEAL protected object space with entries made in the `dynurl.conf` configuration file.

1. Create, edit, or delete a dynamic URL entry in the `dynurl.conf` configuration file.
2. After making your changes, use the **dynurl update** command to update the server:

   ```
   pdadmin> server task instance_name-webseald-host_name dynurl update
   ```

   The *server-name* argument represents the unqualified host name of the WebSEAL machine.

Resolve dynamic URLs in the object space

Resolving a dynamic URL to an object is dependent on the ordering of the entries in the `dynurl.conf` configuration file.

When attempting to map a dynamic URL to an object entry, the list of mappings in the `dynurl.conf` file is scanned from top to bottom until the first matching pattern is found. When the first match is found, the corresponding object entry is used for the subsequent authorization check.

If no matches are found, WebSEAL uses the URL itself, minus the **http://***server* portion of the path.

Keep the mappings that correspond to the most restrictive ACLs higher up in the list. For example, if the **book.sales** procedure of a sales order application is to be restricted to just a book club group, but the rest of the sales order application can be accessed by all users, then the mappings should be in the order shown in the following table:

Object Space Entry	URL Template
/ows/sales/bksale	/ows/db-apps/owa/book.sales*
/ows/sales/general	/ows/db-apps/owa/*

Note that if the mapping entries were in the reverse order, all stored procedures in the `/ows/db-apps/owa` directory would map to the `/ows/sales/general` object. This could lead to possible breaches of security, due to this incorrect object space resolution.

ACL and POP Evaluation:

As soon as the dynamic URL has been resolved to an object space entry, the standard ACL or POP inheritance model is used to determine if the request should be processed or forbidden (due to insufficient privilege).

Configuration of limitations on POST requests

The content of a POST request is contained in the body of the request. In addition, a POST request contains the browser-determined length of this content and lists the value in bytes.

request-body-max-read

The **request-body-max-read** stanza entry in the **[server]** stanza of the WebSEAL configuration file limits the impact of large POST requests on WebSEAL by specifying the maximum number of bytes to read in as content from the body of POST requests. The content read in by WebSEAL is subject to authorization checks, as described earlier in this section.

The **request-body-max-read** stanza entry value is considered when the POST request is used for dynamic URL processing or Forms authentication. The default value is 4096 bytes:

```
[server]
request-body-max-read = 4096
```

Note that this stanza entry does not limit the maximum POST content size (which is unlimited). The stanza entry protects WebSEAL from processing a POST request of unreasonable size. For more information on modifying **request-body-max-read**, see "Modification of request-body-max-read" on page 237.

dynurl-allow-large-posts

Although the **request-body-max-read** stanza entry limits the amount of POST content read and processed by WebSEAL, it does not prevent the request, in its entirety, from being passed through to the application server. In this scenario, content that has not been validated is passed through to the application server. If the application server does not have its own authorization capabilities, the situation might result in a security risk.

The **dynurl-allow-large-posts** stanza entry allows you to control the way WebSEAL handles POST requests that have a content length larger than that specified by **request-body-max-read**. If the stanza entry value is set to "no" (default), WebSEAL rejects, in total, any POST request with a content length larger than that specified by **request-body-max-read**.

```
[server]
dynurl-allow-large-posts = no
```

If the stanza entry value is set to "yes", WebSEAL accepts the entire POST request, but only validates the amount of content equal to the **request-body-max-read** value.

```
[server]
dynurl-allow-large-posts = yes
```

Example 1:

- A large POST request is received (greater than the **request-body-max-read** value).
- **dynurl-allow-large-posts = no**
- Dynamic URLs are enabled.
- Result: 500 "Server Error"

Example 2:

- A large POST request is received (greater than the post-**request-body-max-read**).
- **dynurl-allow-large-posts = yes**
- Dynamic URLs are enabled.

- Result: WebSEAL compares the amount of content up to **request-body-max-read** with each of the regular expressions in the `dynurl.conf` configuration file, and performs an authorization check on the corresponding object if a match is found. Otherwise, the authorization check is performed on the object corresponding to the URL received, as usual. The portion of the request body past **request-body-max-read** is not validated.
- The following template contains the type of pattern matching arrangement that invites misuse by a large POST request:

  ```
  /rtpi153/webapp/examples/HitCount\?*action=reset*
  ```

Dynamic URLs summary and technical notes

Summary:

The following points summarize the WebSEAL configuration for securely handling dynamic URLs.

- To configure WebSEAL to securely handle dynamic URLs, create a dynamic URL configuration file (such as `dynurl.conf`). In the LMI, go to **Secure Web Settings** > **Global Settings** > **URL mapping** to create this file.
- The file must contain one or more lines of the format:

  ```
  object template
  ```
- If the file does not exist at WebSEAL startup, or is empty, dynamic URL capability is not enabled.
- After the file has been processed, the object name appears as a child resource in the WebSEAL object space.
- The template can contain a subset of the standard pattern matching characters. The template can also be an exact string with no pattern matching characters.

The following sample `dynurl.conf` file defines four objects representing some of the sample Web applications that are part of the IBM WebSphere product:

Object Entry	URL Template
/app_showconfig	/rtpi153/webapp/examples/ShowConfig*
/app_snoop	/rtpi153/servlet/snoop
/app_snoop	/rtpi025/servlet/snoop
/app_hitcount/ejb	/rtpi153/webapp/examples/HitCount\?source=EJB
/app_hitcount	/rtpi153/webapp/examples/HitCount*

Technical notes:

- Multiple URL templates can be mapped to the same object (for example, app_snoop maps to URLs on two different servers).
- Objects can be nested (for example, app_hitcount and app_hitcount/ejb).
- An incoming URL request is compared against templates in order, from top to bottom. When a match is found, processing stops. Therefore, place the more restrictive templates at the top of the file.
- To activate the definitions in the `dynurl.conf` file, issue the **dynurl update** command (use **pdadmin server task**).

The update occurs immediately and the objects appear in the Web Portal Manager when you refresh the protected object space view.

- Avoid uppercase characters in the object name. Use lowercase characters only.
- Do not use an object name that already exists in the protected object space.
- Before deleting an object in the `dynurl.conf` file, remove any ACLs attached to the object.
- If WebSEAL receives a POST body that contains characters that are not UTF-8 or not from the character set (code page) used by WebSEAL, WebSEAL will reject the request.
- Because dynamic URL evaluation operates on all requests and uses character pattern matching, any requests containing binary data in the query string (GET) or request body (POST) is rejected with a 500 Server Error.

Dynamic URL example: The Travel Kingdom

The following fictitious example illustrates how a corporate intranet can secure URLs generated by an Oracle Web Listener.

The dynamic URL Web server used in this example is the Oracle Web Listener. This technology can be applied equally to other dynamic URL Web servers.

The application

Travel Kingdom is an organization that offers clients a travel booking service over the Internet. The business intends to operate two Oracle database applications on its Web server — accessible across the Internet and from within the corporate firewall.

1. **Travel Booking System**

   Authorized customers can make bookings remotely and query their own current bookings. Travel Kingdom staff can also make bookings for telephone customers, process changes, and perform many other transactions. Because external customers pay for services with credit cards, the transmission of that information must be strongly secured.

2. **Administration Manager**

   Like most other companies, Travel Kingdom maintains an administration database containing salary, position, and experience information. This data is also accompanied by a photograph of each member of the staff.

The interface

The dynamic URL example provides access to stored procedures in the database.

An Oracle Web Server is configured to provide access to the following stored procedures in the database:

Table 69. Stored procedures in the database

Stored procedure	Description
/db-apps/owa/tr.browse	Gives all users the ability to inquire about travel destinations, prices, and so on.
/db-apps/owa/tr.book	Used to place a booking (travel agent staff or authenticated customers).
/db-apps/owa/tr.change	Used to review or change current bookings.

Table 69. Stored procedures in the database (continued)

Stored procedure	Description
/db-apps/owa/admin.browse	Used by any staff member to view unrestricted staff information, such as extension number, email address, and photograph.
/db-apps/owa/admin.resume	Gives staff members the ability to view or change their own resume information in the Administration database.
/db-apps/owa/admin.update	Used by Administration staff to update information on staff.

Web space structure:

A WebSEAL server is used to provide a secure interface to the unified Web space of Travel Kingdom.

- A junction (`/ows`) is made to the Oracle Web Server running both the travel booking application and the administration application.

The security policy

To provide suitable security to Web resources, while retaining an easy-to-use system, the business has established the following security goals:

- Travel agent staff have complete control over all bookings.
- Authenticated customers can make and change their own bookings, but cannot interfere with the travel data of other authenticated customers.
- Administration staff have complete access to all of the administration information.
- Travel Kingdom staff other than the Administration department can change their own resume information and view partial information of other members of staff.

Dynamic URL to object space mappings:

To achieve the security goals described above, the mappings from dynamic URLs to ACL object entries need to be configured as shown in the following table.

Remember that the ordering of these mappings is an important part of achieving the security goals discussed earlier.

Object Space Entry	URL Pattern
/ows/tr/browse	/ows/db-apps/owa/tr.browse\?dest=*&date=??/??/????
/ows/tr/auth	/ows/db-apps/owa/tr.book\?dest=*&depart=??/??/????& return=??/??/????
/ows/tr/auth	/ows/db-apps/owa/tr.change
/ows/admin/forall	/ows/db-apps/owa/admin.resume
/ows/admin/forall	/ows/db-apps/owa/admin.browse\?empid=[th]???
/ows/admin/auth	/ows/db-apps/owa/admin.update\?empid=????

Secure clients

Clients authenticate to WebSEAL over a secure, encrypted channel.

Customers who want to use the Web interface must additionally register with the Travel Kingdom Webmaster to receive an account.

Account and group structure:

Four groups are created on the system:

Staff Members of the Travel Kingdom organization.

TKStaff
Travel Kingdom travel agents.

AdminStaff
Members of the Travel Kingdom Administration Department. Note that Administration staff members are also in the Staff group.

Customer
Customers of Travel Kingdom who want to make their travel bookings across the Internet.

Each user is given an account in the secure domain to be individually identified by the WebSEAL server. The user's identity is also passed to the Oracle Web Servers to provide a single signon solution to all of the Web resources.

Access control

The dynamic URL example results in several access controls.

The following table lists the access controls resulting from application of the preceding information:

Table 70. Access controls for each object space entry

Object space entry	Access
/ows/tr/browse	unauthenticated Tr any_authenticated Tr
/ows/tr/auth	unauthenticated - any_other - group TKStaff Tr group Customer PTr
/ows/admin/forall	unauthenticated - any_other - group Staff Tr
/ows/admin/auth	unauthenticated - any_other - group AdminStaff Tr

Customers and TKStaff have the same privileges on the booking and travel plan maintenance objects, except that the customers must encrypt information (privacy permission) to give them further security when submitting sensitive data (such as credit card information) across the untrusted Internet.

Conclusion

This example illustrates the concepts of deploying a system capable of:

- Securing sensitive information
- Authenticating users
- Authorizing access to sensitive information

In addition, the identities of the authenticated users of the system are known to both the WebSEAL and Oracle Web Servers, and are used to provide an auditable, single sign-on solution.

Internet Content Adaptation Protocol (ICAP) Support

The Internet Content Adaptation Protocol (ICAP) is designed to offload the processing of Internet-based content to dedicated servers. ICAP helps free up resources and standardize how features are implemented.

A proxy server, such as WebSEAL, can be configured to pass client requests and responses through ICAP servers. These ICAP servers can focus on specific, value-added services, and therefore be more efficient. For example, if an ICAP server handles language translation only, it might be more efficient than a web server that performs many additional tasks.

ICAP is a "lightweight" HTTP-like protocol. ICAP clients can pass HTTP-based (HTML) messages or content to ICAP servers for adaptation. Adaptation refers to performing the particular value added service, such as content manipulation, for the associated client request or response.

WebSEAL supports both TCP connections and SSL connections to the ICAP server.

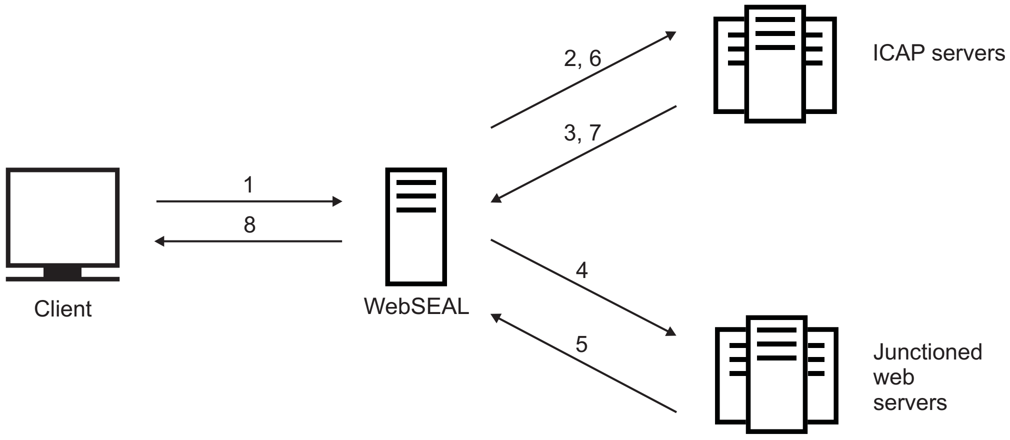
For more information, see Request For Comments (RFC) 3507 - Internet Content Adaptation Protocol (ICAP): http://www.ietf.org/rfc/rfc3507.txt.

