CA SiteMinder® Web Services Security

Policy Configuration Guide

12.52 SP1

CA Technologies Product References

This document references the following CA Technologies products:

- CA SiteMinder®
- CA SiteMinder® Web Services Security (formerly CA SOA Security Manager)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introducing CA SiteMinder® Web Services Security

This section contains the following topics:

CA SiteMinder® Web Services Security Overview

CA SiteMinder® Web Services Security is a policy-based access management system for Service Oriented Architecture (SOA) environments. With CA SiteMinder® Web Services Security, you can protect "big" (XML transaction-processing) web services that are implemented in the following ways:

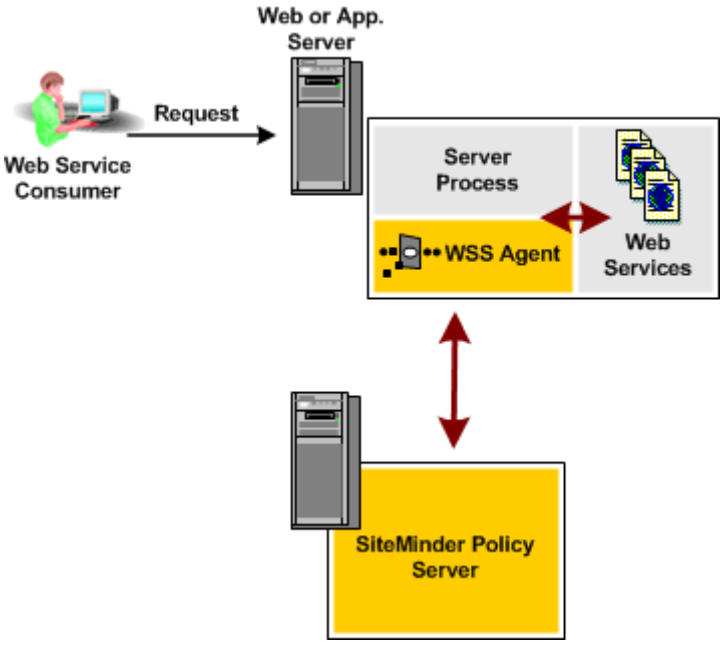
- Implemented as a servlet or Active Server Page (ASP) and exposed by a web server or an application server.
- Implemented using the JAX-RPC binding and deployed on an IBM WebSphere Application Server or Oracle WebLogic Server
- Implemented using the JAX-RPC or JAX-WS binding and deployed on a JBoss Application Server.

CA SiteMinder® Web Services Security protects XML resources in much the same way as CA SiteMinder protects HTML resources, allowing entitlement data to be obtained from any layer of the XML message, depending upon the authentication and authorization needs of the back-end applications.

CA SiteMinder® Web Services Security Architecture and Components

CA SiteMinder® Web Services Security extends SiteMinder® technology, using CA SiteMinder® Web Services Security (WSS) Agents and the Policy Server to protect web service resources hosted on web and application servers.

The following illustration shows a simple CA SiteMinder® Web Services Security environment in which a SiteMinder WSS Agent is deployed into a web or application server that is hosting web services.



More complex architectures can also be configured to support multiple web service implementations where SiteMinder WSS Agents are optionally deployed on web service endpoints to provide an additional layer of security.

Note: This guide describes only how to configure Policy Server infrastructure and policy objects to protect web service resources with CA SiteMinder® Web Services Security. For further information about configuring the Policy Server, see the CA SiteMinder® Policy Server Configuration Guide.

CA SiteMinder® Policy Server

The CA SiteMinder® 12.52 SP1 Policy Server provides a centralized, policy-based security management operating environment for securing your web resources. The CA SiteMinder® 12.52 SP1 Policy Server integrates with SiteMinder WSS Agents to secure SOAP-based web services and other CA SiteMinder® agent types to secure web applications and other resources. As such, the CA SiteMinder® 12.52 SP1 Policy Server can serve as the Policy Decision Point (PDP) in a CA SiteMinder® or CA SiteMinder® Web Services Security environment.

Note: The CA SiteMinder® 12.51 Policy Server was the first to include the CA SiteMinder® Web Services Security extensions that are required to integrate with SiteMinder WSS Agents to secure web services. Previously, only the CA SOA Security Manager Policy Server could integrate with SiteMinder WSS Agents.

The Policy Server provides the following features:

Authentication

The Policy Server supports a range of authentication methods.

Authorization

The Policy Server is responsible for managing and enforcing access control rules that are established by the Policy Server administrator. These rules define the operations that are allowed for each protected resource.

Administration

The Policy Server is configured using the CA SiteMinder® Administrative UI. The Administration service of the Policy Server allows the Administrative UI to record configuration information in the Policy Store.

Accounting

The Policy Server generates log files that contain auditing information about the events that occur within the system. These logs can be printed in the form of predefined reports, so that security events or anomalies can be analyzed.

Health Monitoring

The Policy Server provides features for monitoring activity throughout a CA SiteMinder® Web Services Security deployment.

In a CA SiteMinder® Web Services Security implementation, a web service client sends a web service request in the form of an XML/SOAP message. At the target server, an SiteMinder WSS Agent intercepts that request. The SiteMinder WSS Agent determines whether the resource is protected, and if so, gathers user credentials from the request and passes them to the Policy Server.

The Policy Server authenticates the user against native user directories, then verifies if the authenticated user is authorized for the requested resource using rules and policies that are contained in the policy store. Once a user is authenticated and authorized, the Policy Server grants access to protected resources and delivers permission and entitlement information.

Web Services Security (WSS) Agents

SiteMinder WSS Agents are the Policy Enforcement Points (PEPs) in the CA SiteMinder® Web Services Security environment, responsible for enforcing the policies defined on the Policy Server. Deployed at the end-points (web and application servers), they protect web services deployed in your SOA infrastructure.

SiteMinder WSS Agent for Web Servers

The SiteMinder WSS Agent for Web Servers is an XML-enabled version of the CA SiteMinder Web Agent. The SiteMinder WSS Agent integrates with a supported web server to authenticate and authorize requests for access to "big" web services bound to URLs served by that web server.

The SiteMinder WSS Agent for Web Servers recognizes requests that meet the following criteria as web service requests for CA SiteMinder® Web Services Security to handle:

- **Agent action**—POST; all XML message requests are posted. However, CA SiteMinder® Web Services Security also provides two other agent actions, ProcessSOAP and ProcessXML, that allow you to create rules that fire for posted requests according to the XML message format.
- **Message MIME type**—text/xml by default; configurable using the XMLSDKMimeTypes Agent parameter.

All other requests are handled using the core Web Agent functionality of the Web Agent, letting you also protect other resources on a web server.

SiteMinder WSS Agent for IBM WebSphere

The SiteMinder WSS Agent for IBM WebSphere is a container-native agent for J2EE application servers that can be used to authenticate and authorize request messages sent over HTTP(S) transport to JAX-RPC resources hosted an IBM WebSphere Application Server.

The SiteMinder WSS Agent recognizes requests that meet the following criteria as web service requests for CA SiteMinder® Web Services Security to handle:

- **Agent action**—POST; all XML message requests are posted. However, CA SiteMinder® Web Services Security also provides two other agent actions, ProcessSOAP and ProcessXML, that allow you to create rules that fire for posted requests according to the XML message format.
- **Message MIME type**—text/xml by default; configurable using the XMLSDKMimeTypes Agent parameter.

SiteMinder WSS Agent for Oracle WebLogic

The SiteMinder WSS Agent for Oracle WebLogic is a container-native agent for J2EE application servers that can be used to authenticate and authorize request messages sent over HTTP(S) or JMS transports to JAX-RPC resources hosted on an Oracle WebLogic Server.

The SiteMinder WSS Agent recognizes requests that meet the following criteria as web service requests for CA SiteMinder® Web Services Security to handle:

- **Agent action**—POST; all XML message requests are posted. However, CA SiteMinder® Web Services Security also provides two other agent actions, ProcessSOAP and ProcessXML, that allow you to create rules that fire for posted requests according to the XML message format.
- **Message MIME type**—text/xml by default; configurable using the XMLSDKMimeTypes Agent parameter.

SiteMinder Agent for JBoss

The SiteMinder Agent for JBoss provides access control for web application and web service resources hosted on the JBoss Application Server, providing the following security interceptors:

SiteMinder WSS Agent Security Interceptor

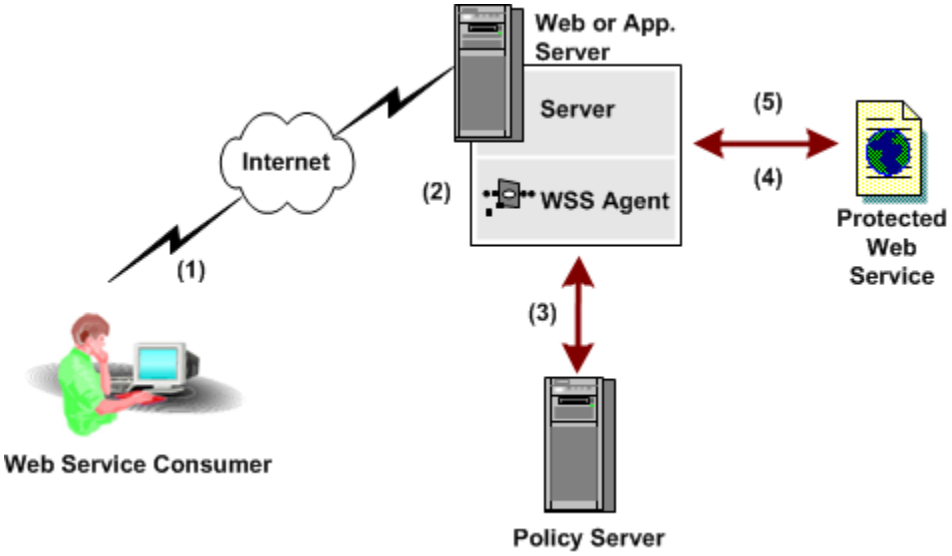
When configured into a CA SiteMinder® Web Services Security environment, the SiteMinder WSS Agent Security Interceptor provides a SiteMinder WSS Agent solution that provides CA SiteMinder® Web Services Security access control for JAX-WS and JAX-RPC web service resources.

CA SiteMinder® Agent Security Interceptor

When configured into a SiteMinder environment, the SiteMinder Agent Security Interceptor provides a SiteMinder Agent solution that provides SiteMinder access control for web application resources (including servlets, HTML pages, JSP, image files) and EJBs.

Web Service Request Processing

CA SiteMinder® Web Services Security supports content-level, XML-based security for "big" web services. The following illustration illustrates the flow of data in a simple, single web service implementation secured with CA SiteMinder® Web Services Security.



The data in the previous illustration flows as follows:

1. A web service consumer (client) application creates a web service request in the form of an XML document and sends it to the web service provider site. An example document could be a purchase order. Credentials and authorization entitlements can be inserted in the message envelope or message body.
2. At the web service provider's site, the SiteMinder WSS Agent intercepts the request, based on its action and content type in the HTTP header, as shown in the following XML sample:

```
POST /CreditRating HTTP/1.1
Content-Type: text/xml
Content-Length: nnnn
SOAPAction:"someURI:CreditRating#GetCreditRating"

<SOAP-ENV:Envelope>
    <!-- request -->
</SOAP-ENV:Envelope>
```

3. The SiteMinder WSS Agent gathers the sender's credentials from the XML message and passes this information to the CA Policy Server for authentication and authorization.
4. The authorized message is passed to the back-end business application for processing.

5. Optionally, the back-end application returns a response to the web service requester with the status of the payload (for example, indicating that the purchase order has been accepted and is being processed).

Authentication Schemes

Authentication schemes that require user intervention are generally not appropriate for securing web services. CA SiteMinder® Web Services Security provides four transport-level and message-level authentication schemes that *do not* require user intervention.

XML Document Credential Collector

Validates XML messages using credentials gathered from the message itself by mapping fields within the document to fields within a user directory.

XML Digital Signature

Validates XML documents digitally signed with valid X.509 certificates.

WS-Security

Validates XML messages using credentials gathered from WS-Security headers in a message's SOAP envelope.

CA SiteMinder® Web Services Security can produce and consume WS-Security tokens. This enables you to use the WS-Security authentication scheme to deploy a multiple-web service implementation across federated sites.

SAML Session Ticket

Validates XML messages using credentials obtained from CA SiteMinder® Web Services Security synchronized-sessioning SAML assertions (which contain an encrypted combination of a CA SiteMinder session ticket and a CA SiteMinder user's public key) placed in a message's HTTP header, SOAP envelope, or a cookie.

CA SiteMinder® Web Services Security can generate and consume SAML Session Ticket assertions. This enables you to use the SAML Session Ticket authentication scheme to deploy a multiple-web service implementation within a single Policy Server domain.

Deciding which authentication scheme or schemes you intend to use to secure your web services is integral to how you design and implement your web services and is best made as part of the broader context of choosing an authentication service model.

More information:

Authentication Service Models (see page 16)
WS-Security Authentication Introduced (see page 40)

Authentication Service Models

The ability of CA SiteMinder® Web Services Security to obtain security information from XML documents without user interaction and produce WS-Security headers, SAML Session Ticket assertions, and SiteMinder session cookies lets you securely deploy web services using a number of service models.

Single-step Authentication Service Model

All requests are authenticated and handled by a single web service.

Multistep Authentication Service Model

All requests are sent to a web service responsible for authentication, which then returns the message and authentication data back to the web service consumer. The web service consumer application can then send requests containing this authentication data to other related web services within or across domains.

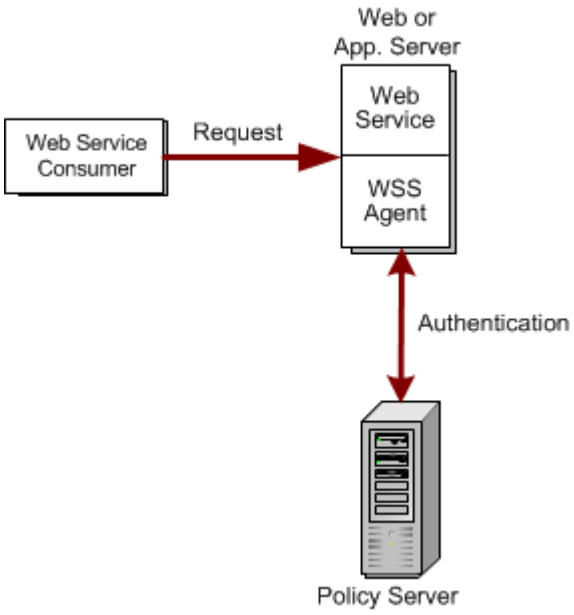
Chain Authentication Service Model

All requests are received by a web service responsible for authentication and then passed, with authentication data, to one or more other web services for handling. That is, message and authentication data always flows from the authentication web service directly to the next required web service, and from there to the next web service and so on, without further interaction from the web service consumer.

Choosing the appropriate authentication service model is the first, and probably most significant, decision you must make when designing a web service implementation. Your choice of service model also plays a significant role in determining the most appropriate CA SiteMinder® Web Services Security authentication schemes to use.

How the Single-Step Authentication Model Works

The single-step service model is the simplest possible model for web services—requests from a web service consumer are authenticated and handled by a single web service. The following diagram shows the process by which web services consumers are authenticated using this simple model:



Appropriate authentication schemes for use in the single-step authentication model are as follows:

- XML Document Collector Authentication Scheme
- XML Digital Signature Authentication Scheme

How the Multistep Authentication Model Works

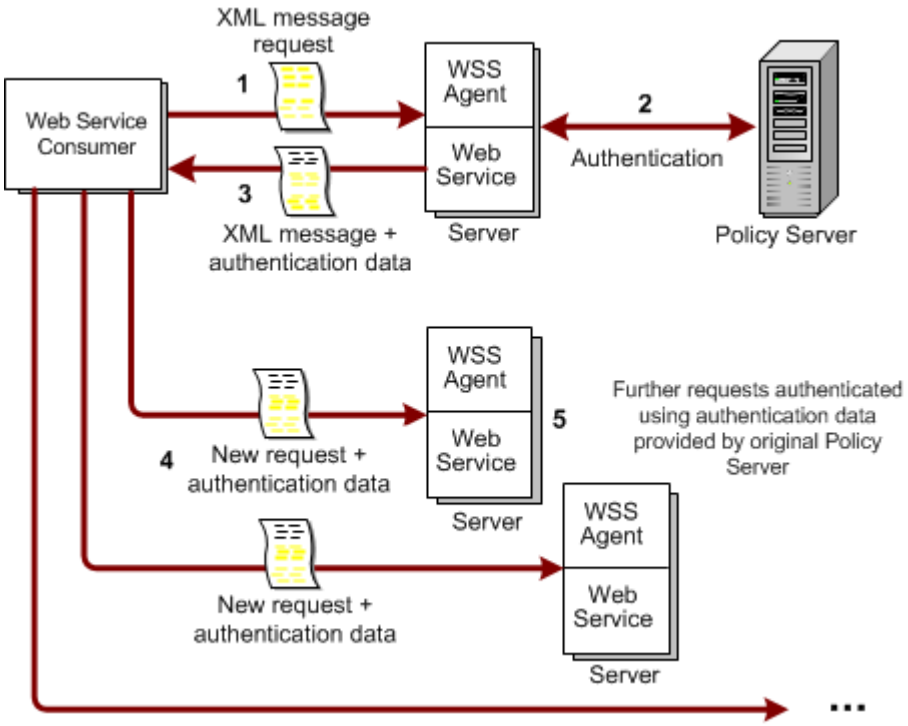
The multistep authentication model is like the CA SiteMinder cookie-based single sign-on implementation, in which WS-Security headers or SAML Session Ticket assertions take the place of the cookie.

In the multistep authentication model, a single web service is responsible for authenticating all incoming web service requests. This authentication service verifies a web service consumer's identity and returns an XML message with authentication data in the form of WS-Security headers or a SAML Session Ticket assertion. The web service consumer can then use this to add to subsequent requests to facilitate authentication by other associated web services.

The process that the web service consumer goes through when making a request has two phases:

1. Obtaining the authentication data
2. Using the authentication data to access other web services

The following illustration shows how request are processed in the multistep authentication service model:



1. The web service consumer sends a request for access to a protected web Service in the form of an XML document.
2. The SiteMinder WSS Agent receives the request, extracts credentials and passes them to the Policy Server, which authenticates the web service request with an appropriate authentication scheme.

   After authentication, the request goes through the authorization process. A response attribute associated with the authorizing policy causes the Policy Server to generate a response which it sends to the SiteMinder WSS Agent, instructing it to return authentication data to the web service.
3. The web service returns the authentication data back to the web service consumer (typically in an XML document, but synchronized sessioning SAML assertions can also be returned in HTTP headers or a cookie).
4. For subsequent requests, the web service consumer passes XML messages that include the authentication data it received from the authentication service to other associated web services.

5. The requests are allowed access without having to reauthenticate because the authentication data is supplied with the request message (in effect, providing single sign-on).

Appropriate authentication schemes for initial authentication by the authentication web service in the multistep authentication model are as follows:

- XML Document Collector Authentication Scheme
- XML Digital Signature Authentication Scheme

The authorizing policy for the authentication web service should trigger one of the following response types:

- WS-Security Responses (appropriate for web services protected by more than one policy store or at multiple sites)
- SAML Session Ticket Responses (appropriate for web services protected by the same policy store)

These responses instruct the SiteMinder WSS Agent to pass authentication data in the form of WS-Security headers or SAML Session Ticket assertions (as appropriate) back to the web service consumer for use in requests to associated web services. The associated web services should be protected using the corresponding authentication scheme:

- WS-Security Authentication Scheme
- SAML Session Ticket Authentication Scheme

More information:

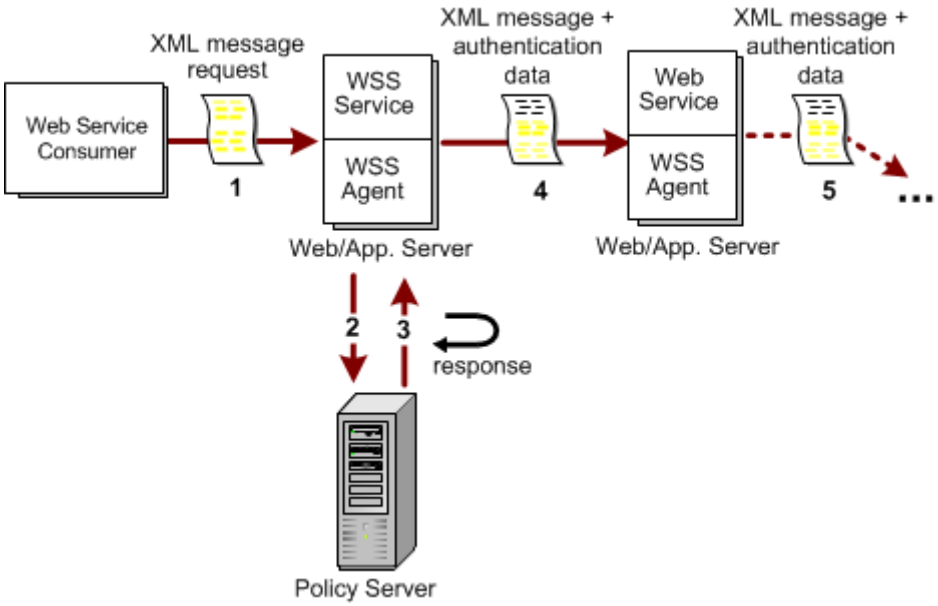
Multistep and Chain Authentication Using SiteMinder Session Tickets (see page 21)

How the Chain Authentication Service Model Works

The chain authentication model is appropriate for solutions that require XML messages to flow between multiple web services without further intervention from the requesting web service consumer.

In the chain authentication service model, a single web service is responsible for authenticating all incoming web service requests. This authentication service verifies a web service consumer's identity, and then adds authentication data in the form of WS-Security headers or a SAML Session Ticket assertion to the XML message. It then passes the document to downstream web services for processing.

The following illustration shows the flow of data in the chain authentication model.



1. The web service consumer sends a request for access to a protected web Service in the form of an XML document.
2. The SiteMinder WSS Agent receives the request, extracts credentials and passes them to the Policy Server, which authenticates the web service request with an appropriate authentication scheme.
3. After authentication, the request goes through the authorization process. A response attribute associated with the authorizing policy causes the Policy Server to generate a response which it sends to the SiteMinder WSS Agent, instructing it to return authentication data to the authentication web service.
4. The authentication web service sends the XML message and authentication data to the next web service downstream.
5. Downstream web services are configured so that each passes the XML message and authentication data to the next web service in the chain. The requests are allowed access without having to reauthenticate because of the authentication data supplied with the request message.

The most appropriate authentication schemes for initial authentication of requests from the web service consumer by the authentication web service in the chain authentication model are as follows:

- XML Document Collector Authentication Scheme
- XML Digital Signature Authentication Scheme

The authorizing policy for the authentication web service should trigger one of the following responses:

- WS-Security Responses (appropriate for web services protected by more than one policy store or at multiple sites)
- SAML Session Ticket Responses (appropriate for web services protected by the same policy store)

These responses instruct the SiteMinder WSS Agent to add WS-Security headers or SAML Session Ticket assertions (as appropriate) to the XML request passed to the next downstream web service in the chain, which should then be protected using the corresponding authentication scheme:

- WS-Security Authentication Scheme
- SAML Session Ticket Authentication Scheme

More information:

Multistep and Chain Authentication Using SiteMinder Session Tickets (see page 21)

Multistep and Chain Authentication Using SiteMinder Session Tickets

Although CA SiteMinder® Web Services Security is primarily designed to provide message content-based security for web services, it also provides limited support for CA SiteMinder® session ticket-based session management. A CA SiteMinder® session cookie contains basic information about the user account that is associated with a request and authentication information for that user. The session cookie can be used to identify a user session across all sites within a single cookie domain in your WSS environment..

SiteMinder WSS Agents always add session cookies to HTTP request headers upon successful authentication and authorization. SiteMinder WSS Agents that have access to HTTP header information can be configured to authenticate a request using a session cookie in the HTTP request header. In this case, there is no need to configure responses to generate other token types at the authentication web service.

For example, in the following environment, session cookies can be used without the need to configure WS-Security or SAML Session Ticket responses at the authentication web service:

- An authentication web service is configured to validate incoming requests using XML Digital Signature authentication.
- All the other web services in the chain authentication implementation reside within the same cookie domain.
- All the SiteMinder WSS Agents that protect web services beyond the authentication web service are configured to accept session cookies.

To configure SiteMinder WSS Agents to validate requests using associated session tickets, set the XMLSDKAcceptSMSessionCookie agent configuration parameter. See the SiteMinder WSS Agent documentation for details.