Example

Additional examples of ICAP server functions are: virus scanning, language translation, and content filtering.

ICAP integration with WebSEAL - Workflow

A client request follows the path depicted in the diagram:



1. A client sends a request to WebSEAL.
2. WebSEAL passes the request to the ICAP server.
3. The ICAP server uses the Internet Content Adaptation Protocol (ICAP) to generate a response and sends it back to WebSEAL. The response can be one of the following:

   a. A modified version of the request that is forwarded to other ICAP servers in the chain, or to the backend Web server.
   b. An HTTP response for this request that must then be sent back to the client.
   c. An error, which is processed by WebSEAL.
4. The modified request is then sent to the backend Web server.
5. A response is generated by the backend Web server.
6. WebSEAL passes the response to the ICAP server
7. A response is generated by the ICAP server and sent back to WebSEAL. The response can be one of the following:
   a. A modified version of the response.
   b. An error.
8. The response is sent to the client.

Note: Steps 2-3 and 6-7 can be repeated multiple times if multiple ICAP servers are configured.

Scope of functionality

The following information captures the scope of integrating ICAP with WebSEAL.

WebSEAL provides support for:
- Request for modification (REQMOD)
- Request for response (RESPMOD)

WebSEAL does not support the following optional aspects of ICAP RFC 3507:
- Message preview
- "204 No Content" Responses outside of previews
- OPTIONS method
- Caching
- Authentication
- Encryption

Note: See the ICAP RFC 3507: http://www.ietf.org/rfc/rfc3507.txt for more information on each of the aspects.

Configuration of ICAP support within WebSEAL

The configuration of ICAP support within WebSEAL is flexible and allows only those transactions that require the ICAP intervention to be sent to the ICAP servers.

An Administrator can configure and control the applications that require ICAP processing. Configuration of ICAP support within WebSEAL consists of two parts:
- Configuration file: Used to define ICAP servers.
- Protected Object Policy (POP): Used to define the resources that trigger a call to the ICAP servers.

Configuration file

A stanza entry called **`[ICAP: <resource>]`** is added to the configuration file. The stanza entry is used to define the different ICAP resources. Each resource consists of:

- A URL for the ICAP server, which defines the ICAP server's address and whether TCP or SSL is used for the connection. When an SSL connection is defined, the system uses the keystore that is configured in the **[junction]** stanza if it exists. If not, the system uses the keystore that is configured in the **[ssl]** stanza.
- A transaction list that defines whether the ICAP server is used in processing the HTTP request or response.
- A timeout value that defines the maximum length of time (in seconds) that WebSEAL waits for a response from the ICAP server.
- An optional SSL keyfile label that defines the certificate to be used if client certificate authentication is required.

For more information, see [ICAP:<resource>] stanza.

Note: The `<resource>` in the stanza name corresponds to the name of the resource in the POP. Multiple resources might be specified in the configuration file.

Example

```
[ICAP:resource_a]
URL = icap://icap_svr.tivoli.com:1344/
transaction = req
timeout = 120
[ICAP:resource_b]
URL = icap://icap_svr.tivoli.com:1344/
transaction = rsp
timeout = 120
```

Note: The preceding example establishes a TCP connection to the ICAP server. If you want to use an SSL connection to the ICAP server, use "`icaps://`" instead of "`icap://`". You can also specify an SSL keyfile label that defines the certificate to be used if client certificate authentication is required.

```
[ICAP:resource_a]
URL = icaps://icap_svr.tivoli.com:1345/
transaction = req
timeout = 120
[ICAP:resource_b]
URL = icaps://icap_svr.tivoli.com:1345/
transaction = rsp
timeout = 120
ssl-keyfile-label = my_certificate
```

Protected Object Policy (POP)

A Protected Object Policy (POP) is used to enable the pre-defined ICAP resource for appropriate parts of the object space. This mechanism provides full control over which resources incur the additional impact of the ICAP processing. The POP must have:

- An extended attribute created with the name 'ICAP', and
- A value that matches the name of one of the configured ICAP resources.

Multiple attributes of the same name can be created if multiple ICAP servers are required to handle the processing of a particular object or request.

The following example shows what the POP might look like:

```
pdadmin sec_master> pop show ICAPPop attribute ICAP
    ICAP
        resource_a
        resource_b
```

Note: `resource_a` and `resource_b` correspond to the following configuration stanzas: `[ICAP:resource_a]` and `[ICAP:resource_b]`.

Chapter 11. Authorization decision information retrieval

The following topics describe how WebSEAL can provide, or acquire, authorization decision information (ADI) required to evaluate authorization rules that protect resources in the Security Access Manager domain.

Overview of ADI retrieval

The Security Access Manager authorization rules evaluator performs authorization decisions based on Boolean logic applied to specific authorization decision information (ADI). You can find detailed information about the construction of authorization rules (using Boolean logic) and ADI in the Access Manager Platform and Supporting Components administration topics.

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. See the Access Manager Platform and Supporting Components administration topics for further information about this topic.
- The user credential.

  The user credential is always included with the function call to the authorization rules evaluator, so it is immediately available.
- The resource manager environment (application context).

  A resource manager, such as WebSEAL, can be configured to provide ADI from its own environment. For example, WebSEAL can provide ADI contained in parts of the client request. A special prefix is used in the authorization rule to "trigger" this type of ADI source.
- An external source through the Security Access Manager attribute retrieval service.

  ADI can be obtained externally through the attribute retrieval service. The entitlement service of the resource manager makes a call to the attribute retrieval service. ADI from the external source is returned in XML format to the authorization rules evaluator.

  Note: The attribute retrieval service is deprecated. IBM might remove this capability in a subsequent release of the product.

ADI retrieval from the WebSEAL client request

In a WebSEAL environment, authorization rules can be written to require authorization decision information (ADI) contained in the client HTTP/HTTPS request. ADI can be contained in the request header, the request query string, and the request POST body.

Authorization decision information is referred to by an XML container name in authorization rules. A special WebSEAL-specific prefix in the container name is used to alert the authorization rules evaluation process that WebSEAL can interpret this parameter correctly and return a value.

Prefixes can be specific to any resource manager. Accordingly, the resource manager must be designed to respond appropriately to a request for ADI.

The following container names contain prefixes that are appropriate for WebSEAL:

- `AMWS_hd_`*`name`*

  Request header container name. The value of the HTTP header called *name* in the HTTP request is returned to the authorization rules evaluator as ADI.
- `AMWS_qs_`*`name`*

  Request query string container name. The value of *name* in the request query string is returned to the authorization rules evaluator as ADI.
- `AMWS_pb_`*`name`*

  Request POST body container name. The value of *name* in the request POST body is returned to the authorization rules evaluator as ADI.

The following process flow helps illustrate how prefixes enable the extraction of ADI from client requests:

1. An authorization rule is written that requires ADI from the client request (for example, a specific HTTP header in the request).

   In this example, the `AMWS_hd_` prefix is used in the container name specified in the rule. The prefix must be specified by the **resource-manager-provided-adi** stanza entry in the **[aznapi-configuration]** stanza of the WebSEAL configuration file. The authorization service incorporates this configuration information during its initialization. This WebSEAL-specific prefix alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.
2. The authorization rules evaluation process tries to evaluate, for example, the `AMWS_hd_host` container name in a rule. The `AMWS_hd_` prefix alerts the authorization evaluation process that WebSEAL can interpret this container name correctly and return a value.
3. The `AMWS_hd_host` container name is sent to WebSEAL.

   WebSEAL is designed to recognize and interpret the `AMWS_hd_` prefix.
4. WebSEAL responds to the `AMWS_hd_host` container name by looking for the "host" header in the client request and extracting the value associated with that header.
5. WebSEAL returns the "host" header value (as an XML container) to the authorization rules evaluation process.
6. The authorization rules evaluation process uses the value as ADI in its evaluation of the rule.

The **resource-manager-provided-adi** stanza entry in the **[aznapi-configuration]** stanza of the WebSEAL 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** stanza entry:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_qs_
resource-manager-provided-adi = AMWS_pb_
resource-manager-provided-adi = AMWS_hd_
```

The **permission-info-returned** stanza entry in the **[aznapi-configuration]** stanza of the WebSEAL configuration file appears by default. This stanza entry specifies the permission information returned to the resource manager (for example, WebSEAL) from the authorization service. The following example is entered as one line, with a single space separating the two permission types:

```
[aznapi-configuration]
permission-info-returned = azn_perminfo_rules_adi_request
azn_perminfo_reason_rule_failed
```

The **azn_perminfo_rules_adi_request** setting allows the authorization service to request ADI from the current WebSEAL client request. The **azn_perminfo_reason_rule_failed** setting specifies that rule failure reasons be returned to the resource manager (this setting is required for –R junctions—see "Supplying a failure reason across a junction" on page 618).

Example: Retrieving ADI from the request header

The following example authorization rule requires the name of the Internet host and port number of the resource being requested. (If port number is omitted, the default port for the requested service is used; for example, port 80 is used for an HTTP URL.) The client request is set up to include the host name value in the "host" header of the request. The use of the `AMWS_hd_` prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_hd_host = "machineA"'>!TRUE!</xsl:if>
```

WebSEAL is designed to know how to handle the extraction of ADI information from the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_hd_
```

WebSEAL looks for this information in the request header name "host." WebSEAL extracts the value contained in the "host" header and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request's "host" header is "machineA."

In a similar manner, information required to evaluate an authorization rule can come from the request POST body or the query string of the request.

Example: Retrieving ADI from the request query string

The following example authorization rule requires the name of the client's ZIP code as passed in the query string of a GET request (as submitted in response to a form). The client request is set up to include the ZIP code value in the "zip" field of the request query string.

```
https://www.service.com/location?zip=99999
```

The use of the `AMWS_qs_` prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_qs_zip = "99999"'>!TRUE!</xsl:if>
```

WebSEAL is designed to know how to handle the extraction of ADI information form the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_qs_
```

WebSEAL looks for this information in the request query string under the field name "zip". WebSEAL extracts the value contained in the "zip" field and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request's query string "zip" field is "99999".In a similar manner, information required to evaluate an authorization rule can come from the request POST body or the request header.

Example: Retrieving ADI from the request POST body

The following example authorization rule requires the name of the client's total purchase amount from a Web shopping cart as passed in the body of a POST request (as submitted in response to a form). The client request is set up to include the total purchase value in the "purchase-total" field of the request POST body.

The use of the `AMWS_pb_` prefix in the rule alerts the authorization evaluation process that the required ADI is available in the client request and that WebSEAL knows how to find, extract, and return this ADI.