How to Develop and Deploy CA SiteMinder® Web Services Security Protected Web Services

To develop a web service implementation protected with CA SiteMinder® Web Services Security, do the following:

1. Determine how many web services, locally or at federated sites, will be used to perform the required functionality.
2. Choose an authentication service model by determining the following:
   - How security information is to be obtained from a request and, in a multiple-web service environment, how that information is to be passed between web services.
   - In a multiple-web service environment, the flow of data between web services.
3. For each web service in your web service implementation, determine the following:
   a. Define the service interface. The simplest form of interface for a web service can be specified as a set of XML schemas. These schemas dictate the type of XML document to be sent to the web service and what type of document the sender can expect in return.
   b. Build the web service implementation to accommodate an incoming XML document of the type specified in the interface and turn that XML document into a meaningful set of calls to the integrated back-end systems that the web service exposes.
   c. Deploy your web service implementation to a web server or application server protected by a SiteMinder WSS Agent. You direct consumers of your web service to send their XML message requests to this URI to access the web service.
   d. Configure CA SiteMinder® Web Services Security policies to determine how the SiteMinder WSS Agent should authenticate, authorize, and process the XML message before it passes it onto the web service implementation for handling.

      Once it receives a message from the SiteMinder WSS Agent, the web service should return an applicable XML response to the calling web service consumer application or the next.

Chapter 2: Configure Authentication Schemes to Verify User Identities Obtained from Web Service Requests

This section contains the following topics:

Authentication Scheme Overview

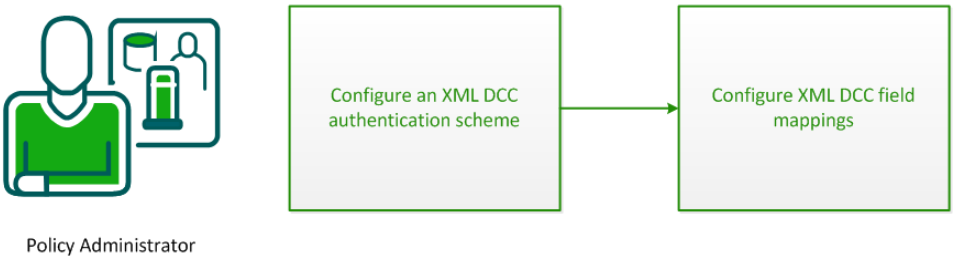
CA SiteMinder® Web Services Security authentication schemes provide a way to collect credentials from an XML message request sent to a protected web service and determine the identity of the user represented by those credentials.

Authentication schemes must be configured using the Administrative UI. During authentication, SiteMinder WSS Agents communicate with the Policy Server to determine the proper credentials that must be retrieved from the request message.

This chapter discusses general information for working with authentication schemes in the Administrative UI, then provides separate sections that explain how to configure each supported scheme using authentication scheme templates. These templates provide the Policy Server with most of the information it needs to process a scheme. An administrator must complete the configuration of an authentication scheme by supplying implementation specific information, such as server IP addresses, or shared secrets required to initialize a scheme.

How to Configure XML DCC Authentication to Verify User Identities Using Credentials Gathered from XML Request Messages

Configure an XML Document Credential Collector (XML DCC) authentication scheme to validate user credentials obtained from incoming web service request documents.



To configure CA SiteMinder® Web Services Security to validate user identities using XML DCC authentication, complete the following process:

1. Configure the XML DCC authentication scheme (see page 24).
2. Configure XML DCC field mappings (see page 25).

Configure the XML DCC Authentication Scheme

To obtain authentication information from an incoming XML document, configure the XML DCC authentication scheme.

Follow these steps:

1. Click Infrastructure, Authentication.
2. Click Web Services Authentication Schemes, Create Authentication Scheme.

   The Create Authentication Scheme pane opens.

   Authentication scheme settings open.
3. Enter a name and a description for the scheme in the General group box.
4. Select XML Document Credential Collector from the Authentication Scheme Type list.
5. Specify a protection level.
6. Configure XML DCC field mappings (see page 25) in the Scheme Setup section.

7. (Optional) Select the Require Secure Transport Layer check box to require that authentication only take place over an SSL connection.

   Important! The Policy Server expects the information in the XML document to be in clear text. To enforce security, we recommend that you configure this authentication scheme over an SSL connection.

8. Click Submit.

   The authentication scheme is saved. You can now assign it in application object components or realms.

More information:

(Optional) Configure Other Field Mappings (see page 27)
Configure the Required "user" Mapping (see page 25)

How to Configure XML DCC Field Mappings

To create XML DCC mappings in the Administrative UI, map a user store field name to an XPath string that identifies an element of an XML document. Create these field mappings by browsing a specific XML schema file (.xsd or .dtd) or by entering an XPath query language string directly.

The XML DCC authentication scheme *requires* only one mapped field—"user"—to identify the XML document element that identifies the user to authenticate. To meet this requirement, the Field Mapping dialog forces the first field mapping that you create to be named "User". The only other specific field mapping name is "Password." To authenticate users by username and password, configure a second mapping named "Password."

To configure XML DCC field mappings, complete these procedures:

1. Configure the required "user" mapping (see page 25)
2. (Optional) Configure other field mappings (see page 27)

Configure the Required "user" Mapping

The required "user" mapping entry maps the user name field in the user store. The "user" mapping is created when you create an XML DCC authentication scheme. Before you configure any further mappings, map this value to a field in the XML document.

Two methods for creating mappings are as follows:

- Open a schema and select elements from it that you want to map to the user store. This method is easier because CA SiteMinder® Web Services Security builds the XPath query for you as you select fields for mappings.
- Use the Advanced XPath query option to build more complex XPath queries manually.

Follow these steps:

1. Locate the Scheme Setup section on the Create Authentication Scheme pane and click the Edit button beside the "user" field mapping entry.

   Field mapping settings open.

2. Do one of the following:
   - Type an XPATH query defining the mapping for "user" in the XPath field.
   - Load a schema (.xsd) file and select the element to map to "user" by browsing using the following procedure:
     a. Unset the Advance XPath query option.
     b. If the schema file you require is remote (for instance, if it is typically accessed over HTTP using its URL), download it to a local drive.
     c. Click Browse and navigate to a schema file in the File Upload dialog that appears.
     d. Click Upload XSD File.

        The schema is uploaded.

     e. Select the schema element that you want to map to the 'user" field name in the Select a node group box.

        The Select a node group box displays the selected schema using a standard tree-style hierarchical view. Click the plus sign (+) next to an element to expand it. Click the minus sign (-) beside an expanded element to contract it. Elements marked with an asterisk (*) are repeatable within the XML document (that is, incoming XML documents may contain multiple instances of that element).

3. (Optional) Specify the XPath function (count, div, index, mod, sum) that you want to apply to the mapping by choosing it from the Function drop-down list.

   The Function option lets you create more complex mappings by processing functions that further evaluate the XML document. For more information about these functions, navigate to the XPath specification at http://www.w3.org.

4. Specify whether the mapped information is located relative to the message body or message header by selecting the Message Body or Message Header option button.

   This defines the root of the XML document and tells XPath where to search for the relevant information. If the document has multiple headers, XPath uses the value of the first header that resolves.

5. Click OK to save your changes and return to the Create Authentication Scheme pane.

(Optional) Configure Other Field Mappings

Aside from the "user" mapping, you can define any number of other XML DCC field mappings.

Two methods for creating mappings are as follows:

- Open a schema and select elements from it that you want to map to the user store. This method is easier because CA SiteMinder® Web Services Security builds the XPath query for you as you select fields to map.
- To build more complex XPath queries manually, use the Advanced XPath query option.

Follow these steps:

1. Locate the Scheme Setup section on the Create Authentication Scheme pane and click Add.

   Field mapping settings open.

2. Enter the name of a field in the user store, such as "email" in the Name field. This specifies the name of the field to which you are mapping the XML element.

   Note: This name must match an entry in the user store; it is not case-sensitive.

3. Do one of the following tasks:
   - Type an XPATH query defining the mapping in the XPath field.
   - Load a schema (.xsd) file and browse to the element to map to the specified field name using the following procedure:
     a. Unset the Advance XPath query option.
     b. Click Browse.
     c. Navigate to a schema file in the File Upload dialog that appears.
     d. Click Upload XSD File.

        The schema is uploaded.
     e. Select the schema element that you want to map to the specified field name in the Select a node group box.

        The Select a node group box displays the selected schema using a standard tree-style hierarchical view. To expand an element, click the plus sign (+) next to it. To contract an element, click the minus sign (-) next to it. Elements marked with an asterisk (*) are repeatable within the XML document (that is, incoming XML documents can contain multiple instances of that element).

     Note: A loaded schema is not persistent; even when creating multiple mapping from the same schema file, you must reload the schema for each mapping.
4. (Optional) Specify the XPath function (count, div, index, mod, sum) that you want to apply to the mapping by choosing it from the Function drop-down list.

   The Function option lets you create more complex mappings by processing functions that further evaluate the XML document. For more information about these functions, see the XPath specification at http://www.w3.org.
5. Specify whether the mapped information is located relative to the message body or message header by selecting the Message Body or Message Header option button.

   This setting defines the root of the XML document and tells XPath where to search for the relevant information. If the document has multiple headers, XPath uses the value of the first resolved header.
6. Click OK to save your changes and return to the Create Authentication Scheme pane.

XML DCC XPath Mapping Examples

The following examples show XPath expressions to perform complex mappings.

Example Namespace-aware XPath Query

In the following XML file, the username and password are in the SOAP body, and the first element below the body is prefixed by a namespace. It would not therefore be possible to obtain these elements using the schema browsing method.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header>
  </env:Header>
  <env:Body>
    <n1:sayHello testnum="purchaseOrder11c"
xmlns:n1="http://www.xyz.com/examples/Trader">
      <BillingInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\data\CredentialHeader.xsd">
        <CustomerCredentials>
                <username>Robm</username>
                <password>password</password>
                <PIN>String</PIN>
        </CustomerCredentials>
      </BillingInformation>
    </n1:sayHello>
  </env:Body>
</env:Envelope>
```

To obtain the username, specify the following XPath query:

```
/*[local-name()='sayHello' and
namespace-uri()='http://www.xyz.com/examples/Trader']/BillingInformation/Customer
Credentials/username
```

To obtain the password, specify the following XPath query:

```
/*[local-name()='sayHello' and
namespace-uri()='http://www.xyz.com/examples/Trader']/BillingInformation/Customer
Credentials/password
```

Example XPath Query to Obtain Credentials From Embedded XML Documents

Required credentials can be present in a SOAP body payload, but the XML screened from the parser by a CDATA section or by replacement of angle brackets by entity references.

The following XPath queries will work for either CDATA or entity-reference screened XML.

XPath query for username

The following XPath query can be used to obtain a username mapping from the CDATA section or from entity-reference screened XML:

```
        substring-before(substring-after(/*[local-name()='sayHello' and
namespace-uri()='http://www.bea.com/examples/Trader']/text(),'<username>'),'</use
rname>')
```

XPath query for password

The following XPath query can be used to obtain a password mapping from the CDATA section or from entity-reference screened XML:

```
        substring-before(substring-after(/*[local-name()='sayHello' and
namespace-uri()='http://www.bea.com/examples/Trader']/text(),'<password>'),'</pas
sword>')
```

Sample document containing credentials in CDATA section

The following sample XML document shows username and password credentials that are screened by a CDATA section.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Header>
<BillingInformation>
        <CustomerCredentials>
                <username>Robm</username>
                <password>password</password>
        </CustomerCredentials>
</BillingInformation>
</env:Header>
<env:Body>
<n1:sayHello testnum="purchOrder05-cdata"
xmlns:n1="http://www.bea.com/examples/Trader">

<![CDATA[<!--Sample XML file generated by XMLSpy v2005 rel. 3 U
(http://www.altova.com)-->
```

```
<BillingInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\data\CredentialHeader.xsd">

        <CustomerCredentials>
                <username>Robm</username>
                <password>password</password>
                <PIN>String</PIN>
        </CustomerCredentials>

</BillingInformation>
]]>
</n1:sayHello>
</env:Body>
</env:Envelope>
```

Sample document containing credentials in entity-referenced screened XML

The following sample XML document shows username and password credentials that are screened by the use of replacement of angle brackets by entity references.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<env:Header>
<BillingInformation>
        <CustomerCredentials>
                <username>Robm</username>
                <password>password</password>
        </CustomerCredentials>
</BillingInformation>
</env:Header>
```

```
<env:Body>
<n1:sayHello testnum="purchaseOrder04"
xmlns:n1="http://www.bea.com/examples/Trader">

&lt;!--Sample XML file generated by XMLSpy v2005 rel. 3 U
(http://www.altova.com)--&gt;

&lt;BillingInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\data\CredentialHeader.xsd"&gt;

        &lt;CustomerCredentials&gt;
                &lt;username&gt;Robm&lt;/username&gt;
                &lt;password&gt;password&lt;/password&gt;
                &lt;PIN&gt;String&lt;/PIN&gt;
        &lt;/CustomerCredentials&gt;
&lt;/BillingInformation&gt;

</n1:sayHello>
</env:Body>
</env:Envelope>
```

Example XPath Query to Obtain Credentials with a Default Namespace for all Elements

In the following XML file, the sayHello element has a default namespace specified by xmlns="http://www.xyz.com/examples/Trader".

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Header>
  </env:Header>
  <env:Body>
    <sayHello testnum="purchaseOrder11c"
xmlns="http://www.xyz.com/examples/Trader">
      <BillingInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\data\CredentialHeader.xsd">
        <CustomerCredentials>
                <username>Robm</username>
                <password>password</password>
                <PIN>String</PIN>
        </CustomerCredentials>
      </BillingInformation>
    </sayHello>
  </env:Body>
</env:Envelope>
```

This following XPath query searches for the element "username" with the namespace "http://www.xyz.com/examples/Trader" anywhere in the document:

```
/*[local-name()='username' and
namespace-uri()='http://www.xyz.com/examples/Trader']
```

Example XPath Query that Explicitly Specifies the Namespace Prefix

To extract the username and password (without namespace prefix) from a SOAP message, you can use an XPath query with an explicit tag including the namespace prefix and colon (:) as a simple text string.

For example, you could use the following XPath query to extract the username and password (without namespace prefix) from the sample SOAP message.

Example XPath query with explicit tag

This XPath query could be used to extract the username and password (without namespace prefix) from the sample SOAP message that follows.

```
//*[name()='wsu:dccuser']
//*[name()='wsu:dccpwd']
```

Sample SOAP message

The preceding XPath query could be used to extract the username and password (without namespace prefix) from this sample SOAP message.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"

xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <soap:Header>

<wsa:Action>http://example.com/services/XMLProcess_WebReq_portWebReq/Operation_1</wsa:Action>
   <wsa:MessageID>urn:uuid:e0b940b0-7d44-4e1e-b391-2e65c5b1de3f</wsa:MessageID>
   <wsa:ReplyTo>

<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
   </wsa:ReplyTo>
```

```
<wsa:To>http://ex01/XMLProcess_Proxy/XMLProcess_WebReq_portWebReq.asmx</wsa:To>
   <wsse:Security>
    <wsu:Timestamp wsu:Id="Timestamp-cedeb96e-9b12-45e6-bdf1-6e6c323a24cb">
     <wsu:Created>2006-04-12T18:31:33Z</wsu:Created>
     <wsu:Expires>2006-04-12T18:36:33Z</wsu:Expires>
     <wsu:dccuser>catest1</wsu:dccuser>
     <wsu:dccpwd>msimsi</wsu:dccpwd>
    </wsu:Timestamp>
   </wsse:Security>
 </soap:Header>
 <soap:Body>
   <Operation_1 xmlns="http://example.com/services">
    <XMLClaim MemberID="123456789" SubscriberID="987654321" TimeStamp="20060412
13:31:32:952" TranNumber="270" ControlID="1" xmlns="http://Example.Claim_XML" />
   </Operation_1>
 </soap:Body>
</soap:Envelope>
```

Example XPath Query With Namespace and Element-by-Element Navigation

Use the following XPath query, with namespace and element-by-element navigation:

```
/*[local-name()='Security' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-secext-1.0.xsd'][1]/*[local-name()='Timestamp' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-utility-1.0.xsd'][1]/*[local-name()='dccuser' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-utility-1.0.xsd'][1]

/*[local-name()='Security' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-secext-1.0.xsd'][1]/*[local-name()='Timestamp' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-utility-1.0.xsd'][1]/*[local-name()='dccpwd' and
namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
ty-utility-1.0.xsd'][1]

/*[local-name()='Security' and
    namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
    curity-secext-1.0.xsd'][1]
/*[local-name()='Timestamp' and
    namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
    curity-utility-1.0.xsd'][1]
/*[local-name()='dccuser' and
    namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
    curity-utility-1.0.xsd'][1]
```

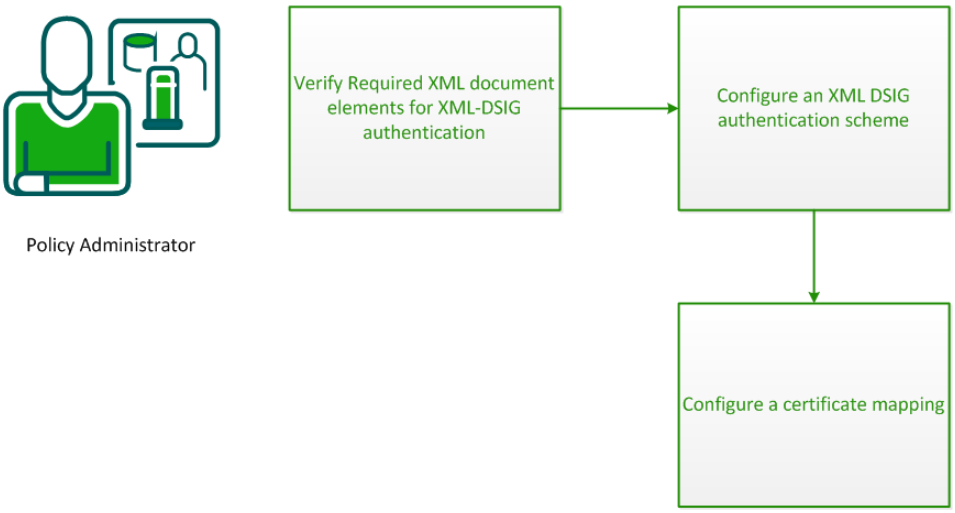
To extract the username and password from the following SOAP message:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"

xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">


<wsa:Action>http://example.com/services/XMLProcess_WebReq_portWebReq/Operation_1</wsa:Action>
   <wsa:MessageID>urn:uuid:e0b940b0-7d44-4e1e-b391-2e65c5b1de3f</wsa:MessageID>
   <wsa:ReplyTo>

<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
   </wsa:ReplyTo>

<wsa:To>http://ex01/XMLProcess_Proxy/XMLProcess_WebReq_portWebReq.asmx</wsa:To>
   <wsse:Security>
    <wsu:Timestamp wsu:Id="Timestamp-cedeb96e-9b12-45e6-bdf1-6e6c323a24cb">
     <wsu:Created>2006-04-12T18:31:33Z</wsu:Created>
     <wsu:Expires>2006-04-12T18:36:33Z</wsu:Expires>
     <wsu:dccuser>catest1</wsu:dccuser>
     <wsu:dccpwd>msimsi</wsu:dccpwd>
    </wsu:Timestamp>
   </wsse:Security>
 </soap:Header>
<soap:Body>
   <Operation_1 xmlns="http://example.com/services">
    <XMLClaim MemberID="123456789" SubscriberID="987654321" TimeStamp="20060412
13:31:32:952" TranNumber="270" ControlID="1" xmlns="http://Example.Claim_XML" />
   </Operation_1>
 </soap:Body>
</soap:Envelope>
```

How to Configure XML DSIG Authentication to Verify User Identities Associated with X.509 Certificates

Configure an XML Digital Signature (XML DSIG) authentication scheme to verify user identities that are associated with the X.509 certificates used to sign XML request messages.

To use an X.509 certificate to identify a user, the XML DSIG authentication scheme uses a certificate mapping to compare a certificate with a user in a user directory. A certificate mapping defines how data in the certificate is mapped to form a user Distinguished Name (DN), which the Policy Server uses to authenticate the client.



To configure CA SiteMinder® Web Services Security to validate user identities using XML DSIG authentication, complete the following process:

1. Verify required XML document elements for XML-DSIG authentication (see page 37)
2. Configure the XML DSIG authentication scheme (see page 37)
3. Configure a certificate mapping (see page 38)

Verify Required XML Document Elements for XML-DSIG Authentication

For the XML-DSIG authentication scheme to work, the XML document sent by the web service consumer must contain the following elements:

<Signature>

As the parent element for the XML signature, it specifies all information relevant to the digital signature.

To verify the signature, CA SiteMinder® Web Services Security requires that an X.509 certificate be part of the <Signature> element in the XML document.

Because the Policy Server does not interact with a Certificate Authority for this scheme, you *must configure a certificate mapping* that maps the Issuer DN in the certificate to a corresponding entry in the referenced user store. For LDAP user directories only, you can configure the certificate mapping to require that a copy of the certificate is in the user store to be compared against the certificate in the document.

<KeyInfo>

This element specifies the key needed to validate the signature. This information may include keys, names, and certificates for the sender.

For the Policy Server to authenticate a client, this element must have enough information to determine the public key that created the signature.

<KeyName>

This is a child element of <KeyInfo>; it contains a string value that identifies the key to the recipient of the XML document. This string could be a key index, a distinguished name (DN), or an email address, for example.

The Policy Server maps the value of this element to an entry in the user store.

Configure the XML DSIG Authentication Scheme

To obtain authentication information from digital signatures associated with incoming XML documents, you configure the XML DSIG authentication scheme.

Follow these steps:

1. Click Infrastructure, Authentication.
2. Click Web Services Authentication Schemes, Create Authentication Scheme.

   The Create Authentication Scheme pane opens.

   Authentication scheme settings open.
3. Enter a name and a description for the scheme in the General group box.
4. Select XML Digital Signature from the Authentication Scheme list.
5. Specify a protection level.
6. In the Scheme Setup group box, select how much of the XML document content is signed. A digital signature can apply only to one portion of an XML document. The choices are as follows:

   - Must cover the entire document
   - Must cover the body of the message
   - Only needs to apply to headers

   Note: If the XML document uses raw XML, select the Must cover entire document option, because the entire document is the payload. With raw XML, no envelope headers or body tags exist to distinguish the payload from other content.
7. To perform authentication over an SSL connection, select the Require Secure Transport Layer check box.
8. Click Submit.

   The authentication scheme is saved. You can now assign it in application object components or realms.
9. Configure certificate mapping for the XML-DSIG scheme.

   A certificate mapping defines how data in the certificate is mapped to form a user Distinguished Name (DN), which the Policy Server uses to authenticate the client.

Configure a Certificate Mapping

To determine how to compare user certificate information with the information stored in the user directory, configure a certificate mapping.

Follow these steps:

1. Click Infrastructure, Directory.
2. Click Certificate Mappings.
3. Click Create Certificate Mapping.

4. Type the Issuer DN exactly as it appears in the certificate. Do not add any additional spaces or characters.

   When entering the DN, escape reserved special characters with a backslash (\). Special characters include:

   - semicolons (;)
   - quotes (")
   - backslashes (\)
   - plus character (+)
   - greater than character (>)
   - less than character (<)

   More information about reserved special characters for DNs exists at http://www.faqs.org/rfcs/rfc2253.html.

   Note: If you use a relational database as a policy store, Issuer DNs cannot exceed 255 characters. If you use an LDAP directory as a policy store, verify the character limit for your specific directory.

5. Select the directory type against which the certificate is mapped.

   For LDAP directories only, you can configure the Policy Server to verify that the certificate the user presents matches the certificate that is stored in the user record in the user directory. The Certificate Required in Directory option lets you require this verification.

   Note: The attribute in the user record where the certificate is stored is named **usercertificate**.

6. Specify how to map X.509 user certificate information to a user entry in the user directory. The Policy Server can apply a mapping using one of the following properties to locate the correct user entry:

   - A single attribute
   - A custom mapping expression
   - The entire Subject Name from the user certificate

7. Select an attribute name from the list.
8. Click Test to test the certificate mapping.
9. (Optional) Select Perform CRL Checks and specify the CRL settings.

   If you do not select CRLs, you can use OCSP.

10. Click Submit.

    The certificate is mapped with the selected user directory.

WS-Security Authentication Introduced

The WS-Security standard defines a set of SOAP header extensions that provide mechanisms for securely passing authentication data and protecting message content between web services, particular those at federated enterprises. WS-Security allows web service implementers to do the following:

- Send authentication data as part of a message using one of several supported security token types.
- Ensure message integrity using digital signatures.
- Ensure message confidentiality using XML encryption.

These mechanisms can be used independently (for example, to pass authentication data in a security token) or in combination (for example, signing and encrypting a message and providing authentication data in a security token).

For more information about the WS-Security standard, see the OASIS Standard, *Web Services Security: SOAP Message Security 1.0*.

XML Encryption

CA SiteMinder® Web Services Security supports encryption and decryption of any combination of WS-Security message header elements and body elements using XML encryption compliant with OASIS Standard 200401, *Web Services Security: SOAP Message* Security *1.0*.

XML encryption of WS-Security messages provides end-to-end security for web service applications that require secure exchange of structured data. XML encryption provides security functionality that cannot be provided by point-to-point security protocols such as SSL. Specifically, it allows you to:

- Encrypt only part of the data being exchanged
- Secure sessions between more than two parties

How CA SiteMinder® Web Services Security Obtains Credentials from Encrypted WS-Security Documents

The WS-Security authentication scheme automatically attempts to decrypt any XML-encrypted elements in incoming WS-Security messages for CA SiteMinder® Web Services Security to use for authentication/authorization. No additional configuration is required.