```
<xsl:if test='AMWS_pb_purchase-total &lt; "1000.00"'>!TRUE!</xsl:if>
```

WebSEAL is designed to know how to handle the extraction of ADI information from the request:

```
[aznapi-configuration]
resource-manager-provided-adi = AMWS_pb_
```

WebSEAL looks for this information in the request POST body under the field name "purchase-total". WebSEAL extracts the value contained in the "purchase-total" field and returns it to the authorization evaluation process.

The example authorization rule is evaluated to be true if the value provided in the request's POST body "purchase-total" field is less than "1000.00". In a similar manner, information required to evaluate an authorization rule can come from the request header or the query string of the request.

ADI retrieval 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 user's credential information. The authorization rules evaluator always looks through this credential information for any ADI required by the rule being processed. The authorization rule can use the value from any field in the credential, including extended attributes added to the credential during authentication.

Supplying a failure reason across a junction

About this task

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, WebSEAL receives the reason for the rule's failure along with the standard "forbidden" message from the authorization service. The failure reason is usually ignored and the "forbidden" decision is enforced

You can optionally configure WebSEAL to reject this standard response and allow denied requests to proceed across a junction to a back-end service application. The request is accompanied by the failure reason provided in the authorization rule. The back-end service application can then have the opportunity to proceed with its own response to the situation. This optional configuration occurs during the creation of the junction to the back-end service application.

Authorization rules are typically used in conjunction with service applications that can understand and handle this more sophisticated level of access control. In some cases, it is necessary for the service application to receive a request that is denied by the 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 user's credit limit. It is important for the shopping cart application to receive the entire request and the reason for failure. Now the shopping cart application can take matters into its own hands and provide a user-friendly response, such as advising the user to eliminate a portion of the order. The interaction with the user is preserved rather than cut off.

Procedure

To allow denied requests and failure reason information to proceed across a junction to the back-end service application, configure the junction with the **–R** option. When WebSEAL receives an access denied decision on a request for an object located on a **–R** junction, WebSEAL reverses the denial response, inserts the failure reason into an HTTP header called "AM_AZN_FAILURE", inserts that header into the request, and passes the request on to the back-end application. Always use this option with caution. It is important to coordinate the use of failure reasons in authorization rules with a service application's ability to interpret and respond to this information. You do not want to accidently create a situation where access is granted to a resource controlled by an application that cannot respond accurately to the AM_AZN_FAILURE header.

Dynamic ADI retrieval

Rules can be written requiring ADI that cannot be found in any of the information that the Security Access Manager authorization service has access to. In these cases, it is necessary to retrieve the ADI from an outside source. A dynamic ADI entitlement retrieval service can perform this retrieval in real time. The attribute retrieval service, currently provided with WebSEAL, is one type of entitlement retrieval service.

The attribute retrieval service provides communication and format translation services between the WebSEAL entitlement service library and an external provider of authorization decision information. The process flow for the attribute retrieval service is described in the following diagram:

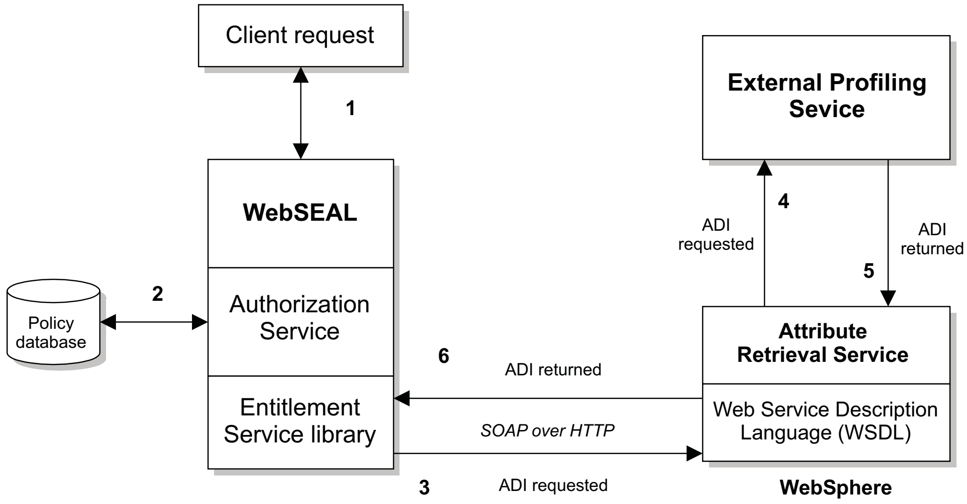


Figure 58. Dynamic ADI retrieval

1. The client makes a request for a resource protected by an authorization rule.
2. The authorization rules evaluator, which is part of the authorization service, determines that specific ADI is required to complete the rule evaluation. The ADI requested is not available from the user credential, the authorization service, or WebSEAL.
3. The task of ADI retrieval is sent to the attribute retrieval service through the entitlements service library. This service formats the request for ADI as a SOAP request. The SOAP request is sent over HTTP to the Web Service Description Language (WSDL) interface of the attribute retrieval service.
4. The attribute retrieval service formats the request appropriately for the external provider of ADI.
5. The external provider of ADI returns the appropriate ADI.
6. The ADI is formatted in another SOAP container and returned to the WebSEAL entitlements service. Now the authorization rules evaluator has the necessary information to evaluate the rule and decide whether to accept or deny the original client request.

Note: The WebSEAL attribute retrieval service is deprecated. IBM might remove this capability in a subsequent release of the product.

Alternatively, you can create and deploy a custom attribute retrieval service. The Security Access Manager Application Development Kit includes a WSDL file to get you started. The file is in the following locations:

AIX, Linux, or Solaris

```
/opt/PolicyDirector/example/amwebars/azn_ent_amwebars.wsdl
```

Windows

```
C:\Program Files\Tivoli\Policy Director\example\amwebars\
azn_ent_amwebars.wsdl
```

For more information about creating and deploying web services, see the IBM WebSphere Application Server Information Center at http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp.

Configuring WebSEAL to use the attribute retrieval service

About this task

Perform the following tasks to configure WebSEAL to use the attribute retrieval service.

Procedure

1. In the WebSEAL configuration file, specify the identification name (ID) of the attribute retrieval service that is queried when missing ADI is detected during a rules evaluation. In this case, the attribute retrieval service is specified:

   ```
   [aznapi-configuration]
   dynamic-adi-entitlement-services = AMWebARS_A
   ```

2. In the WebSEAL configuration file, use the service ID for the configured attribute retrieval service as a parameter to specify the appropriate built-in library that formats outbound ADI requests and interprets incoming responses:

   For example:

   ```
   [aznapi-entitlement-services]
   AMWebARS_A = azn_ent_amwebars
   ```

3. In the WebSEAL configuration file, specify the URL to the attribute retrieval service located in the WebSphere environment.

   For a TCP connection (entered as one line):

   ```
   [amwebars]
   service-url = http://websphere_hostname:websphere_port
    /amwebars/amwebars/ServiceToIServicePortAdapter
   ```

Chapter 12. Guidelines for changing configuration files

These guidelines are provided to help you make changes to the Security Access Manager configuration files. The guidelines are divided into the following categories:

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, `usessl` and `UseSSL` 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 recommended.
- 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 **[aznapi-configuration]** stanza section contains the following entry:

```
auditcfg = {azn|authn|mgmt}
```

The value for **auditcfg** is expected to be `azn`, `authn`, or `mgmt`. 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, specify the file name exactly as it is displayed in the LMI. Do not include any directory path information.

Integers

Many stanza entries expect the value for the entry to be expressed as an integer. When defining an entry with an integer, consider the following guidelines:

- Stanza entries that take an integer value expect integer values within a valid range. The range is described in terms of a *minimum* value and a *maximum* value.

  For example, in the **[logging]** stanza, the **logflush** stanza entry has a minimum value of `1` second and a maximum value of `600` seconds.
- For some entries, the integer value must be positive, and the minimum value is 1. For other entries, a minimum integer value of 0 is allowed.

  Use caution when setting an integer value to 0. For example, an integer value of 0 might disable the function that is controlled by that stanza entry. For example, in the `[ivacld]` stanza, the entry `tcp-req-port = 0` disables the port number. Or, an integer value of 0 might indicate that the number is unlimited. For example, in the **[ldap]** stanza, the entry `max-search-size = 0` means there is no limit to the maximum search size.
- For some entries requiring integer values, Security Access Manager does not impose an upper limit for the maximum number allowed. For example, there is typically no maximum for timeout-related values, such as `timeout = `*`number`* in the **[ldap]** stanza.

  For this type of entry, the maximum number is limited only by the size of memory allocated for an integer data type. This number can vary, based on 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. Security Access Manager 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, is 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.

Chapter 13. Command reference

This appendix contains a subset of the **pdadmin** commands that are specific to WebSEAL tasks.

See the Web command reference topics.

Reading syntax statements

The reference documentation uses the following special characters to define syntax:

[ ] Identifies optional options. Options not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous option.

| Indicates mutually exclusive information. You can use the option to the left of the separator or the option to the right of the separator. You cannot use both options in a single use of the command.

{ } Delimits a set of mutually exclusive options when one of the options is required. If the options are optional, they are enclosed in brackets ([ ]).

\ Indicates that the command line wraps to the next line. It is a continuation character.

The options for each command are listed alphabetically in the Options section. When the order of the options must be used in a specific order, this order is shown in the syntax statement.

help

Obtains system help for **pdadmin** commands and options.

This command does not require a login or authentication to use.

Syntax

help {*topic* | *command*}

Options

topic Specifies the help command topic for which help is needed.

command
Specifies the miscellaneous command for which help is needed.

Return codes

0 The command completed successfully.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

- The following example lists help topics and commands:

  ```
  pdadmin> help
  ```

  Output is similar to:

  ```
  Type 'help <topic>' or 'help <command> for more information

  Topics:
  acl
  action
  admin
  authzrule
  config
  context
  domain
  errtext
  exit
  group
  help
  login
  logout
  object
  objectspace
  policy
  pop
  quit
  rsrc
  rsrccred
  rsrcgroup
  server
  user

  Miscellaneous Commands:
  exit
  help
  quit
  ```

- The following example lists the options and descriptions that are available whether you specify the topic `action` or `action create`:

  ```
  pdadmin> help action
  ```

  Or:

  ```
  pdadmin> help action create
  ```

  Output is similar to:

  ```
  action create <action-name> <action-label> <action-type>
  Creates a new ACL action definition
  action create <action-name> <action-label> <action-type> <action-group-name>
  Creates a new ACL action definition in a group
  ...
  ```

server list

Lists all registered Security Access Manager servers.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server list`

Description

Lists all registered Security Access Manager servers. The name of the server for all server commands, except for the **server list** command, must be entered in the exact format as it is displayed in the output of this command.

Options

None.

Return codes

0 The command completed successfully.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example lists all registered servers if the Security Access Manager component is the authorization server:

```
pdadmin> server list
```

Output is similar to:

```
ivacld-topserver
ivacld-server2
ivacld-server3
ivacld-server4
```

server task add

The **server task add** command adds an additional back-end application server to an existing WebSEAL junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **add -h** *host_name* [*options*] *junction_point*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

junction_point
Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

options Specifies the options that you can use with the **server task add** command. These options include:

-D "*dn*"
Specifies the distinguished name of the back-end server certificate. This value, matched with the actual certificate DN, enhances authentication and provides mutual authentication over SSL. For example, the certificate for `www.example.com` might have a DN of

```
"CN=WWW.EXAMPLE.COM,OU=Software,O=example.com\, Inc,L=Austin,
ST=Texas,C=US"
```

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

-H *host_name*
Specifies the DNS host name or IP address of the proxy server. Valid values for *host_name* include any valid IP host name. For example:

```
www.example.com
```

This option is used for junctions that were created with the type of `tcpproxy` or `sslproxy`.

–i Indicates that the WebSEAL server does not treat URLs as case-sensitive. This option is used for junctions that were created with the type of `tcp` or `ssl`.

-p *port* Specifies the TCP port of the back-end server. The default value is 80 for TCP junctions and 443 for SSL junctions. This option is used for junctions that were created with the type of `tcp` or `ssl`.

-P *port* For proxy junctions that were created with the type of `tcpproxy` or `sslproxy` this option specifies the TCP port number for the HTTP proxy server. The default value is 7138.

For *port*, use any valid port number. A valid port number is any positive number that is allowed by TCP/IP and that is not currently being used by another application. Use the default port number value, or use a port number that is greater than 1000 that is currently not being used.

This option is also valid for mutual junctions to specify the HTTPS port of the back-end third-party server.

-q *url* Required option for back-end Windows servers. Specifies the relative path for the **query_contents** script. By default, Security Access Manager looks for this script in the `/cgi_bin` subdirectory. If this directory is different or the **query_contents** file is renamed, use this option to indicate to WebSEAL the new URL to the file.

This option is used for junctions that were created with the type of `tcp` or `ssl`.

-u *uuid*

Specifies the UUID of this back-end server when connected to WebSEAL over a stateful junction that was using the **–s** option. This option is used for junctions that were created with the type of `tcp` or `ssl`.

-v *virtual_hostname*

Specifies the virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. Use this option when the back-end junction server expects a host name header, because you are junctioning to one virtual instance of that server. The default HTTP header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host. This option is used for junctions that were created with the type of `tcp` or `ssl`.

-V *virtual_hostname*

Virtual host name represented on the back-end server. This option supports a virtual host setup on the back-end server. This option is only used for mutual junctions and corresponds to the virtual host which is used for HTTPS requests.

You use **–V** when the back-end junction server expects a host name header because you are junctioning to one virtual instance of that server. The default HTTPS header request from the browser does not know that the back-end server has multiple names and multiple virtual servers. You must configure WebSEAL to supply that extra header information in requests destined for a back-end server set up as a virtual host.

-w Indicates Microsoft Windows file system support.

This option is used for junctions that were created with the type of `tcp` or `ssl`.

-h *host_name*

Required option. Specifies the DNS host name or IP address of the target back-end application server. Valid values for *host_name* include any valid IP host name. For example:

```
www.example.com
```

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name/junction_point`* object. For example, the **sec_master** administrative user has permission by default.

Note: This command is available only when WebSEAL is installed.

Return codes

0 The command completed successfully. For WebSEAL **server task** commands, the return code will be 0 when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example creates a new junction for the WebSEAL server named WS1 to the back-end server named APP1 and adds another back-end server named APP2 to the same junction point:

```
pdadmin> server task default-webseald-WS1 create -t tcp -h APP1 -s /mnt

pdadmin> server task default-webseald-WS1 add -h APP2 /mnt
```

See also

"server task create" on page 635
"server task delete" on page 642
"server task remove" on page 654
"server task show" on page 656

server task cache flush all

The **server task cache flush all** command flushes the HTML document cache.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **cache flush all**

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/` object. For example, the **sec_master** administrative user has permission by default.

Note: This command is available only when WebSEAL is installed.

Return codes

0 The command completed successfully.

Note: For WebSEAL **`server task`** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **`pdadmin`** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example flushes all Web document caches:

```
pdadmin> server task default-webseald-abc.ibm.com cache flush all
```

See also

None.

server task cluster restart

The **`server task cluster restart`** command applies any configuration changes to the entire cluster and restarts the updated servers.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *server_name* **cluster restart** [*-ripple*|*-status*]

Description

If the **-status** option is used then the command provides a status update on the most recent cluster restart.

If the **-status** option is not used then the command causes each server in the cluster to examine the master for configuration updates. If required, the server updates its configuration data to synchronize with the master configuration information. If updates are applied then the server is restarted. The master server is restarted after all of the slave servers in the cluster have been updated and restarted as required.

Note: This command is only available on the configured cluster master server.

Options

server_name
Specifies the name of the master authorization server on which the configuration data resides.

-ripple Indicates that each WebSEAL server in the cluster must restart in sequence rather than being restarted in parallel.

-status Monitors the progress of a cluster restart. This option returns one of the following messages based on the current status of the cluster:

```
DPWAD0444I   The cluster has been restarted.

DPWAD0443I   The cluster is in the process of being restarted.
```

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the master `/WebSEAL/`*`host_name-instance_name`*`/` object. For example, the **sec_master** administrative user has permission by default.

Note:

An error is returned if the server is not configured as the master of the cluster. Before executing this command, you must ensure that the cluster is configured correctly using appropriate **[cluster]** stanza entries. For more information, see [cluster] stanza.

You cannot use the two options **-ripple** and **-status** at the same time. The **-ripple** option is available when initiating a cluster restart, while the **-status** option monitors the progress of the most recent cluster restart request.

Return codes

0 The command completed successfully.

Note: For WebSEAL **`server task`** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **`pdadmin`** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Default value

By default the cluster is restarted in parallel.

Examples

The following example restarts the cluster in sequence. This command must be executed on the cluster master server, which in this example is **default-webseald-master.ibm.com**.

```
server task default-webseald-master.ibm.com cluster restart -ripple
```

In this example, the following command can be used at any time after the previous request to monitor the progress of the cluster restart:

```
server task default-webseald-master.ibm.com cluster restart -status
```

server task create

The **`server task create`** command creates a WebSEAL junction point.

Requires authentication (administrator ID and password) to use this command.

Syntax

For local junctions:

`server task` *instance_name*-**webseald**-*host_name* **create -t** *type* [*options*] *junction_point*

For non-local junctions:

`server task` *instance_name*-**webseald**-*host_name* **create -t** *type* **-h** *host_name* [*options*] *junction_point*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **`server list`** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

junction_point

Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

options Specifies the options that you can use with the **`server task create`** command. The options include:

-a *address*

Specifies the local IP address for WebSEAL to use when communicating with the target back-end server. If this option is not provided, WebSEAL uses the default address as determined by the operating system.

If an address is supplied for a particular junction, WebSEAL is modified to bind to this local address for all communication with the junctioned server.

-A
Enables or disables lightweight third-party authentication mechanism (LTPA) junctions. This option requires the **-F** and **-Z** options. The **-A**, **-F**, and **-Z** options all must be used together.

This option is valid for all junctions except for the type of `local`.

-2
You can use this option in conjunction with the **-A** option to specify that LTPA version 2 cookies (LtpaToken2) are used. The **-A** option without the **-2** option specifies that LTPA version 1 cookies (LtpaToken) are used.

-b *BA_value*
Defines how the WebSEAL server passes the HTTP BA authentication information to the back-end server, which is one of the following values:
- `filter` (default)
- `ignore`
- `supply`
- `gso`

This option is valid for all junctions except for the type of `local`.

-B
Indicates that WebSEAL uses the BA header information to authenticate to the back-end server and to provide mutual authentication over SSL. This option requires the **-U** and **-W** options.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

-c *header_type*
Inserts the Security Access Manager client identity in HTTP headers across the junction. The *header_type* argument can include any combination of the following Security Access Manager HTTP header types:
- `{iv-user|iv-user-l}`
- `iv-groups`
- `iv-creds`
- `all`

The header types must be comma separated, and cannot have spaces between the types. For example: `-c iv_user,iv_groups`

Specifying **-c** `all` is the same as specifying **-c** `iv-user,iv-groups,iv-creds`.

This option is valid for all junctions except for the type of `local`.

-C
Indicates single signon from a front-end WebSEAL server to a back-end WebSEAL server. The **-C** option is not mutual authentication.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

-D "*dn*"

Specifies the distinguished name of the back-end server certificate. This value, matched with the actual certificate DN enhances authentication and provides mutual authentication over SSL. For example, the certificate for www.example.com might have a DN of

```
"CN=WWW.EXAMPLE.COM,OU=Software,O=example.com\, Inc,L=Austin,
ST=Texas,C=US"
```

This option is valid only with junctions that were created with the type of ssl or sslproxy.

-e *encoding_type*

Specifies the encoding to use when generating HTTP headers for junctions. This encoding applies to headers that are generated with both the **-c** junction option and tag-value. The following values for encoding are supported:

utf8_bin

WebSEAL sends the headers in UTF-8.

utf8_uri

WebSEAL sends the headers in UTF-8 but URI also encodes them. This behavior is the default behavior.

lcp_bin

WebSEAL sends the headers in the local code page of the WebSEAL server.

lcp_uri

WebSEAL sends the headers in the local code page of the WebSEAL server, but URI also encodes them.

This option is valid for all junctions except for the type of local.

-f Forces the replacement of an existing junction.

This option is used for junctions that were created with any junction type.

-F *keyfile*

Specifies the name of the keyfile used to encrypt LTPA cookie data.

The **-F** option requires **-A** and **-Z** options. The **-A**, **-F**, and **-Z** options all must be used together.

This option is valid for all junctions except for the type of local.

-H *host_name*

Specifies the DNS host name or IP address of the proxy server. The **-P** option also supports proxy server junctions. Valid values for *host_name* include any valid IP host name. For example:

```
proxy.www.example.com
```

This option is valid only with junctions that were created with the type of tcpproxy or sslproxy.

-i Indicates that the WebSEAL junction does not treat URLs as case-sensitive. To correctly authorize requests for junctions that are not case-sensitive, WebSEAL does the authorization check on a lowercase version of the URL. For example, a Web server that is running on a Windows operating system treats requests for INDEX.HTM and index.htm as requests for the same file.

Junctions to such a Web server should be created with the **-i** or **-w** option. ACLs or POPs that are attached to objects beneath the junction point should use the lowercase object name. An ACL attached to `/junction/index.htm` will apply to all of the following requests if the **-i** or **-w** option is used:

```
/junction/INDEX.HTM
/junction/index.htm
/junction/InDeX.HtM
```

This option is valid for all junctions except for the type of `local`. Local junctions are not case-sensitive only on Win32 platforms; all other platforms are case-sensitive.

-I Ensures a unique Set-Cookie header name attribute when using the **-j** option to modify server-relative URLs in requests.

This option is valid for all junctions except for the type of `local`.

-j Supplies junction identification in a cookie to handle script-generated server-relative URLs.

This option is valid for all junctions except for the type of `local`.

-J `trailer,inhead,onfocus,xhtml10`

Controls the junction cookie JavaScript block.

Use **–J trailer** to append (rather than prepend) the junction cookie JavaScript to HTML page returned from back-end server.

Use **–J inhead** to insert the JavaScript block between <head> </head> tags for HTML 4.01 compliance.

Use **–J onfocus** to use the onfocus event handler in the JavaScript to ensure the correct junction cookie is used in a multiple-junction/multiple-browser-window scenario.

Use **–J xhtml10** to insert a JavaScript block that is HTML 4.01 and XHTML 1.0 compliant.

For complete details on this option, see "Control on the junction cookie JavaScript block" on page 436.

-k Sends WebSEAL session cookies to the junction server. By default, cookies are removed from requests that are sent to the server.

This option is valid for all junctions except for the type of `local`.

-K "*key_label*"
Specifies the key label of the client personal certificate that WebSEAL should present to the back-end server. Use of this option allows the junction server to authenticate the WebSEAL server using client certificates.

This option is valid only with junctions that were created with the type of `ssl` and `sslproxy`.

-l *percent*
Defines the soft limit for consumption of worker threads.

This option is valid for all junctions except for the type of `local`.

-L *percent*
Defines the hard limit for consumption of worker threads.

This option is valid for all junctions except for the type of `local`.

-n Indicates that no modification of the names of non-domain cookies are to be made. Use when client side scripts depend on the names of cookies.

By default, if a junction is listed in the JMT or if the **-j** junction option is used, WebSEAL will modify the names of non-domain cookies that are returned from the junction to prepend `AMWEBJCT!`*`junction_point`*.

This option is valid for all junctions except for the type of `local`.

-o Indicates the hostname WebSEAL connects to at the start of the handshaking process. Sends the string `<hostname>` in the SSL/TLS handshake as the Server Name Indicator (SNI) value.

For example, `-o sni=www.test.local`.

-p *port* Specifies the TCP port of the back-end third-party server. The default value is 80 for TCP junctions and 443 for SSL junctions.

This option is valid for all junctions except for the type of `local`.

-P *port* For proxy junctions that were created with the type of `tcpproxy` or `sslproxy` this option specifies the TCP port number for the HTTP proxy server. The **-P** option is required when the **-H** option is used.

This option is also valid for mutual junctions to specify the HTTPS port of the back-end third-party server.

-q *path*

Required option for back-end Windows servers. Specifies the relative path for the `query_contents` script. By default, Security Access Manager looks for the `query_contents` script in the `/cgi_bin` directory. If this directory is different or the `query_contents` file name is renamed, this option will indicates to WebSEAL the new URL to the file.

This option is valid for all junctions except for the type of `local`.

-r Inserts the incoming IP address into the HTTP header across the junction. This option is valid for all junctions except for the type of `local`.

-R Allows the request to proceed but provides the rule failure reason to the junction in an HTTP header. If the **-R** option is not used and a rule failure occurs, WebSEAL will not allow the request to proceed. This option is valid for all junctions except for the type of `local`.

-s Indicates that the junction support stateful applications. By default, junctions are not stateful. This option is valid for all junctions except for the type of `local`.

-S Specifies the name of the forms single signon configuration file. This option is valid for all junctions except for the type of `local`.

-T {*resource* | *resource_group*}

Specifies the name of the resource or resource group. This option is required only when the **-b** `gso` option is used. This option is valid for all junctions except for the type of `local`.

-u *uuid*

Specifies the Universally Unique Identifier (UUID) of a back-end server connected to WebSEAL by using a stateful junction (**-s** option). This option is valid for all junctions except for the type of `local`.

-U "*user_name*"
Specifies the WebSEAL server user name. This option requires the **-B** and **-W** options. WebSEAL uses the BA header information to authenticate to the back-end server and to provide mutual authentication over SSL. This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

-v *virtual_hostname[:HTTP-port]*
Specifies the virtual host name for the back-end server. This option supports multiple virtual hosts being served from the same Web server. Use **-v** when the back-end junction server expects a host name header different from the DNS name of the server. This option is valid for all junctions except for the type of `local`. For mutual junctions this value corresponds to the virtual host which is used for HTTP requests.

-V *virtual_hostname[:HTTPS-port]*
Specifies the virtual host name for the back-end server. This option supports multiple virtual hosts being served from the same Web server. Use **-V** when the back-end junction server expects a host name header different from the DNS name of the server. This option is only used for mutual junctions and corresponds to the virtual host which is used for HTTPS requests.

-w Indicates Microsoft Windows file system support. This option provides all of the functionality provided by the **–i** junction option but disallows requests that contain file names that might be interpreted as Windows file name aliases. This option is valid for all junctions except for the type of `local`. Local junctions prohibit URLs that contain Windows file name aliases on Windows but allow such URLs on other platforms.

-W "*password*"
Specifies the WebSEAL server password. This option requires the **-B** and **-U** options. WebSEAL uses the BA header information to authenticate to the back-end server and to provide mutual authentication over SSL. This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

-x Creates a transparent path junction.

This option is valid for all junctions except for the type of `local` .

-Y Enables the Federation Runtime single sign-on (SSO) for the junction.

Note: Before using this option, you must first configure the WebSEAL configuration file to support the Federation Runtime single sign-on over junctions.

-Z *keyfile_pwd*
Specifies the password of the keyfile used to encrypt LTPA cookie data. This option requires the **-A** and **-F** options. The **-A**, **-F**, and **-Z** options all must be used together. This option is valid for all junctions except for the type of `local`.

-h *host_name*
Required option for non-local junctions. Specifies the DNS host name or IP address of the target server. This option is valid only for non-local junctions; local junctions do not need a host name. Valid values for *host_name* include any valid IP host name. For example:

```
www.example.com
```

-t *type* Required option. Specifies the type of junction; must be one of the following types:
- `tcp`
- `tcpproxy`
- `ssl`
- `sslproxy`
- `local`

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/`*`junction_point`* object. For example, the **sec_master** administrative user is given this permission by default.

Note:

For more information about gathering statistics, see the Troubleshooting topics in the Knowledge Center..

Return codes

0 The command completed successfully. For WebSEAL **`server task`** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. However, even after the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and returns an error message.

1 The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

- The following example (entered as one line) creates a basic WebSEAL junction `/pubs` on the `default-webseald-cruz` WebSEAL server. The junction type is TCP, and the host name is `doc.tivoli.com`:

  ```
  pdadmin> server task default-webseald-cruz create -t tcp
  -h doc.tivoli.com /pubs
  ```

  Output is similar to:

  ```
  Created junction at /pubs
  ```

- The following example (entered as one line) limits worker thread consumption on a per junction basis with a soft thread limit of `60` and a hard thread limit of `80` on the `/myjunction` junction:

  ```
  pdadmin> server task default-webseald-cruz create -t tcp
  -h cruz.dallas.ibm.com -l 60 -L 80 /myjunction
  ```

See also

"server task add" on page 629
"server task delete"
"server task remove" on page 654
"server task show" on page 656

server task delete

The **`server task delete`** command deletes a WebSEAL junction point.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **delete** *junction_point*

Options

*instance_name***-webseald-***host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **`server list`** command.

  The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

  For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

  If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

junction_point
: Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name/junction_point`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0
: The command completed successfully. For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors.

  Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example deletes the junction point `/pubs` from the WebSEAL server `default-webseald-abc.ibm.com`:

```
pdadmin> server task default-webseald-abc.ibm.com delete /pubs
```

See also

"server task add" on page 629
"server task create" on page 635
"server task remove" on page 654
"server task show" on page 656

server task dynurl update

The **`server task dynurl update`** command reloads the dynamic URL configuration file.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **dynurl update**

Options

instance_name-**webseald-***host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **`server list`** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/` object. For example, the **sec_master** administrative user is given this permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example reloads the dynamic URL configuration file:

```
pdadmin> server task default-webseald-abc.ibm.com dynurl update
```

See also

None.

server task help

Lists detailed help information about a specific **server task** command.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **help** *task*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

task Lists detailed help for the specified task, such as the command syntax, the description, and the valid options.

Authorization

No special authorization required.

Return codes

0 The command completed successfully.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

- The following example displays output after requesting help for the **server task add** command at the abc.ibm.com WebSEAL server:

  ```
  pdadmin> server task default-webseald-abc.ibm.com help add
  ```

  Output is similar to:

  ```
  Command:
  add <options> <junction point>
  Description:
  Adds an additional server to a junction
  Usage:
  TCP and SSL Junction Flags
  -iServer treats URLs as case insensitive.
  -h <hostname>Target host (required flag).
  -p <port>TCP port of server.
  Default is 80 for TCP junctions
  443 for SSL junctions.
  -H <hostname>Proxy hostname.
  -P <port>Port of proxy server.
  -D <"DN">The Distinguished Name of the server
  -q <relative url> URL for query_contents script.
  -u <UUID>(stateful junctions only).
  -v <hostname>Virtual hostname for server.
  -wWin32 file system support.
  -jScripting support for junction.
  Common Flags
  <junction point>Where to create the junction
  ```

- The following example displays the output after requesting help for the **server task create** command at the abc.ibm.com WebSEAL server:

  ```
  pdadmin> server task default-webseald-abc.ibm.com help create
  ```

  Output is similar to:

  ```
  Command:
  create -t <type> <options> <junction point>
  Description:
  Creates a new junction
  Usage:
  create -t <type> <options> <junction point>

  TCP and SSL Junction Flags
  ...
  Common Flags
  -t <type>Type of junction.
  One of: tcp, tcpproxy, ssl, sslproxy, local.
  -fForce the creation: overwrite existing junction.
  -RWebSEAL will send the Boolean Rule Header to these
  junctions when a rule failure reason is provided.
  <junction point>Where to create the junction
  ```

See also

"help" on page 627

server task jmt

The **server task jmt** command clears or loads the junction mapping table data.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **jmt load**

server task *instance_name*-**webseald**-*host_name* **jmt clear**

Options

instance_name-**webseald**-*host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

  The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

  For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

  If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

jmt clear
: Clears the junction mapping table data.

jmt load
: Loads the junction mapping table data, which is located in the `jmt.conf` file. This file does not exist by default, so you must create the file and add data.

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/` object. For example, the **sec_master** administrative user is given this permission by default.

Return codes

0
: The command completed successfully.

  Note: For WebSEAL **server task** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **`pdadmin`** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example loads the junction mapping table data from the `jmt.conf` file so that WebSEAL has knowledge of the new information:

```
pdadmin> server task default-webseald-abc.ibm.com jmt load
```

Output is similar to:

```
JMT table successfully loaded.
```

See also

"server task reload" on page 653

server task list

The **`server task list`** command lists all junction points on a WebSEAL server or instance.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **list**

Options

instance_name-**webseald**-*host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **`server list`** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

Authorization

Users and groups that require access to this command must be given the **l** (list) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name/ per_junction_point`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code will be 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example lists all junction points on the default-webseald-cruz WebSEAL server:

```
pdadmin> server task default-webseald-cruz list
```

Output is similar to:

```
/
/ssljct
/tcpjct
```

See also

"server task add" on page 629
"server task create" on page 635
"server task delete" on page 642
"server task remove" on page 654
"server task show" on page 656

server task offline

The **server task offline** command places the server that is located at this junction in an offline operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **offline** [**–i** *server_uuid*] *junction_point*

Description

The **server task offline** command places the server that is located at this junction in an offline operational state. No additional requests are sent to the specified server. If a server is not specified, all servers that are at this junction are placed in an offline operational state.

Options

*instance_name***-webseald-***host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

–i *server_uuid*

Specifies the UUID of the server to place in an offline operational state. If a server is not specified, all servers that are located at this junction are placed in an offline operational state. Use the **server task show** command to determine the ID of a specific back-end server.

junction_point

Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/*junction_point* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0

The command completed successfully.

Note: For WebSEAL **server task** commands, the return code will be 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1

The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example places the backappl server located at the /pubs junction point in an offline operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Throttled
Throttled at: 2005-03-01-17:07:24
Hostname: backapp1.diamond.example.com
...
Current requests: 0
...
```

Place this server in an offline operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz offline
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

See also

"server task online"
"server task throttle" on page 661
"server task virtualhost offline" on page 675
"server task virtualhost online" on page 677
"server task virtualhost throttle" on page 683

server task online

The **server task online** command places the server that is located at this junction in an online operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **online** [**–i** *server_uuid*] *junction_point*

Description

The **server task online** command places the server that is located at this junction in an online operational state. The server now resumes normal operation. If a server is not specified, all servers that are located at this junction are placed in an online operational state.

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

–i server_uuid
Specifies the UUID of the server to place in an online operational state. If a server is not specified, all servers that are located at this junction are placed in an online operational state. Use the **server task show** command to determine the ID of a specific back-end server.

junction_point
Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name/junction_point`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example places the `backappl` server located at the `/pubs` junction point in an online operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Offline
Hostname: backappl.diamond.example.com
...
Current requests: 0
...
```

Place this server in an online operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz online
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

See also

"server task offline" on page 648
"server task throttle" on page 661
"server task virtualhost offline" on page 675
"server task virtualhost online" on page 677
"server task virtualhost throttle" on page 683

server task refresh all_sessions

The **server task refresh all_sessions** command refreshes the credential for all sessions for a specified user.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **refresh all_sessions** *user_id*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

user_id Refreshes the credential for all sessions that are associated with the specified user. Examples of user names are dlucas, sec_master, and "Mary Jones".

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/ object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code will be 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example refreshes all sessions for the test_user user:

```
pdadmin> server task default-webseald-cruz refresh all_sessions test_user
```

See also

"server task terminate session" on page 660
"server task terminate all_sessions" on page 659

server task reload

The **server task reload** command reloads the junction mapping table from the database.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **reload**

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the /WebSEAL/*host_name*-

instance_name/ object. For example, the **sec_master** administrative user has permission by default.

Return codes

0
The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example reloads the junction mapping table from the database:

```
pdadmin> server task default-webseald-abc.ibm.com reload
```

See also

“server task jmt” on page 646

server task remove

The **server task remove** command removes the specified installed WebSEAL server or instance from a WebSEAL junction point.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **remove –i** *server_uuid junction_point*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

–i *server_uuid*

Specifies the UUID of the server to be removed from the junction point. See the **`server task show`** command for details about obtaining the UUID.

junction_point

Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/`*`junction_point`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0

The command completed successfully. For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1

The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example removes the `backappl` junctioned server from the `/pubs` junction point. To determine the UUID of the server to be removed, run the **`server task show`** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
...
Hostname: backapp1.cruz.ibm.com
...
```

Remove the server from the junction (entered as one line):

```
pdadmin> server task default-webseald-cruz remove
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

See also

"server task add" on page 629
"server task create" on page 635
"server task delete" on page 642
"server task show" on page 656

server task server restart

The **server task server restart** command restarts a WebSEAL server by using the Security Access Manager server task framework.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *server_name* **server restart**

Options

server_name
: Specifies the name of the WebSEAL server to be restarted.

Authorization

Users and groups that require access to this command must be given the **s** (administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0
: The command that completed successfully. For WebSEAL **server task** commands, the return code is `0` when the command is sent to the WebSEAL server without errors.

  Note: Even if the command is successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
: The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example restarts `server03`:

```
pdadmin> server task server03 server restart
```

server task show

The **server task show** command displays detailed information about the specified WebSEAL junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **show** *junction_point*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

junction_point

Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **l** (list) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/*junction_point* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0

The command completed successfully. For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1

The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example shows information for the local root junction point / on the WebSEAL server abc.ibm.com:

```
pdadmin> server task default-webseald-abc.ibm.com show
```

Output is similar to:

```
Junction point: /
Type: Local
Junction hard limit: 0 - using global value
Junction soft limit: 0 - using global value
Active worker threads: 0
Root Directory: /opt/pdweb/www-default/docs
```

```
...
Server 1:
ID: 78a1eb8c-074a-11d9-abda-00096bda9439
...
```

See also

"server task add" on page 629
"server task create" on page 635
"server task delete" on page 642
"server task remove" on page 654

server task server sync

The **`server task server sync`** command synchronizes configuration data between two WebSEAL servers by using the Security Access Manager server task framework.

Requires authentication (administrator ID and password) to use this command.

Note: The two WebSEAL servers must be of the same type. The WebSEAL server type is either a:

- WebSEAL running on an appliance.
- WebSEAL running on a standard operating system.

Syntax

`server task` *webseal_server* **server sync** *server_name*

Options

webseal_server
Specifies the fully qualified server name of the installed WebSEAL instance. The *webseal_server* is the target.

server_name
Specifies the name of the WebSEAL server from which data is extracted. Configuration data on the host system is backed up and then synchronized with this data. The *server_name* is the source.

Authorization

Users and groups that require access to this command must be given the **s** (administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`* object. The **sec_master** administrative user has permission by default.

Return codes

0
The command that completed successfully. For WebSEAL **`server task`** commands, the return code is `0` when the command is sent to the WebSEAL server without errors.

Note: Even if the command is successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **`pdadmin`** command

provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example synchronizes configuration data with server master-webseald-abc.ibm.com:

```
pdadmin> server task default-webseald-abc.ibm.com server sync
master-webseald-abc.ibm.com
```

server task terminate all_sessions

The **server task terminate all_sessions** command terminates all user sessions for a specific user.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **terminate all_sessions** *user_id*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

user_id Specifies the name of the user. Examples of user names are dlucas, sec_master, and "Mary Jones".

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/ object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example terminates all sessions for the dlucas user on the default-webseald-cruz WebSEAL server:

```
pdadmin> server task default-webseald-cruz terminate all_sessions dlucas
```

See also

"server task terminate session"
"server task refresh all_sessions" on page 652

server task terminate session

The **server task terminate session** command terminates a user session using a session ID.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **terminate session** *session_id*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

session_id
Specifies the ID of a user session.

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/` object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **`pdadmin`** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example (entered as one line) terminates a specific session on the `default-webseald-cruz` WebSEAL server:

```
pdadmin> server task default-webseald-cruz terminate
session 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
```

See also

"server task refresh all_sessions" on page 652
"server task terminate all_sessions" on page 659

server task throttle

The **`server task throttle`** command places the server that is located at this junction in a throttled operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **throttle** [**–i** *server_uuid*] *junction_point*

Description

The **`server task throttle`** command places the server that is located at this junction in a throttled operational state. Only requests from users who have created a session with WebSEAL prior to the invocation of this command continue to have their requests processed by the specified server. If a server is not specified, all servers that are located at this junction are placed in a throttled operational state.

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

–i server_uuid
Specifies the UUID of the server to throttle. If a server is not specified, all servers that are located at this junction are placed in a throttled operational state. Use the **server task show** command to determine the ID of a specific back-end server.

junction_point
Specifies the name of the directory in the WebSEAL protected object space where the document space of the back-end server is mounted.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/*junction_point* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example places the backappl server located at the /pubs junction point in a throttled operational state. To determine the UUID of this junctioned server, run the **server task show** command:

```
pdadmin> server task default-webseald-cruz show /pubs
```

Output is similar to:

```
Junction point: /pubs
...
Server 1:
ID: 6fc3187a-ea1c-11d7-8f4e-09267e38aa77
Server State: running
Operational State: Online
Hostname: backapp1.diamond.example.com
...
Current requests: 0
...
```

Place this server in a throttled operational state (entered as one line):

```
pdadmin> server task default-webseald-cruz throttle
-i 6fc3187a-ea1c-11d7-8f4e-09267e38aa77 /pubs
```

See also

"server task offline" on page 648
"server task online" on page 650
"server task virtualhost offline" on page 675
"server task virtualhost online" on page 677
"server task virtualhost throttle" on page 683

server task virtualhost add

The **server task virtualhost add** command adds an additional installed WebSEAL server or instance to an existing virtual host junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost add –h** *host_name* [*options*] *vhost_label*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

options Specifies the options that you can use with the **server task virtualhost add** command. These options include:

–D "*dn*"

Specifies the distinguished name of the back-end server certificate. This value, matched with the actual certificate DN enhances authentication and provides mutual authentication over SSL. For example, the certificate for `www.example.com` might have a DN of

```
"CN=WWW.EXAMPLE.COM,OU=Software,O=example.com\, Inc,L=Austin,
ST=Texas,C=US"
```

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–H *host_name*

Specifies the DNS host name or IP address of the proxy server.

Valid values for *host_name* include any valid IP host name. For example:

```
proxy.www.example.com
```

This option is used for junctions that were created with the type of `tcpproxy` or `sslproxy`.

–i Indicates that the WebSEAL server does not treat URLs as case-sensitive.

This option is used for junctions that were created with the type of `tcp` or `ssl`.

–p *port*

Specifies the TCP port of the back-end server. The default value is `80` for TCP junctions and `443` for SSL junctions. This option is used for junctions that were created with the type of `tcp` or `ssl`.

–P *port*

Specifies the TCP port of the proxy server. The default value is `7138`.

For *port*, use any valid port number. A valid port number is any positive number that is allowed by TCP/IP and that is not currently being used by another application. Use the default port number value, or use a port number that is greater then 1000 that is currently not being used.

This option is used for junctions that were created with the type of `tcpproxy` or `sslproxy`.

–q *path*

Required option for back-end Windows virtual hosts. Specifies the relative path for the **query_contents** script. By default, Security Access Manager looks for this script in the `/cgi_bin` subdirectory. If this directory is different or the **query_contents** file is renamed, use this option to indicate to WebSEAL the new URL to the file.

This option is valid for all junction types except `localtcp` and `localssl`.

–u *uuid*

Specifies the UUID of this back-end server when connected to WebSEAL over a stateful junction that was using the **–s** option. This option is used for junctions that were created with the type of `tcp` or `ssl`.

–w Indicates Microsoft Windows file system support.

This option is used for junctions that were created with the type of tcp or ssl.

vhost_label
Specifies the label name of the virtual host junction.

–h *host_name*
Required option. Specifies the DNS host name or IP address of the target server. Valid values for *host_name* include any valid IP host name. For example:

```
www.example.com
```

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/@*vhost_label* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully. For WebSEAL **server task** commands, the return code will be 0 when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example (entered as one line) adds an additional server with host name xyz.ibm.com to an existing virtual host junction with the label support-vhost-http, located on the WebSEAL server abc.ibm.com:

```
pdadmin> server task default-webseald-abc.ibm.com virtualhost add
-h xyz.ibm.com support-vhost-http
```

See also

"server task virtualhost create"
"server task virtualhost delete" on page 672
"server task virtualhost list" on page 674
"server task virtualhost remove" on page 679
"server task virtualhost show" on page 681

server task virtualhost create

The **server task virtualhost create** command creates a virtual host junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

For local junctions:

`server task` *instance_name*-**webseald**-*host_name* **virtualhost create –t** *type* **–v** *virtual_host_name* [*options*] *vhost_label*

For non-local junctions:

`server task` *instance_name*-**webseald**-*host_name* **virtualhost create –t** *type* **–h** *host_name* [*options*] *vhost_label*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify the full server name in the exact format as displayed in the output of the **`server list`** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

options Specifies the options that you can use with the **`server task virtualhost create`** command. These options include:

–A Enables a virtual host junction to support the lightweight third-party authentication mechanism (LTPA). This option requires the **–F** and **–Z** options. The **–A**, **–F**, and **–Z** options all must be used together.

This option is valid for all junction types except `localtcp` and `localssl`.

-2 You can use this option in conjunction with the **-A** option to specify that LTPA version 2 cookies (LtpaToken2) are used. The **-A** option without the **-2** option specifies that LTPA version 1 cookies (LtpaToken) are used.

–b *BA_value*

Defines how the WebSEAL server passes client identity information in HTTP basic authentication (BA) headers to the back-end virtual host, which is one of the following values:

- `filter`
- `ignore`
- `supply`
- `gso`

This option is valid for all junction types except `localtcp` and `localssl`.

The default value is `filter`.

–B Indicates that WebSEAL uses the BA header information to authenticate to the back-end virtual host and to provide mutual authentication over SSL. This option requires the **–U** and **–W** options.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–c *header_type*

Inserts the Security Access Manager client identity in HTTP headers across the virtual host junction. The *header_type* argument can include any combination of the following Security Access Manager HTTP header types:

- `{iv-user|iv-user-l}`
- `iv-groups`
- `iv-creds`
- `all`

The header types must be comma separated, and cannot have a spaces between the types. For example: `-c iv_user,iv_groups`

Specifying **–c** `all` is the same as specifying **–c** `iv-user,iv-groups,iv-creds`.

This option is valid for all junction types except `localtcp` and `localssl`.

–C

Supports mutual authentication by enabling the front-end WebSEAL server to pass its identity information to the back-end WebSEAL server in a Basic Authentication (BA) header. Additionally, the **–C** option enables single signon functionality provided by the **–c** option.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–D "*dn*"

Specifies the distinguished name of the back-end server certificate. This value, matched with the actual certificate DN enhances authentication and provides mutual authentication over SSL. For example, the certificate for `www.example.com` might have a DN of

```
"CN=WWW.EXAMPLE.COM,OU=Software,O=example.com\, Inc,L=Austin,
ST=Texas,C=US"
```

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–e *encoding_type*

Specifies the encoding to use when generating HTTP headers for virtual host junctions. This encoding applies to headers that are generated with both the **–c** junction option and tag-value. Possible values for encoding are as follows:

utf8_bin

WebSEAL sends the headers in UTF-8.

utf8_uri

WebSEAL sends the headers in UTF-8 but URI also encodes them. This behavior is the default behavior.

lcp_bin

WebSEAL sends the headers in the local code page of the WebSEAL server.

lcp_uri
WebSEAL sends the headers in the local code page of the WebSEAL server, but URI also encodes them.

This option is valid for all junction types except `localtcp` and `localssl`.

–f
Forces the replacement (overwrite) of an existing virtual host junction.

This option is used for junctions that were created with the any junction type.

–F "*keyfile*"

Specifies the name of the keyfile used to encrypt LTPA cookie data.

The **–F** option requires **–A** and **–Z** options. The **–A**, **–F**, and **–Z** options all must be used together.

This option is valid for all junction types except `localtcp` and `localssl`.

–g *vhost_label*
The **–g** option causes a second additional virtual host junction to share the same protected object space as the initial virtual host junction.

This option is appropriate for junction pairs only (two junctions using complementary protocols). The option does not support the association of more than two junctions.

–H *host_name*
Specifies the DNS host name or IP address of the proxy server. The **–P** option also supports proxy server junctions. Valid values for *host_name* include any valid IP host name. For example:

```
proxy.www.example.com
```

This option is valid only with junctions that were created with the type of `tcpproxy` or `sslproxy`.

–i
Indicates that the WebSEAL junction does not treat URLs as case-sensitive. To correctly authorize requests for junctions that are not case-sensitive, WebSEAL does the authorization check on a lowercase version of the URL. For example, a Web server that is running on a Windows operating system treats requests for `INDEX.HTM` and `index.htm` as requests for the same file.

Junctions to such a Web server should be created with the **–i** or **–w** option. ACLs or POPs that are attached to objects beneath the junction point should use the lower case object name. An ACL attached to `/junction/index.htm` will apply to all of the following requests if the **–i** or **–w** option is used:

```
/junction/INDEX.HTM
/junction/index.htm
/junction/InDeX.HtM
```

This option is valid for all junction except for the type of `localtcp` and `localssl`. Local junctions are not case-sensitive only on Win32 platforms; all other platforms are case-sensitive.

–k Sends WebSEAL session cookies to the back-end virtual host. By default, cookies are removed from requests that are sent to the server.

This option is valid for all junction types except `localtcp` and `localssl`.

–K "*key_label*"
Specifies the key label of the client-side certificate that WebSEAL should present to the back-end server. Use of this option allows the virtual host to authenticate the WebSEAL server using client certificates.

This option is valid only with junctions that were created with the type of `ssl` and `sslproxy`.

–l *percent*
Defines the soft limit for consumption of worker threads.

This option is valid for all junction types except `localtcp` and `localssl`.

–L *percent*
Defines the hard limit for consumption of worker threads.

This option is valid for all junction types except `localtcp` and `localssl`.

–p *port*
Specifies the TCP port of the back-end third-party server. The default value is `80` for TCP junctions and `443` for SSL junctions.

This option is valid for all junction types except `localtcp` and `localssl`.

–P *port*
Specifies the TCP port number for the HTTP proxy server. The **–P** option is required when the **–H** option is used.

This option is valid only with junctions that were created with the type of `tcpproxy` or `sslproxy`.

–q *path*
Required option for back-end Windows virtual hosts. Specifies the relative path for the `query_contents` script. By default, Security Access Manager looks for the `query_contents` script in the `/cgi_bin` directory. If this directory is different or the `query_contents` file name is renamed, this option will indicate to WebSEAL the new URL to the file.

This option is valid for all junction types except `localtcp` and `localssl`.

–r Inserts the incoming IP address into the HTTP header across the junction.

This option is valid for all junction types except `localtcp` and `localssl`.

–R Allows the request to proceed but provides the rule failure reason to the junction in an HTTP header. If the **–R** option is not used and a rule failure occurs, WebSEAL will not allow the request to proceed.

This option is valid for all junction types except `localtcp` and `localssl`.

–s Indicates that the virtual host junction support stateful applications. By default, virtual host junctions are not stateful.

This option is valid for all junction types except `localtcp` and `localssl`.

–S Indicates the name of the forms single signon configuration file.

This option is valid for all junction types except `localtcp` and `localssl`.

–T {*resource* | *resource_group*}
Specifies the name of the GSO resource or resource group. This option is required only when the **–b** `gso` option is used.

This option is valid for all junction types except `localtcp` and `localssl`.

–u *uuid*
Specifies the Universally Unique Identifier (UUID) of a back-end server connected to WebSEAL by using a stateful virtual host junction (**–s** option).

This option is valid for all junction types except `localtcp` and `localssl`.

–U "*user_name*"
Specifies the WebSEAL server user name. This option requires the **–B** and **–W** options. WebSEAL uses the BA header information to authenticate to the back-end virtual host and to provide mutual authentication over SSL.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–v *vhost_name*[**:***port*]

WebSEAL selects a virtual host junction to process a request if the request's HTTP **Host** header matches the virtual host name and port number specified by the **-v** option.

The **-v** option is also used to specify the value of the **Host** header of the request sent to the back-end server.

The port number is required if the virtual host uses a non-standard port for the protocol. Standard port for TCP is 80; standard port for SSL is 443.

If **–v** is not specified for `tcp`, `ssl`, `tcpproxy`, and `sslproxy` type junctions, the junction is selected from the information contained in the **–h** *host* and **–p** *port* options.

The **–v** option is required for `localtcp` and `localssl` type junctions.

–w Indicates Microsoft Windows file system support. This option provides all of the functionality provided by the **–i** junction option but disallows requests that contain file names that might be interpreted as Windows file name aliases.

This option is valid for all junction types except `localtcp` and `localssl`. Local junctions prohibit URLs that contain Windows file name aliases on Win64 but allow such URLs on other platforms.

–W "*password*"
Specifies the WebSEAL server password. This option requires the **–B** and **–U** options. WebSEAL uses the BA header information to authenticate to the back-end virtual host and to provide mutual authentication over SSL.

This option is valid only with junctions that were created with the type of `ssl` or `sslproxy`.

–Y Enables the Federation Runtime single sign-on (SSO) for the junction.

Note: Before using this option, you must first configure the WebSEAL configuration file to support the Federation Runtime single sign-on over junctions.

–z *replica_set*
Specifies the replica set, as follows:

For distributed session cache environments:
The replica set that sessions on the virtual host junction are managed under and provides the ability to group or separate login sessions among multiple virtual hosts.

For environments that do not use the distributed session cache:
The replica set that sessions on the virtual host junction are managed under and controls the partitioning of the WebSEAL session cache so the virtual host can be part of the same replica set as any standard junction that is assigned to that same replica set through the **`standard-junction-replica-set`** entry of the **`[session]`** stanza.

–Z *keyfile_pwd*
Specifies the password of the keyfile used to encrypt LTPA cookie data. This option requires the **–A** and **–F** options. The **–A**, **–F**, and **–Z** options all must be used together.

This option is valid for all junction types except `localtcp` and `localssl`.

vhost_label
Specifies the label name of the virtual host junction.

–h *host_name*
Required option for non-local junctions. Specifies the DNS host name or IP address of the target server. This option is valid only for non-local junctions; local junctions do not need a host name. Valid values for *host_name* include any valid IP host name. For example:

```
www.example.com
```

–t *type* Required option. Specifies the type of virtual host junction. This option is required and must be one of the following types:
- `tcp`
- `tcpproxy`
- `ssl`
- `sslproxy`

- `localtcp`
- `localssl`

Authorization

Users and groups that require access to this command must be given the **s** (server administration) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/@`*`vhost_label`* object. For example, the **sec_master** administrative user has permission by default.

Note: For more information about gathering statistics, see the Auditing topics in the IBM Knowledge Center.

Return codes

0 The command completed successfully. For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example (entered as one line) creates an SSL type virtual host junction with the `vhost-xy-https` label. This junction serves the virtual host `x.y.com` located on the junctioned server `cruz1.ibm.com`. WebSEAL responds to the `Host: x.y.com` header in SSL (HTTPS) requests by forwarding the requests across this virtual host junction:

```
pdadmin> server task default-webseald-abc.ibm.com virtualhost create
-t ssl -h cruz1.ibm.com -v x.y.com vhost-xy-https
```

See also

"server task virtualhost add" on page 663
"server task virtualhost delete"
"server task virtualhost list" on page 674
"server task virtualhost remove" on page 679
"server task virtualhost show" on page 681

server task virtualhost delete

The **`server task virtualhost delete`** command deletes a virtual host junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **virtualhost delete** *vhost_label*

Description

The **server list virtualhost delete** command deletes a virtual host junction. A virtual host junction cannot be deleted if a second virtual host junction refers to it through the **–g** option. An error message is returned at such an attempt.

Options

*instance_name***-webseald-***host_name*
: Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

  The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

  For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

  If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

vhost_label
: Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the /WebSEAL/*host_name-instance_name*/@*vhost_label* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully. For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example (entered as one line) deletes the virtual host junction support-vhost-https from the WebSEAL server abc.ibm.com:

```
pdadmin> server task default-webseald-abc.ibm.com virtualhost delete
support-vhost-https
```

See also

"server task virtualhost add" on page 663
"server task virtualhost create" on page 665
"server task virtualhost list"
"server task virtualhost remove" on page 679
"server task virtualhost show" on page 681

server task virtualhost list

The **server task virtualhost list** command lists all configured virtual host junctions by label name.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost list**

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

Authorization

Users and groups that require access to this command must be given the **l** (list) permission in the ACL that governs the /WebSEAL/*host_name*-*instance_name*/@*per_vhost_label* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example lists the label names of all virtual host junctions configured on the `abc.ibm.com` WebSEAL server:

```
pdadmin> server task default-webseald-abc.ibm.com virtualhost list
```

Output is similar to:

```
pubs-vhost-http
sales-vhost-https
support-vhost-http
```

See also

"server task virtualhost add" on page 663
"server task virtualhost create" on page 665
"server task virtualhost delete" on page 672
"server task virtualhost remove" on page 679
"server task virtualhost show" on page 681

server task virtualhost offline

The **`server task virtualhost offline`** command places the server that is located at this virtual host junction in an offline operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

`server task` *instance_name*-**webseald**-*host_name* **virtualhost offline** [**–i** *server_uuid*] *vhost_label*

Description

The **`server task virtualhost offline`** command places the server that is located at this virtual host junction in an offline operational state. No additional requests are sent to the specified server. If a server is not specified, all servers that are located at this virtual host junction are placed in an offline operational state.

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **`server list`** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

–i server_uuid
Specifies the UUID of the server to place in an offline operational state. If a server is not specified, all servers that are located at this virtual host junction are placed in an offline operational state. Use the **server task virtualhost show** command to determine the ID of a specific back-end server.

vhost_label
Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/@`*`vhost_label`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

In the following example, the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`.

There is a requirement to place the `int3.ibm.com` server in an offline operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command (entered as one line):

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
```

```
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Throttled
Throttled at: 2005-03-01-17:07:24
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in an offline operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost offline
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

See also

"server task offline" on page 648
"server task online" on page 650
"server task throttle" on page 661
"server task virtualhost online"
"server task virtualhost throttle" on page 683

server task virtualhost online

The **server task virtualhost online** command places the server that is located at this virtual host junction in an online operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost online** [**–i** *server_uuid*] *vhost_label*

Description

The **server task virtualhost online** command places the server that is located at this virtual host junction in an online operational state. The server now resumes normal operation. If a server is not specified, all servers that are located at this virtual host junction are placed in an online operational state.

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

–i server_uuid
UUID of the server to place in an online operational state. If a server is not specified, all servers that are located at this virtual host junction are placed in an online operational state. Use the **server task virtualhost show** command to determine the ID of a specific back-end server.

vhost_label
Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name`*`-`*`instance_name`*`/@`*`vhost_label`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0
The command completed successfully.

Note: For WebSEAL **server task** commands, the return code will be `0` when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

In the following example, the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`.

There is a requirement to place the `int3.ibm.com` server in an online operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command (entered as one line):

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Offline
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in an online operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost online
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

See also

"server task offline" on page 648
"server task online" on page 650
"server task throttle" on page 661
"server task virtualhost offline" on page 675
"server task virtualhost throttle" on page 683

server task virtualhost remove

The **server task virtualhost remove** command removes the specified server from a virtual host junction.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost remove –i** *server_uuid vhost_label*

Options

instance_name-**webseald**-*host_name*

Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

–i *server_uuid*
Specifies the UUID of the server to be removed from the virtual host junction. For this command, the **–i** option, normally used to treat URLs as case-sensitive,operates like the **–u** option. See the **`server task show`** command for details about obtaining the UUID.

vhost_label
Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the `/WebSEAL/`*`host_name-instance_name`*`/@`*`vhost_label`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully. For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example removes the junctioned server `int4.ibm.com` from the virtual host junction `support-vhost-https`. To determine the UUID of the server to be removed, run the **`server task virtualhost show`** command (entered as one line):

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
Junction hard limit: 0 - using global value
Junction soft limit: 0 - using global value
Active worker threads: 0
Basic authentication mode: filter
Forms based SSO: disabled
Authentication HTTP header: do not insert
Remote Address HTTP header: do not insert
Stateful junction: no
Boolean Rule Header: no
Delegation support: no
```

```
Mutually authenticated: no
Insert WebSphere LTPA cookies: no
Insert WebSEAL session cookies: no
Request Encoding: UTF-8, URI Encoded
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Total requests: 1
Server 2:
ID: xycecc77-19ve-81y5-4h0a-90267hj5nn57
Server State: running
Hostname: int4.ibm.com
Port: 444
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Total requests: 1
```

Remove the server from the virtual host junction (entered as one line):

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost remove -i xycecc77-19ve-81y5-4h0a-90267hj5nn57 support-vhost-https
```

See also

"server task virtualhost add" on page 663
"server task virtualhost create" on page 665
"server task virtualhost delete" on page 672
"server task virtualhost list" on page 674
"server task virtualhost show"

server task virtualhost show

The **server task virtualhost show** command displays information about the specified virtual host junction. The virtual host junction must exist, or an error is displayed.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost show** *vhost_label*

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The `webseald` designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is `default`, and host machine name where the WebSEAL server is installed is `abc.ibm.com`, the full WebSEAL server name is `default-webseald-abc.ibm.com`.

If an additional WebSEAL instance is configured and named `web2`, the full WebSEAL server name is `web2-webseald-abc.ibm.com`.

vhost_label
Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **l** (list) permission in the ACL that governs the `/WebSEAL/`*`host_name`*`-`*`instance_name`*`/@`*`vhost_label`* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0
The command completed successfully. For WebSEAL **`server task`** commands, the return code becomes `0` when the command is sent to the WebSEAL server without errors.

Note: Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1
The command failed. When a command fails, the **`pdadmin`** command provides a description of the error and an error status code in hexadecimal format (for example, `0x14c012f2`). See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

The following example (entered as one line) shows information for the virtual host junction with the label `support-vhost-https`, configured on the WebSEAL server `abc.ibm.com`, that supports the virtual host `support.ibm.com`, located on the back-end junctioned server `int3.ibm.com`:

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
Junction hard limit: 0 - using global value
Junction soft limit: 0 - using global value
Active worker threads: 0
Basic authentication mode: filter
Forms based SSO: disabled
Authentication HTTP header: do not insert
Remote Address HTTP header: do not insert
Stateful junction: no
Boolean Rule Header: no
```

```
Delegation support: no
Mutually authenticated: no
Insert WebSphere LTPA cookies: no
Insert WebSEAL session cookies: no
Request Encoding: UTF-8, URI Encoded
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Total requests: 1
```

See also

"server task virtualhost add" on page 663
"server task virtualhost create" on page 665
"server task virtualhost delete" on page 672
"server task virtualhost list" on page 674
"server task virtualhost remove" on page 679

server task virtualhost throttle

The **server task virtualhost throttle** command places the server that is located at this virtual host junction in a throttled operational state.

Requires authentication (administrator ID and password) to use this command.

Syntax

server task *instance_name*-**webseald**-*host_name* **virtualhost throttle** [**–i** *server_uuid*] *vhost_label*

Description

The **server task virtualhost throttle** command places the server that is located at this virtual host junction in a throttled operational state. Only requests from users who have created a session with WebSEAL prior to the invocation of this command continue to have their requests processed by the specified server. If a server is not specified, all servers that are located at this virtual host junction are placed in a throttled operational state..

Options

instance_name-**webseald**-*host_name*
Specifies the full server name of the installed WebSEAL instance. You must specify this full server name in the exact format as displayed in the output of the **server list** command.

The *instance_name* specifies the configured name of the WebSEAL instance. The webseald designation indicates that the WebSEAL service performs the command task. The *host_name* is the name of the physical machine where the WebSEAL server is installed.

For example, if the configured name of a single WebSEAL instance is default, and host machine name where the WebSEAL server is installed is abc.ibm.com, the full WebSEAL server name is default-webseald-abc.ibm.com.

If an additional WebSEAL instance is configured and named web2, the full WebSEAL server name is web2-webseald-abc.ibm.com.

–i server_uuid
Specifies the UUID of the server to throttle. If a server is not specified, all servers that are at this virtual host junction are placed in a throttled operational state. Use the **server task virtualhost show** command to determine the ID of a specific back-end server.

vhost_label
Specifies the label name of the virtual host junction.

Authorization

Users and groups that require access to this command must be given the **c** (control) permission in the ACL that governs the /WebSEAL/*host_name*-*instance_name*/@*vhost_label* object. For example, the **sec_master** administrative user has permission by default.