Note: Where an incoming SOAP message contains multiple WS-Security header elements, each is identified by a unique SOAP actor/role attribute. In such cases, CA SiteMinder® Web Services Security attempts to decrypt only XML-encrypted elements specified in the header from which the WS-Security authentication scheme is configured to obtain security tokens.

More information:

SOAP Actor/Role Attributes in Messages with Multiple WS-Security Headers (see page 43)

Configure CA SiteMinder® Web Services Security to Perform Encryption and Decryption of WS-Security Documents

CA SiteMinder® Web Services Security can encrypt any WS-Security message that contains the recipient's X.509 certificate in a WS-Security header. CA SiteMinder® Web Services Security extracts the recipient's public key from their X.509 certificate and uses this to encrypt a symmetric key, which it then uses to encrypt the desired header and message elements. Multiple encryption algorithms are available; different encryption algorithms can be used for encryption of the symmetric key and header/message elements.

Configure CA SiteMinder® Web Services Security to perform XML encryption on elements of outgoing messages by adding appropriate response attribute variables to a response configured to generate WS-Security headers.

Although the WS-Security authentication scheme automatically decrypts encrypted elements in incoming messages, the default behavior of CA SiteMinder® Web Services Security is to deliver messages to the recipient web service in encrypted form. However, you can configure CA SiteMinder® Web Services Security to deliver decrypted versions of incoming encrypted WS-Security messages by configuring a response with the TXM_WSSEC[_SAML]_ENCRYPT_DECRYPT response attribute variable and associating it with the authorizing policy.

XML Encryption and Decryption Service Use Case

In multistep and chain authentication service models, encryption or decryption may be considered part of message preparation before sending to the ultimate destination. Thus, the CA SiteMinder® Web Services Security XML encryption and decryption functionality might typically be used to implement a WS-Security encryption or decryption web service.

For example, consider a business relationship between two companies—Company A and Company B. Company A wants to end detailed bids on contracts to Company B without unauthorized personnel at Company B seeing the message (as would be the case if it was simply sent over an SSL link).

To implement this business logic using CA SiteMinder® Web Services Security-protected web services, Company A develops the following:

- A web service consumer application that takes the bid, places it in XML format and wraps it with SOAP headers, placing Company B's X.509 certificate in a WS-Security header. The application then sends it to Company A's encryption web service.
- An encryption web service protected by the WS-Security authentication scheme and an authorization policy configured to do the following:
  - Obtain the intended recipient's public key certificate (in this case Company B's certificate) from the message headers
  - Encrypt the required header and message elements.

  The encryption web service then forwards the encrypted message to a decryption web service at Company B.

Company B develops a Decryption web service protected by the WS-Security authentication scheme and an authorization policy configured to deliver the decrypted version of message header/body elements.

Encryption Algorithms

CA SiteMinder® Web Services Security supports the following XML encryption algorithms:

- **Key transport algorithms** (used to encrypt the symmetric key):
  - rsa-1_5
  - rsa-oaep-mgf1p
- **Block Encryption algorithms** (used to encrypt message data):
  - tripledes-cbc
  - aes128-cbc
  - aes256-cbc
  - aes192-cbc

Message Timestamps

Regardless of the particular security token used by any WS-Security document, a utility timestamp element, which specifies the expiry time of a message, can be specified. If this element is covered by an XML signature, then the timestamp provides a protection against replay attacks for the entire XML document (different from the replay attack defense provided by the Username and Password Digest token) by giving an indication of the document's “freshness.”

Note: The expiry feature does not completely address the problems introduced by unsynchronized clocks. The receiving party in a WS-Security message exchange may receive a document before the timestamp's created time; the issue of acceptable skew is a receiving policy issue, while the expiry offset is a creation policy issue.

XML Signature Scope

CA SiteMinder® Web Services Security provides three options for defining what elements of an incoming SOAP message are digitally signed when configuring WS-Security authentication using either Username and Password Digest or X509v3 tokens:

- Signature must cover the entire document
- Signature must cover the body of the message
- Signature need apply only to headers

Notes:

For the Username and Password Digest token, XML digital signatures are optional.

If the authentication scheme is configured to require the timestamp element, the digital signature must cover that timestamp.

SAML token authentication has its own requirements for what elements of a SOAP message must be digitally signed; these are defined implicitly based on the subject confirmation methods that you require.

SOAP Actor/Role Attributes in Messages with Multiple WS-Security Headers

If a SOAP document has multiple WS-Security headers (intended for different recipients), the WS-Security specification requires that each be identified uniquely using the SOAP actor/role attribute (at most, one header can omit the SOAP actor attribute).

The WS-Security authentication scheme lets you specify the value of the SOAP actor/role attribute that identifies the header element from which CA SiteMinder® Web Services Security should obtain security tokens.

Note: If a message has only one WS-Security header, it does not need to include a SOAP actor attribute. However, if you specify an actor/role attribute when configuring the authentication scheme, a matching actor attribute must be present in the document to allow successful authentication.

Username and Password Digest Token Age Restrictions

The WS-Security authentication scheme provides protection against replay attacks using Username and Password Digest tokens by imposing a "freshness" restriction (60 minutes by default) on the age of the token. That is, if a token was created more than 60 minutes ago according to its <wsu:Created> timestamp, authentication fails.

The token age restriction for Username and Password Digest Tokens can be configured at the agent level. For more information, see the SiteMinder WSS Agent configuration documentation.

How to Configure WS-Security Authentication to Verify User Identities Obtained from WS-Security Headers

Configure a WS-Security authentication scheme to verify user identities using credentials obtained from WS-Security tokens in the SOAP header of a request message. The WS-Security authentication scheme can also validate digital signatures and decrypt XML encrypted headers as necessary..

To configure CA SiteMinder® Web Services Security to validate user identities using WS-Security authentication, complete the following process:

1. Verify that certificates required to validate signed tokens are present in the certificate data store (see page 45)
2. Configure the WS-Security authentication scheme (see page 45)

Verify That Certificates Required to Validate Signed Tokens are Present in the Certificate Data Store

CA SiteMinder® Web Services Security uses the public key certificates of trusted issuers to validate signed WS-Security tokens.

Public key certificates are stored in the certificate data store (CDS). The certificate data store is collocated with the policy store. All Policy Servers that share a common view into the same policy store have access to the same certificates.

The following table shows the certificates that must be present in the CDS to handle your WS-Security validation requirements.

Token Type	Required Certificates
SAML Assertion; Sender-vouches	Certificate of issuing web service consumer application
SAML Assertion; Holder-of-key	Certificates of XML request subject and issuing web service consumer application.
X.509v3; Username (if signed)	Certificate of trusted issuer

Configure the WS-Security Authentication Scheme

To obtain security information from WS-Security headers in incoming XML messages, configure the WS-Security authentication scheme.

Follow these steps:

1. Click Infrastructure, Authentication.
2. Click Web Services Authentication Schemes, Create Authentication Scheme.

   The Create Authentication Scheme pane opens.

   Authentication scheme settings open.
3. Enter a name and a description for the scheme in the General section.
4. Select WS-Security from the Authentication Scheme Type list.

5. Specify a protection level.
6. In the Scheme Setup section, select one of the following required Security Token Types:
   - Username and Password Digest (the default). Also valid for Username and Password tokens (clear text).
   - X509v3 Certificate
   - SAML Assertion

   If you select Username and Password Digest or X509v3 Certificate, the XML Signature Restrictions section is displayed. If you select SAML Assertion, the SAML Token Restrictions section is displayed.
7. If you selected the Username and Password Digest or X509v3 Certificate security token type, specify how restrictions should be applied in the XML Signature Restrictions section.
8. If you selected the SAML Assertion security token type, complete the options in the SAML Token Restrictions section to specify how token restrictions should be applied.
9. (Optional) For messages with multiple WS-Security headers, specify the value of the SOAP actor (role) attribute that identifies the header element from which CA SiteMinder® Web Services Security should obtain security tokens in the SOAP Role field (located in the Advanced group box). For example:

   `http://www.example.com/soap/MySOAPRole`
10. (Optional) To prevent authentication errors caused by clock skew between token producer and consumer systems, specify the maximum allowable skew time in the Timestamp Skew Time field (located in the Advanced group box).

    Default: 30 seconds
11. Click Submit.

    The authentication scheme is saved. You can now assign it in application object components or realms.

(Optional) Strip Standard Prefixes from XPath Queries That Identify WS-Security SAML Assertion Attributes

When specifying an XPath expression to identify a SAML assertion attribute that specifies the user identity for WS-Security authentication in the Attribute Name/XPATH field, you may need to strip standard prefixes to return the attribute value itself. The XPath substring-after function provides a standard method to perform this operation.

For example, consider a SAML assertion created by the CA SiteMinder SAML Assertion Generator. This assertion contains an attribute “username” that specifies the user identify that you want to use for authentication in the following format:

`header:uid=username`

To remove the unwanted prefix, “header:uid=”, use the XPath substring-after function in the XPath query in which you specify the target attribute. For example the following Xpath query will return “username” rather than the whole string “header:uid=username”:

```
substring-after(//SMprofile/NVpair[1]/text(),"header:uid=")
```

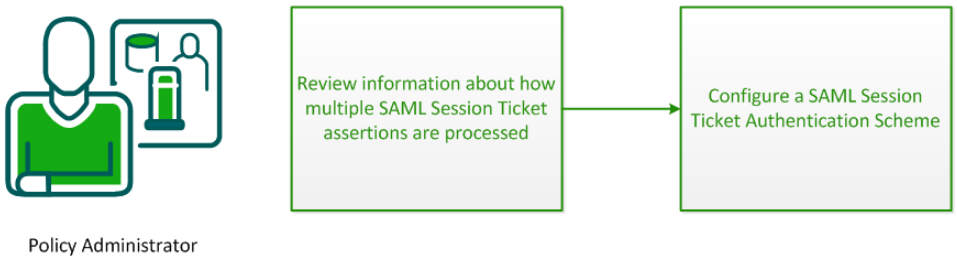
How to Configure SAML Session Ticket Authentication to Verify User Identities Obtained from SAML Session Ticket Assertions

The SAML Session Ticket authentication scheme provides a mechanism for single sign-on across web services that are protected by the same policy store. The scheme authenticates XML messages using credentials that are obtained from SAML Session Ticket assertions in an HTTP header, a SOAP envelope, or a cookie. SAML Session Tickets are strongly secure assertions that a SiteMinder WSS Agent in the same Policy Server domain generates after initial authorization of the request.

A SAML Session Ticket assertion is a data structure that contains a SiteMinder session ticket and a public key (both encrypted). The SAML Session Ticket authentication scheme uses the assertions to do the following operations:

- Verify that a valid SiteMinder session exists.
- Ensure the integrity of the signed XML document.

By including the session ticket and the public key in the assertion, a web service consumer can access web services protected by SiteMinder WSS Agents in the same Policy Server domain without being rechallenged for credentials.

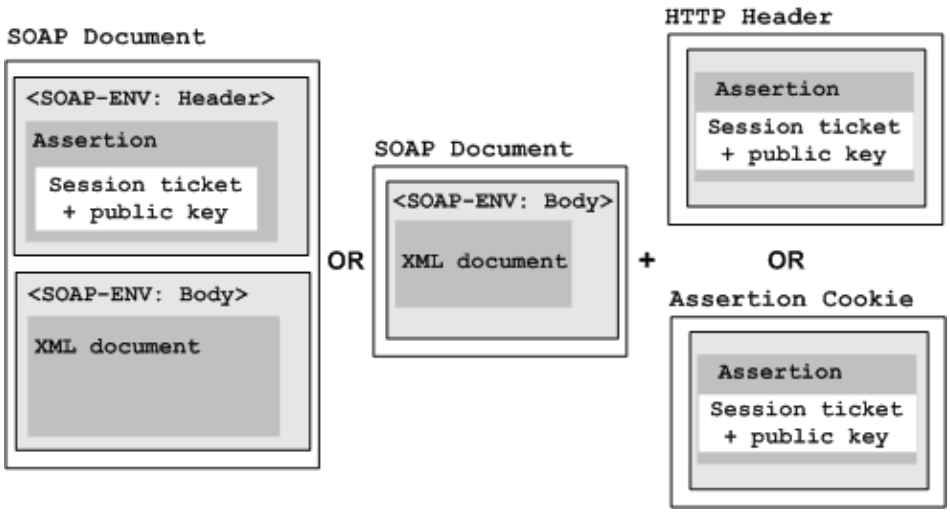


To configure CA SiteMinder® Web Services Security to validate user identities using SAML Session Ticket authentication, complete the following process:

1. Review information about how multiple SAML Session Ticket assertions are processed (see page 48)
2. Configure a SAML Session Ticket authentication scheme (see page 49)

Review Information About How Multiple SAML Session Ticket Assertions are Processed

SAML Session Ticket assertions can be placed the in a SOAP document, in an HTTP header separate from the XML document, or in a cookie as shown in the following illustration:



If a request message contains more then one associated assertion, assertions found within assertion cookies take precedence over assertions in the SOAP envelope or HTTP header. The SiteMinder WSS Agent first collects all SAML Session Ticket assertions from the cookie and the header or envelope as specified in the authentication scheme. The agent then tests each assertion until it finds the first one with a valid session ticket (that is, it can be decrypted with the agent key) and valid signatures, if they are required. Authentication is then performed using this assertion.