Return codes

0 The command completed successfully.

Note: For WebSEAL **server task** commands, the return code becomes 0 when the command is sent to the WebSEAL server without errors. Even if the command was successfully sent, the WebSEAL server might not be able to successfully complete the command and can return an error message.

1 The command failed. When a command fails, the **pdadmin** command provides a description of the error and an error status code in hexadecimal format (for example, 0x14c012f2).

See "Error messages" in the IBM Knowledge Center which provides a list of the Security Access Manager error messages by decimal or hexadecimal codes.

Examples

In the following example, the virtual host junction with the label support-vhost-https, configured on the WebSEAL server abc.ibm.com, supports the virtual host support.ibm.com, located on the back-end junctioned server int3.ibm.com.

There is a requirement to place the int3.ibm.com server in a throttled operational state. To determine the UUID of this junctioned server, run the **server task virtualhost show** command (entered as one line):

```
pdadmin> server task default-webseald-abc.ibm.com
virtualhost show support-vhost-https
```

Output is similar to:

```
Virtual Host label: support-vhost-https
Type: SSL
...
Virtual hostname: support.ibm.com
Alias: ibm.com
Alias: support
Virtual Host junction protocol partner: support-vhost-http
Server 1:
ID: bacecc66-13ce-11d8-8f0a-09267ea5aa77
Server State: running
Operational State: Online
Hostname: int3.ibm.com
Port: 443
Server DN:
Query_contents URL: /cgi-bin/query_contents
Query-contents: unknown
Case insensitive URLs: no
Allow Windows-style URLs: yes
Current requests: 0
Total requests: 1
```

Place this server in a throttled operational state using the following command (entered as one line):

```
pdadmin> server task default-webseald-cruz virtualhost throttle
-i bacecc66-13ce-11d8-8f0a-09267ea5aa77 support-vhost-https
```

See also

"server task throttle" on page 661
"server task offline" on page 648
"server task online" on page 650
"server task virtualhost offline" on page 675
"server task virtualhost online" on page 677

Chapter 14. Rate limiting

Web reverse proxies allow incoming requests to be limited based on a user-defined set of criteria and policy.

Rate limiting achieves the following protections:

- Brute force attacks on sensitive information such as passwords or PINs
- Denial of Service attacks on a server or the Web Reverse Proxy

Rate limiting is performed by taking the incoming request and identifying the parts of the request that makes it unique to a client. Information such as the IP address that made the connection, a session cookie, other header information or the URL and HTTP method that were used can all be included to identify a client. When duplicate requests come in they fill up a counter; when full causes the reverse proxy to return an error to the client or drop the incoming connection.

Rate limiting can be deployed with two types of end users in consideration. The **malicious end user** who tries to cause damage to the service or steal a credential with brute force. A **valid user** who needs to be stopped from making too many requests in order to stop service degradation or overloading a back end server. There are two approaches to handle a malicious user:

- The first approach is to handle them in the most efficient manner by closing the connection without performing any further request processing.
- The second approach is to mislead the malicious user by returning false information, making it unclear to the user, if the operation was successful or even processed.

However, for a valid end user who is intended to use the service, a page that provides more information is more suitable.

The web reverse proxy that performs the rate limiting might not be the perimeter device in a connection by a client, and thus the incoming IP is not from the end user. In this instance, the device that is at the network perimeter includes the client address information in a header such as **`X-Forwarded-For`**, which can then be included by the reverse proxy when identifying the request.

Rate limiting is configured by authoring a policy file. This policy file contains the following items:

- Requests that the policy applies to. The request is identified by the HTTP method and URL.
- The information in the request that identifies it. This includes; cookies, headers, query string parameters, the client IP address and credential attributes if they are available.
- The threshold for how many identical requests a client can make before they need to be rate limited and how long a client remains rate limited.
- How to respond when a client is rate limited.

Once a rate limiting policy is created, the reverse proxy configuration file must be updated to attach it to an instance. After the configuration file is attached to the instance, restart the instance. Once a policy is added to the reverse proxy configuration file and the instance is reset, any changes to the policy file is dynamically reloaded once they are deployed, without any additional restart of the reverse proxy.

Rate limiting can occur in two places during the reverse proxy processing flow. When the rate limiting policy does not include credential attributes as a value to use in the request then rate limiting occurs very early in processing a request. This includes authentication and authorization. When the credential attributes are configured, then the rate limiting occurs once the users session is verified. This is after HTTP transformations (when the transformation is not attached with POP) and authentication, but prior to authorization checks. If rate limiting is applied early in the processing and the URL is re-written, then rate limiting is not applied again.

On every request all rate limiting polices are applied, such that if a request satisfies multiple policies they are all applied until one results in a reaction.

Rate limiting policies can be layered. An example of applying multiple policies is how you can rate limit a user account on a device. This can be done with two rate limiting policies. The first policy limits a user/s ability to log in based on client IP, and the POST to the login form. Then you can use a second policy to rate limit the use of a **`PDS-SESSION-ID`** cookie, with the reaction of invoking the logout page. This way, when a user abuses the session, they are logged out and must log in again. With the first policy in place, logging in is also a limited operation.

Trace for rate limiting is enabled by using the trace string `pdweb.http.ratelimit`.

Rate Limiting Policy Files

Configure rate limiting with a policy file.

The policy file consists of four key portions.

The first portion is matching criteria. If a request matches the criteria the rest of the policy is applied. The two parts of matching criteria are HTTP method and the request URL. The URL pattern supports wild cards. The HTTP method is a list of methods to match on, or the value of "*" to indicate that all HTTP methods must match. A single rate limiting policy can contain multiple matching criteria. For example, multiple URLs and methods. Each matching criteria will correspond to a single rate limiting bucket. In other words, the access rate for each matching criteria is treated separately.

All matching on URL and method is case insensitive.

An example of limiting attempts to log into the reverse proxy with the login form is:

```
resources:
  - url: /pkmslogin.form
    method:
      - POST
```

`method: POST` If you want to match this configuration to every request the following can be specified:

```
resources:
  - url: "*"
    method: "*"
```

An example to match all POST and GET requests regardless of URL:

```
resources:
  - url: "*"
    method:
      - GET
      - POST
```

An example to apply this policy to all requests to a given application path:

```
resources:
  - url: "/jct_a"
    method: "*"
```

Once a request is matched, the configured values of cookies, headers, query string parameters, client IP, and credential attributes are included.

These criteria support wild cards for the value. When a match occurs the matched value is used to identify the request not the pattern. All matching is case insensitive.

When these values are matched and the request does not contain a specified header, cookie, query string parameter, or credential attribute, the request is not rate limited.

An example of limiting incoming requests based on the Basic Authentication credentials:

```
header:
   Authorization: "Basic *"
```

An example to limit bearer token usage

```
header:
   Authorization: "Bearer *"
```

An example of limiting a session involves rate limiting on the `PD-S-SESSION-ID` cookie:

```
cookie:
   PD-S-SESSION-ID: "*"
```

An example of including a query string. For a request like `/junction?resource=123`, limit with:

```
query:
   resource: 123
```

An example of limiting any resource:

```
query:
   resource: "*"
```

An example of limiting on a credential attribute, to limit access of users based on username:

```
credential:
   AZN_CRED_PRINCIPAL_NAME: "*"
```

An example of limiting access is to limit based on an OAuth client ID when you are using OAuth. See OAuth Authentication

```
credential:
    oauth_token_client_id: "*"
```

There's a true and false flag to set to include the client IP in the request:

```
ip: true
```

The matching information identified from the request, such as URL, Method, IP, cookies, headers, and query parameters are then built into a consistent lookup key.

A counter is kept for this value. This counter has two properties, the maximum value and how often the count is set down to zero. These values are controlled by two configuration properties; a capacity which indicates the number of requests that are allowed until rate limiting occurs and an interval which is the number of seconds that must pass until the capacity is reset.

To allow 100 requests per minute, set the following:

```
capacity: 100
interval: 60
```

The final part of the configuration is the reaction method. There are three ways to react:

CLOSE
Closes the connection without any information written to the client.

TEMPLATE
Writing back the rate limiting template with the status code 429.

Specify URL
A URL can be specified. This URL rewrites the request URL to the defined value. This is used to route a rate limited request to a dummy resource. This allows the concept of hosting a dummy resource which appears to produce the same functionality as the actual resource, only never actually making any change. The intent of this is to mislead a malicious client into thinking they are performing numerous operations while actually not having any negative impact. This can also be used to log a user out. For example, you can direct them to `/pkmslogout` to terminate their session.

Note: Providing a reaction configuration is optional, if nothing is specified **TEMPLATE** behavior is the default.

Close the connection with:

```
reaction: CLOSE
```

Return the template with configured with:

```
reaction: TEMPLATE
```

Rewrite the URL with:

```
reaction: /dummy-login
```

Rate limiting policy files support comments by including a leading # character. Here is a full and commented policy file for rate limiting bearer token usage per client IP to 10 times / second:

```
resources:
  - url: "*"
    method:
      - "*"

# Limit based on the authorization header, with a leading "Bearer " prefix:
header:
  Authorization: "Bearer *"
```

```
# Tokens can be used 10 times in a second
capacity: 10
interval: 1

# Include the IP of the client
ip: true

# Return the template if a client is rate-limited.
reaction: TEMPLATE
```

When performing rate limiting with a reverse proxy, some information about the number of requests made for a client and policy must be stored. This information is stored in a cache which has a size limit. When this limit is exceeded the oldest entry is ejected. It is important to ensure that you set the configuration entry **max-ratelimiting-buckets** in the **[server]** stanza to a suitably high number so a malicious client cannot saturate the cache. For example, a base rate limiting policy on all URLs, methods and IP as a matching criteria can be used to ensure a user is rate limited before they can saturate the cache.

Limit the number of POST requests to the reverse proxy login form

The following rate limiting configuration allows an administrator to limit the number of POST requests made to the reverse proxy login form by any one IP address.

It allows a user to try logging for 5 times in a minute. If the user becomes rate limited then a 429 is returned.

This policy can be enhanced by creating a resource that mimics the login-failed. HTML template and rewriting the request to this dummy resource. This stops a brute force attacker from knowing they are rate limited and save resources in the way of reduced LDAP look ups.

```
#=========================
# Rate limiting pkmslogin
#=========================
# This policy limits the number of times a given IP address can be used to
# attempt to log into the reverse proxy in 60 seconds.
#
# If the reverse proxy is behind a load balancer, instead of the IP Address, a
# header down streamed from the load balancer can be used to downstream the
# client IP address.

#=========================
# Matching criteria
#=========================
# The login form.
# The same principals could be applied to a EAI form too.
resources:
  - url: /pkmslogin.form
    method:
      - POST

#=========================
# Tokenizing Criteria
#=========================
# Use the client IP to identify them
ip: true

# If a load balancer is involved instead use:
#header:
#  x-forwarded-for: "*"
```

```
#=========================
# Capacity Criteria
#=========================
# The client is allowed to try 20 times in a minute.  A more than reasonable
# number for a regular user.

capacity: 20
interval: 60

#=========================
# Reaction
#=========================
# Return a 429.
reaction: TEMPLATE

# Or if there is a dummy login page available:
#reaction: /dummy-login-url
```

Attach rate limiting policy to a reverse proxy

Once a rate limiting policy is written, it must be attached to a web reverse proxy.

This policy attachment is done with the reverse proxy configuration file under the stanza **[rate-limiting]**.

```
[rate-limiting]

policy = LimitBearerToken.yaml
policy = LimitLoginPolicy.yaml
```

Once rate limiting policy is attached to a reverse proxy, changes can be made to the policy and applied to the reverse proxy without restarting the instance. Changes are dynamically reloaded by the instance when pending changes for rate limiting are deployed.

For more information on the **[rate-limiting]** stanza, see **[rate-limiting]** stanza.

Rate limiting with a load balancer

When rate limiting is performed by the reverse proxy, and when there is a load balancer or other network device terminating traffic and then forwarding onto the reverse proxy, the IP address flag `ip: true` is not useful. The IP address is effectively static (as it is the address of the network terminating device). In order to rate limit on an IP address in this situation, have the network terminating device include the client IP address as a header, and include this in the rate limiting configuration.

```
headers:
  X-Forwarded-For: "*"
```

Rate limiting request log entries

When rate limiting occurs, the entries in the request log might be inaccurate due to the request that is being handled prior to the session being inspected.

This means that the username entry of the **request.log** is always `unauthenticated`.

When a connection is closed due to rate limiting, the status code in the log is -1. This is to signify the request was dropped without writing a status.

Ensure that the Client IP address or the **X-Forwarded-For** header is included in the request log when rate limiting because no iv-user value is present in the log entry. A value such as an IP address is necessary to correlate entries.

Index

Printed in USA