Note: If the authentication fails later in the authentication process because the first valid session ticket is found to be expired or revoked, authentication will fail—potential session tickets included in other assertions are *not* subsequently evaluated.

Configure a SAML Session Ticket Authentication Scheme

To obtain security information from SAML Session Ticket assertions in an HTTP header, a SOAP envelope, or a cookie that is associated with an incoming message, configure the SAML Session Ticket authentication scheme.

Follow these steps:

1. Click Infrastructure, Authentication.
2. Click Web Services Authentication Schemes, Create Authentication Scheme.

   The Create Authentication Scheme pane opens.

   Authentication scheme settings open.
3. Enter a name and a description for the scheme in the General group box.
4. Select SAML Session Ticket from the Authentication Scheme Type list.
5. Enter a protection level.
6. In the Scheme Setup group box, set the following options, as required:
   - Specify where the SAML assertion is located by selecting one of the following options:
     - Envelope Header
     - HTTP Header
   - (Optional) To require that incoming messages are signed, set the Require signature on XML message option. If this option is set, the following statements must be true for authentication to occur:
     - The document must be signed and the signature must be valid.
     - If the assertion is signed, it must also be valid.
   - (Optional) To require that incoming messages are sent over an SSL connection, set the Require Secure Transport Layer option.
7. Click Submit.

   The authentication scheme is saved. You can now assign it in application object components or realms.

Chapter 3: (Optional) Configure Responses to Generate SAML Session Tickets or WS-Security Headers for Outgoing Messages

This section contains the following topics:

Responses Overview

A response is a container for one or more response attributes which the Policy Server sends to a SiteMinder WSS Agent after processing the response. The Policy Server supports several types of response corresponding to different Agent types, each of which provides a different set of attributes, which take the form of name/value pairs.

SiteMinder WSS Agents accept only accept *Web Agent* responses, which provide attributes (name/value pairs) that can be used by SiteMinder WSS Agents and other CA SiteMinder® agents.

Policies contains rules and responses which are bound to users and user groups. In a policy, responses are bound to specific rules or rule groups. When a rule fires, the associated response returns information to a SiteMinder WSS Agent, such as user attributes, DN attributes, static text, or customized active responses. Responses can also be used to instruct a SiteMinder WSS Agent to generate WS-Security headers and SAML Session Tickets.

Response Attribute Types

CA SiteMinder® supports different types of response attributes. The types of response attributes determine where the Policy Server finds the proper values for the response attributes.

You can specify the following types of response attributes when you add response attributes to a CA SiteMinder® response:

Static

Returns data that remains constant.

Use a static attribute to return a string as part of a CA SiteMinder® response. This type of response can be used to provide information to a Web application. For example, if a group of users has specific customized content on a Web site, the static response attribute, show_button = yes could be passed to the application.

User Attribute

Returns profile information from a user's entry in a user directory.

This type of response attribute returns information associated with a user in a directory. A user attribute can be retrieved from an LDAP, WinNT, Microsoft SQL Server or Oracle user directory.

Note: In order for the Policy Server to return values from user directory attributes as response attributes, the user directories must be configured on the CA SiteMinder® User Directory pane.

DN Attribute

Returns profile information from a directory object in an LDAP, Microsoft SQL Server or Oracle user directory.

This type of response attribute is used to return information associated with directory objects to which the user is related. Groups to which a user belongs, and Organizational Units (OUs) that are part of a user DN, are examples of directory objects whose attributes can be treated as DN attributes.

For example, you can use a DN attribute to return a company division for a user, based on the user's membership in a division.

Note: In order for the Policy Server to return values from DN attributes as response attributes, the user directories must be configured on the CA SiteMinder® User Directory pane.

Active Response

Returns values from a customer supplied library that is based on the CA SiteMinder® Authorization API.

An Active Response is used to return information from an external source. An Active Response is generated by having the Policy Server invoke a function in a customer-supplied shared library. This shared library must conform to the interface specified by the Authorization API (available separately with the Software Development Kit; if installed, see the API Reference *Guide for C* for more information).

Note: It is up to you to make sure the value returned by an active response is valid. For example, if an active response returns a numeric type, the library and function must return a string whose value is a number.
When you configure a response attribute, the correct Value Type for the response attribute is displayed on the Response Attribute pane.

Variable Definition

Returns the value of the specified variable at runtime.

Select Variable Definition when you want to select and use a variable from a list of already-defined variables.

CA SiteMinder® Web Services Security Web Agent Response Attributes

In addition to the CA SiteMinder® Web Agent response attributes, CA SiteMinder® Web Services Security provides the following Web Agent response attributes that are only applicable for use with WSS Agents:

WebAgent-SAML-Session-Ticket-Variable

Provides Policy Server data that the SiteMinder WSS Agent uses to generate a SAML assertion. The data is inserted into an XML message HTTP or SOAP envelope header or a cookie (as specified by associated response attributes).

When you configure a SAML Session Ticket response, the Policy Server generates the response data. This data instructs the SiteMinder WSS Agent how to build the assertion. The SiteMinder WSS Agent encrypts a session ticket (and optionally, the public key from a web service consumer) and the response data. The agent then generates the assertion. The agent delivers the assertion to the web service. The token can only be encrypted and decrypted by the SiteMinder WSS Agent using its Agent key.

WebAgent-WS-Security-Token

Provides Policy Server data that the SiteMinder WSS Agent uses to generate WS-Security Username, X509v3, or SAML tokens (as specified by associated response attributes). These tokens are added to a SOAP message header.

When you configure a WS-Security response, the Policy Server generates the response data. This data instructs the SiteMinder WSS Agent how to build the token. The agent then generates and adds the token to the SOAP request and delivers it to the web service.

Responses and Directory Mappings

Directory mappings let you specify a separate authorization user directory in application object component or a realm. When you define a separate authorization directory, a user is authenticated based on the information contained in one directory, but authorized based on the information contained in another directory.

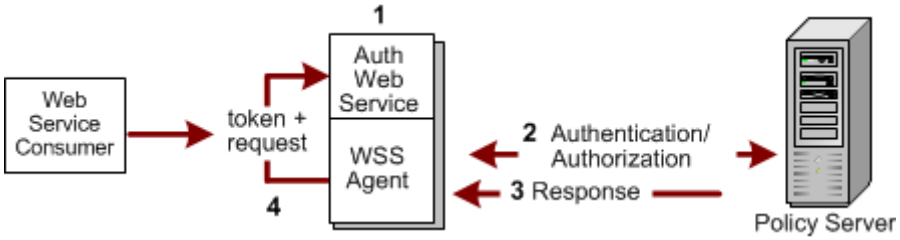
When you create a response and associate it with a authentication (OnAuth) event, any information retrieved from a user directory is retrieved from the authentication directory. If you create an authorization (OnAccess) event, any information retrieved from a user directory is retrieved from the authorization directory.

WS-Security Header Production Overview

How WS-Security Responses are Used

WS-Security responses are typically used to instruct the SiteMinder WSS Agent protecting an authentication web service to create WS-Security headers and, optionally, to perform XML encryption on those headers and the message content.

The following illustration shows the response process in such an environment.



1. A web service consumer sends a request (in the form of an XML message) to the authentication web service.
2. The SiteMinder WSS Agent obtains credentials and passes them to the Policy Server. Authentication is handled by any *supported* authentication scheme.

   Note: Although any authentication scheme can be configured to obtain credentials from a request, not every authentication scheme is suitable for creating every type of WS-Security token.
3. After the web service consumer is authenticated, the client is authorized. The policy that authorizes the consumer has a WS-Security response configured with it, which instructs the SiteMinder WSS Agent to generate WS-Security headers.
4. The SiteMinder WSS Agent generates the WS-Security headers and delivers them, together with the request message, to the authentication web service.

However, for a web service that receives requests with XML-encrypted elements, but that does not have the logic to decrypt those requests internally, WS-Security responses can be used to instruct the SiteMinder WSS Agent to pass the web service decrypted versions of those requests (see TXM_WSSEC_ENCRYPT_PUB_KEY_ROLE).

More information:

Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

How WS-Security Headers Are Produced

WS-Security headers are generated by a SiteMinder WSS Agent (or a third-party security application) after initial authorization of the request, making the WS-Security authentication scheme the ideal basis for multiple web services at federated enterprises.

For CA SiteMinder® Web Services Security to produce WS-Security headers, a web service consumer request must first be authorized by the Policy Server using an appropriate authentication scheme (not every authentication scheme obtains everything that is required from the incoming request to create any type of token). The authorizing policy must have a response configured with it that issues WS-Security response data. This data is used by the SiteMinder WSS Agent to generate WS-Security headers. These headers are inserted into the SOAP message header and delivered to the protected web service application. The web service may then pass these headers to the following locations:

- Back to web service consumer applications that can then use them in further requests to gain access to other web services protected by the WS-Security authentication scheme.
- Downstream to further web services protected by the WS-Security authentication scheme.

More information:

Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

WS-Security Token Types

CA SiteMinder® Web Services Security can produce SOAP messages with WS-Security headers containing the following supported token types:

- Username and Password Digest Token (see page 56)
- Username and Password Token (Clear Text) (see page 57)
- X509v3 Certificate Token (see page 57)
- SAML Assertion Token (see page 58)

Username and Password Digest Token

The Username and Password Digest token provides password element confidentiality without requiring channel-level security for the entire document.

The Username token includes a username and password, a cryptographic nonce (a parameter that varies with time) and, optionally, a timestamp. The password is hashed as an SHA1 digest using the nonce, timestamp, and password:

password_digest = SHA1[*nonce* + *timestamp* + *password*]

When a timestamp is included, creating SHA1 password digests provides protection against replay attacks that prevents an eavesdropper from cutting out and replaying the <wsse:UsernameToken> element in a different document at a later time. Also, hashes of the same password along with the same nonce still resolve to different digest values, assuming that the timestamp has been updated.

The Username and Password Digest Token authentication scheme provides protection against replay attacks (where an eavesdropper might cut out and replay the token at a later time) by imposing a limit (60 minutes by default) on the age of the token. That is, if a token was created more than 60 minutes ago according to its <wsu:Created> timestamp, authentication fails.

Note: The Username and Password Digest token is supported only with LDAP and ODBC-based user directories. For LDAP user directories, CA SiteMinder® Web Services Security must be configured (using the Credentials and Connection tab in the Policy Server User Interface) to connect to the user store using an LDAP administrative identity if the directory implementation requires such credentials to return the userPassword attribute. For ODBC user directories, a “password” user property must be added to the SQL query scheme used by the directory.

Note: The password storage schemes used by the Username token-generating site must be consistent with the password storage scheme used by the Username token-consuming site. For instance, if the generating site uses SHA-1 password hashes in its user directory, then the consuming site must do the same.

Username and Password Token (Clear Text)

The Username and Password token provides the token subject's username and clear-text password.

Note: The password storage schemes used by the Username and Password token-generating site must be consistent with the password storage scheme used by the Username token-consuming site. For instance, if the generating site uses SHA-1 password hashes in its user directory, then the consuming site must do the same.

Important! CA recommends that you always use Username and Password tokens with digital signatures or XML encryption to prevent malicious parties from intercepting the message and obtaining the username and password from it.

More information:

(Mandatory) Response Attribute Variable for Specifying the Generated WS-Security Token Type (see page 67)

X509v3 Certificate Token

The X509v3 certificate security token provides the token subject's X.509v3 certificate in a SOAP document.

When configured to require X509v3 certificate tokens, the WS-Security authentication scheme provides basically the same functionality as the XML Digital Signature authentication scheme, but without requiring certificate mapping, since the signature and key information are contained in standard header elements.

Using the X509v3 certificate token enables the SiteMinder WSS Agent to do the following:

- Verify the signature
- Ensure that the signature is signed by using a trusted certificate
- Confirm that the document has not been altered.

After the signature is verified, the Policy Server does the following:

- Uses the digital signature and other information in the user store entry to confirm that the XML document is actually from a trusted client.
- Checks that the web service consumer has a valid entry in the user store.

When generating X.509v3 tokens, CA SiteMinder® Web Services Security uses the host web service enterprise's certificate, which it obtains from the certificate data store (CDS).

WS-Security SAML Assertion Token

The SAML assertion token specification (see OASIS Working Draft 14, *Web Services Security: SAML Token Profile*, July 12, 2004) extends the token-independent processing model defined by the core WS-Security specification, allowing SAML assertions to be used to provide secure authentication data.

The SAML assertion includes the identity of the web service consumer (typically as its subject) and, optionally, other associated attributes. Additionally, the SAML token specification provides for the use of digital signatures to guarantee the integrity and authenticity of the SAML assertion, its issuer, and the subject of the assertion, using one of the following:

- The Enterprise certificate/public key of the assertion issuer (a trusted site where the request is authenticated and the SAML token generated) to sign the assertion and its contents.
- The Enterprise private key of the assertion subject (the organization being represented by the web service consumer making the request) or the assertion issuer to sign the SOAP document (depending on the subject confirmation method.

So, when using the WS-Security authentication scheme to authenticate requests with SAML assertion tokens, CA SiteMinder® Web Services Security validates the request, to ensure that the assertion comes from a trusted source, by authenticating the assertion subject and the assertion issuer. For example, in a multiple web service implementation using SAML tokens, CA SiteMinder® Web Services Security would validate the assertion subject (the web service consumer that made the initial web service request) and the assertion issuer (a CA SiteMinder® Web Services Security-protected authentication service configured to produce SAML WS-Security tokens).

How SAML Session Ticket Responses are Used

The SAML Session Ticket response provides the data that the SiteMinder WSS Agent uses to create an assertion. The only authentication scheme that can evaluate the assertion is the SAML Session Ticket authentication scheme.

When an XML assertion document arrives at a web service protected by the SAML Session Ticket authentication scheme, the SiteMinder WSS Agent does the following:

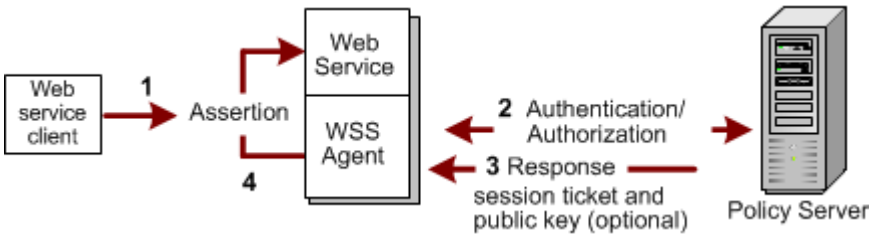
- Extracts the encrypted session ticket from the assertion
- Decrypts the session ticket using the agent key
- Signs the document using the public key from the assertion

  The binding of the session ticket and the public key ensures that the XML document is signed by a client that is authenticated by CA SiteMinder® Web Services Security and that has a valid session.

Then, if the Agent does not have the session ticket in its cache, the Policy Server validates the client with the session ticket from the assertion. If the Agent does have the session ticket in its cache, the Policy Server is not invoked.

Note: The web service that returns the assertion is not protected by the SAML Session Ticket authentication scheme. Only subsequent services in the single sign-on environment require this authentication scheme.

The following illustration shows the response process.

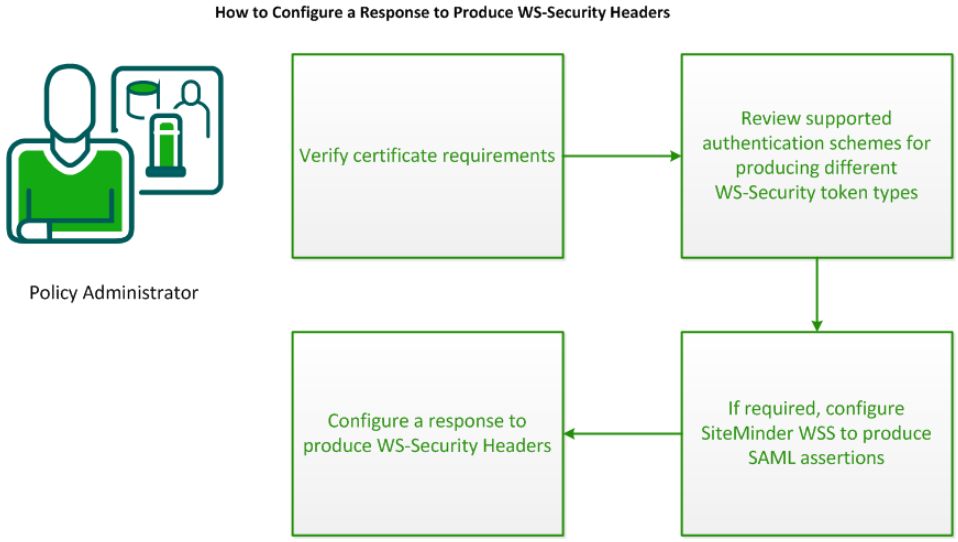


1. Client sends a request.
2. SiteMinder WSS Agent passes credentials to Policy Server. Authentication handled by any CA SiteMinder-supported authentication scheme.
3. After the client is authenticated, the client is authorized. The policy that authorizes the client has a SAML response configured with it, which generates a session ticket and, optionally, a public key.
4. SiteMinder WSS Agent generates the assertion and delivers it to the web service.

How to Configure Responses to Produce WS-Security Headers

To configure CA SiteMinder® Web Services Security to produce WS-Security headers, create a response and associate it with resources in a web service security policy. This response data is used by the SiteMinder WSS Agent to generate WS-Security headers. The SiteMinder WSS Agent then inserts these headers into the SOAP message header and delivers them to the protected web service.

Use variable types, if needed, to pass data back to the web service. Variables are resolved by the Policy Server at run time, when it generates the response.



To configure responses to produce WS-Security Headers for outgoing messages, do the following procedures:

1. Verify certificate requirements (see page 60).
2. Review supported authentication schemes for producing different WS-Security token types (see page 61).
3. If required, configure CA SiteMinder® Web Services Security to produce SAML assertions (see page 62).
4. Configure a response to produce WS-Security headers (see page 66).

More information:

Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

Verify Certificate Requirements

Before you configure a response to produce WS-Security headers, verify that the following requirements are met:

- To produce signed WS-Security tokens, verify that your enterprise private key and certificate chain are present in the certificate data store (CDS).
- To produce WS-Security X509v3 tokens, verify that your enterprise private key and certificate chain are present in the certificate data store (CDS).

Review Supported Authentication Schemes for Producing Different WS-Security Header Types

You can configure responses to produce any type of WS-Security token upon successful authorization of a request. However, not every authentication scheme gathers all the necessary information (username, clear text password, SOAP message) from an incoming request to create every type of token.

If a response is configured to create a token that requires anything that the configured authentication scheme does not provide, header creation fails. Verify that the authentication method that you plan to use is suitable to produce the WS-Security token that you want to produce in response.

The following table shows which WS-Security tokens can be produced for each authentication method.

Authentication Method	**WS-Security Token Types That Can be Produced** ***Username and Password***	***Username and Password Digest***	***SAML***	***X.509***
Basic (SiteMinder WSS Agent for Web Servers only)	No	Yes	No	No
XML-DCC	Yes	Yes	Yes	Yes
XML-DSIG	No	No	Yes	Yes
SAML Session Ticket	No	No	Yes	Yes
WS-Security Username and Password Token	Yes	Yes	Yes	Yes
WS-Security Username and Password Digest Token	No	Yes	Yes	Yes
WS-Security SAML Token	No	No	Yes	Yes
WS-Security X.509 Token	No	No	Yes	Yes
SiteMinder Session (SMSESSION) Cookie	No	No	No	No

Configure CA SiteMinder® Web Services Security to Generate SAML Assertions

If you are configuring CA SiteMinder® Web Services Security to add WS-Security headers containing SAML assertions tokens to outgoing messages, you must first perform some additional configuration in the Administrative UI

In all SAML token generation situations, you must create an Affiliate domain. Additionally, you must perform the following steps depending on the version of SAML tokens you need to generate:

- To create SAML 1.1 assertions, configure the SAML 1.x Assertion Generator properties file and add Affiliate objects to the Affiliate domain.
- To create SAML 2.0 assertions, add SAML 2.0 Service Provider objects to the Affiliate domain.

Note: Affiliate domains and related SAML token functionality are implemented as part of legacy federation on the Policy Server.

SAML 1.x Assertion Generator

If you are configuring CA SiteMinder® Web Services Security to add WS-Security tokens containing SAML 1.x assertions to SOAP documents for consumption by other web services, you must configure the SAML Assertion Generator to produce the SAML 1.1 assertions that will be used in those tokens.

Note: The SAML Assertion Generator is a component of CA SiteMinder® legacy federation on the Policy Server.

The SAML Assertion Generator uses static configuration data from two sources to determine how to construct assertions:

SAML Assertion Generator Properties File

Specifies domain-wide SAML assertion generation parameters

Affiliate Objects

Define a set of parameters for the SAML Assertion Generator

Once configured, the SAML Assertion Generator is triggered to generate an assertion when a WS-Security SAML response (which specifies the affiliate to use to generate the assertion and dynamic information about how the assertion and message should be signed) is triggered by an authorizing policy.

Configure the AMAssertionGenerator.properties File

The AMAssertionGenerator.properties file contains domain-wide configuration parameters required for generating SAML assertions.

To configure the AMAssertionGenerator.properties file for CA SiteMinder® Web Services Security

1. Navigate to the following location: *policy_server_home*/config/properties
2. Open the AMAssertionGenerator.properties file in a text editor.
3. Modify the following parameters:

   AssertionIssuerID

   Specifies the URL of the authentication web service that is issuing the assertion. Must match the Issuer DN in the enterprise certificate. This value is used for unsigned assertions. For example:

   AssertionIssuerID = http://www.acmewidget.com/ordering

   SecurityDomain

   Specifies the domain name of the enterprise issuing the assertion. For example:

   SecurityDomain = www.example.com

   SourceID

   Not used by CA SiteMinder® Web Services Security.
4. Save the file and exit the text editor.
5. Restart the Policy Server. (Changes made to the AmAssertionGenerator.properties file will not be picked up by the Policy Server until it is restarted.)

Configure Affiliate Domains and Affiliate Objects

You require an affiliate domain containing and affiliate object to configure the SAML Assertion Generator produce SAML 1.x assertions in WS-Security SAML tokens.

To configure affiliate domains and affiliate objects, follow the associated procedures in *Federation Manager Guide: Legacy Federation*. However, because CA SiteMinder® Web Services Security does not use the affiliate object to define an affiliate organization, you do not need to specify all the options.

Note: Affiliate objects configured for CA SiteMinder® Web Services Security do *not* define the affiliate organization for which the assertion is intended. Assertions generated for CA SiteMinder® Web Services Security can be sent to any web service protected by the WS-Security authentication scheme (or similarly capable third-party security application).

The following list summarizes the affiliate configuration parameters that CA SiteMinder® Web Services Security uses to generate WS-Security SAML tokens. The wizard may require you to enter values for other parameters, but these are not used by CA SiteMinder® Web Services Security.

General page

General section: Name

Specifies the name of the affiliate object (must be unique across all affiliate domains).

This name is referenced by WS-Security policy responses (by defining a txm_wssec_saml_affiliate attribute whose value matches the name of the affiliate object).

General section: Active

Activates the affiliate object.

This option must be set for CA SiteMinder® Web Services Security to produce SAML 1.x assertions.

Restrictions section: Time

(Optional) Specifies times when assertion can be issued.

Restrictions section: IP Address

(Optional) Specifies IP addresses that are allowed to generate SAML assertions.

User page

Add Members

Specifies the users and groups (from the user directory or directories defined in the affiliate domain) for whom assertions should be generated.

Assertions page

General section: Audience

(Optional) Specifies the URI of a document that describes the terms and conditions of the agreement between the token issuer and consumer. This value is added to the assertion and can be used for authentication purposes. (If a request's assertion token contains an audience value, that value must match one specified in the WS-Security scheme for the request to be authenticated.)

Additionally, the web service can parse the actual audience document to obtain additional information.

General section: Validity Duration Second(s)

(Optional) Specifies the amount of time, in seconds, that the assertion will be valid.

General section: Skew Time Second(s)

(Optional) Specifies the difference, in seconds, between the system clock time of the SAML assertion producer and the system clock time of the SAML assertion consumer.

Session section: Shared Sessioning

Not used by CA SiteMinder® Web Services Security (leave option unset).

Attributes section: Add

(Optional) If specified, an attribute statement will be included in the assertion that can be used for use in authentication and authorization decisions.

Configure SAML 2.0 Service Providers

SAML Service Provider objects define parameters used by the SAML Assertion Generator to produce SAML 2.0 assertions for use in WS-Security SAML tokens.

Note: When you configure a service provider object for use by CA SiteMinder® Web Services Security, you are *not* defining a service provider organization for which the assertion is intended. Assertions generated for CA SiteMinder® Web Services Security can be sent to any web service protected by the WS-Security authentication scheme (or similarly capable third-party security application).

To configure SAML Service Provider objects, generally follow the associated procedures in the CA SiteMinder Federation Security Services Guide. However, because CA SiteMinder® Web Services Security does not use the affiliate object to define a service provider organization, you do not need to specify all the options. Fields whose use is different for use by CA SiteMinder® Web Services Security are described below.

Name

Specifies the name of the service provider object (must be unique across all affiliate domains). This name is referenced by WS-Security policy responses (by defining a txm_wssec_saml_affiliate attribute whose value matches the name of the affiliate object).

Enabled

Sets the Enabled check box to activate the service provider object. This option must be set for CA SiteMinder® Web Services Security to produce SAML 2.0 assertions.

Authentication URL

Not used by CA SiteMinder® Web Services Security. However, a valid value *is* required. CA recommends using "http:\\localhost\"

Application URL

Not used by CA SiteMinder® Web Services Security.

SSO Tab: Bindings

Choose the HTTP-Post option.

Configure a Response to Produce WS-Security Headers

To define the properties of WS-Security headers you want CA SiteMinder® Web Services Security to produce, create a WS-Security header response in an application security policy.

Note: If you are using the domain security model, create a WS-Security header response in the web service domain.

Follow these steps:

1. Create an application object or modify an existing application object that defines the security policy for a web service.
2. Click the Response tab.
3. Click Create Response.

   The Create Application Response pane opens.
4. Type the response name in the General section.
5. Add response attributes that define the properties of the WS-Security headers, by doing the following steps:
   a. Click Create Response Attribute. The Create Response Attribute pane that opens
   b. Select the WebAgent-WS-Security-Token response attribute type from the Attribute drop-down list in the Attribute Type section.
   c. Select the attribute type (Static, User Attribute, DN Attribute, or Active Response) in the Attribute Kind section.

      The fields on the Attribute Fields group box are updated to match the specified attribute type.
   d. Specify a required name/value pair (listed in the following sections) in the Attribute Fields section. Enter values directly in the Variable Name and Variable Value fields or populate those fields with valid values from the Select a Name and Select a Value drop-down lists.

e. Specify Cache Value or Recalculate value every ... seconds on the Attribute Caching group box.

f. Click Submit.

The Create Response Attribute Task is submitted for processing, and the response attribute is added to the Attribute List on the Create Response Attribute pane.

6. Create further response attributes as required.

7. Click OK.

The Create Response Task is submitted for processing and you are returned to the Responses tab.

More information:

(Mandatory) Response Attribute Variable for Specifying the Generated WS-Security Token Type (see page 67)
Response Attribute Variables for Encrypting/Decrypting WS-Security Messages (see page 72)
Response Attribute Variables for Handling WS-Security Headers (see page 74)

(Mandatory) Response Attribute Variable for Specifying the Generated WS-Security Token Type

The TXM_WSSEC_TOKEN_TYPE variable name/value pair determines the WS-Security token type to generate. A response attribute that defines TXM_WSSEC_TOKEN_TYPE is required in all WS-Security header responses.

TXM_WSSEC_TOKEN_TYPE

Specifies the type of WS-Security token the SiteMinder WSS Agent should create and add to the WS-Security header for message authentication:

Attribute kind: Static

Variable value: One of the following:

- **password**—Creates a Username and Password Digest token, providing for password digest message authentication. For this token type, the TXM_WSSEC_USER_
PASSWORD response variable must also be configured.
- **password_nodigest**—Creates a Username and Password token, providing clear text password message authentication. For this token type, the TXM_WSSEC_USER_
PASSWORD response variable must also be configured.
- **X509**—Creates an X509v3 token.
- **SAML**—Creates a SAML token, providing for SAML message authentication. For this token type, the TXM_WSSEC_SAML_AFFILIATE response variable must also be configured.

More information:

Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

Response Attribute Variables for Generating Username and Password and X.509 Certificate Tokens

The following table describes the response variable name/value pairs for generating username and password (digest or clear text) and X.509 certificate tokens in WS-Security headers. That is, if the TXM_WSSEC_TOKEN_TYPE response attribute variable is set to password, password_nodigest, or X509.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_USER_PASSWORD (Required)	userpassword (Value most common for LDAP user directories -- if you have used a custom naming scheme for your LDAP directory, the value will be different.)	User Attribute	Specifies the LDAP query string that the SiteMinder WSS Agent uses to retrieve the web service consumer's password from the user store. This value is then placed in the token.
	Or *password*	Static	Specifies a static password value to be used in the token.
TXM_WSSEC_ROLE (Optional)	*token_role_name*	Static	Specifies the value of a SOAP role attribute that identifies the WS-Security header element containing the Username and Password or X.509 token.
TXM_WSSEC_TIMESTAMP (optional)	■ True ■ False	Static	If True, tells the agent to add a wsu:Timestamp element to the WS-Security SOAP header that specifies the time that the message was created
TXM_WSSEC_TIMESTAMP_EXPIRY (Valid only if TXM_WSSEC_TIMESTAMP is True)	*message lifespan in seconds*	Static	Tells the agent to add a wsu:Expires element to the wsu:Timestamp element in the WS-Security SOAP header. The value of the wsu:Expires element is an absolute time based on the time of message creation and the specified message lifespan.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_SIGNATURE (optional)	■ all ■ body_ts ■ body ■ headers	Static	For WS-Security tokens of type password or X509, tells the agent to retrieve the enterprise private key from the certificate data store (CDS) and use it to digitally sign all or part of the SOAP document: ■ all—the generated signature will cover the entire SOAP envelope. ■ body_ts—the generated signature will cover the SOAP body and the generated <wsu:Timestamp> element. If a timestamp response attribute is not configured, a message will be logged and the signature will cover only the SOAP body. ■ body—the generated signature will cover the SOAP body. ■ headers—the generated signature will cover the SOAP header containing the generated/modified WS-Security element. By default, tokens are signed using RSA-SHA1.
TXM_WSSEC_SIGNATURE_ALG (Valid only if TXM_WSSEC_SIGNATURE is set)	■ rsa-sha1 (default) ■ rsa-sha256	Static	■ For WS-Security tokens of type password or X509, defines the signature algorithm the agent uses to sign the part of the SOAP document defined by TXM_WSSEC_SIGNATURE.

More information:

Username and Password Digest Token (see page 56)
Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

Response Attribute Variables for Generating SAML Tokens

The following table describes the response attribute variable name/value pairs for generating SAML tokens in WS-Security headers. That is, if the TXM_WSSEC_TOKEN_TYPE response attribute variable is set to SAML.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_SAML20_ASSERTION (Required for SAML 2.0)	■ Yes ■ No (default)	Static	Specifies whether the generated SAML assertion token is SAML 2.0 compliant.
TXM_WSSEC_SAML20_SPID (required for SAML 2.0)	*SAML_20_audience_value*	Static	Specifies the value of the <saml:Audience> element in a generated SAML 2.0 assertion token.
TXM_WSSEC_SAML_AFFILIATE (Required)	*affiliate_or_service_provider_object_name*	Static	Identifies the affiliate (SAML 1.x) or service provider (SAML 2.0) object that configures how SAML assertions will be produced for inclusion in SAML tokens.
TXM_WSSEC_ SAML_ROLE (optional)	*SAML_assertion_token_role_name*	Static	Specifies the value of a SOAP role attribute that identifies the WS-Security header element containing the SAML assertion token.
TXM_WSSEC_SAML_SIG_REQUIRED	■ HK ■ SV ■ SVS	Static	Specifies how the assertion and document should be signed: ■ HK (for holder-of-key)—Only the assertion will be signed (enveloped). ■ SV (for sender-vouches with SSL-based issuer confirmation)—Both assertion and document will be signed (external). ■ SVS (for sender-vouches with signature-based issuer validation)—Assertion is explicitly signed (enveloped) in addition to SV signing. (This option is only supported for SAML 1.x assertions.) Any other value or no value results in the default—an assertion with a **bearer** confirmation method.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_ SAML_USER_CERT_SRC (Required for holder-of-key signing)	■ XMLDSIG ■ Client_Cert ■ User_Store	Static	If TXM_WSSEC_SAML_SIG_REQUIRED is set to HK, this value specifies where CA SiteMinder® Web Services Security should obtain the web service consumer's public key: ■ XMLDSIG—The public key will be retrieved from a signed request sent to a web service protected by the XML DSIG authentication scheme. ■ Client_Cert—The public key will be retrieved from SSL. ■ User_Store—the public key should be retrieved from an associated user store. If this value is set, the TXM_WSSEC_SAML_USER_CERT response variable must also be configured. **Note:** If TXM_WSSEC_SAML_SIG _REQUIRED is set to SV, this option is ignored because no user public key is required.
TXM_WSSEC_ SAML_USER_CERT (Required if web service consumer public key is obtained from a user store for signing)	usercertificate This value is the most common for LDAP user directories. If you have used a custom naming scheme for your LDAP directory, the value will be different.	User Attribute	If TXM_WSSEC_SAML_USER_ CERT_SRC is set to User_Store, specifies the LDAP query string that the SiteMinder WSS Agent uses to retrieve the web service consumer's public key from the user store for signing SAML assertion tokens. **Note:** CA SiteMinder® Web Services Security automatically completes the query string using the value you specify.
TXM_WSSEC_ SAML_TIMESTAMP (optional)	■ True ■ False (default)	Static	A value of True causes a timestamp to be generated for use in SAML assertions. **Note:** If TXM_WSSEC_SAML_SIG _REQUIRED is set to SV or SVS, the timestamp is signed.
TXM_WSSEC_ SAML_TIMESTAMP _EXPIRY (optional)	*message _lifespan_in_seconds*	Static	Tells the agent to add an expiry element to the timestamp used in SAML assertions. The value of this expiry element is an absolute time based on the time of assertion creation and the specified message lifespan.

More information:

WS-Security SAML Assertion Token (see page 58)
Configure CA SiteMinder® Web Services Security to Generate SAML Assertions (see page 62)
Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

Response Attribute Variables for Encrypting/Decrypting WS-Security Messages

The following table describes response attribute variable name/value pairs that can be configured to tell the SiteMinder WSS Agent to encrypt message elements or to pass a decrypted version of a message to the recipient web service.

Note: There are two versions of each XML encryption-related name/value pair—use the former for use with messages with username/password or X.509 tokens, use the latter for use with messages with SAML tokens.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_ENCRYPT_PUB_KEY_ROLE or TXM_WSSEC_SAML_ENCRYPT_PUB_KEY_ROLE (required)	*name_of_WS-Security_token_consumer*	Static	Specifies the value of a SOAP role attribute that identifies the WS-Security header element containing the recipient's X.509 certificate. The public key in this certificate is used to encrypt the symmetric key. The corresponding private key must be held by the intended message recipient. This element is *required*. If no role is specified, the variable must be declared with a null value; CA SiteMinder® Web Services Security will then obtain the key in the WS-Security header with no role, of which only one is allowed.
TXM_WSSEC_ENCRYPT_DECRYPT or TXM_WSSEC_SAML_ENCRYPT_DECRYPT	■ True ■ False (default)	Static	Specifies whether the SiteMinder WSS Agent should pass an incoming encrypted message to the web service in its encrypted or decrypted form. If True, the SiteMinder WSS Agent will replace the current message with the decrypted version of the message, if available.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_ENCRYPT_ELEMENT or TXM_WSSEC_SAML_ENCRYPT_ELEMENT	■ UsernameToken ■ Assertion ■ Body	Static	Identifies the message element to be encrypted. You should add one such name value/pair for *each* element you want encrypted. For example, configure one name/value pair for the message body and one name/value pair for the token. For TXM_WSSEC_ENCRYPT_ELEMENT: If **UsernameToken**, Username and Password and Username and Password Digest tokens will be encrypted. If **Body**, the message body will be encrypted. For TXM_WSSEC_SAML_ENCRYPT_ELEMENT: If **Assertion**, SAML assertion token will be encrypted. If **Body**, the message body will be encrypted.
TXM_WSSEC_ENCRYPT_OR_SIGN_FIRST or TXM_WSSEC_SAML_ENCRYPT_OR_SIGN_FIRST	■ Sign (default) ■ Encrypt	Static	Indicates whether encryption or signing should be performed first.
TXM_WSSEC_ENCRYPT_ALG_KEY or TXM_WSSEC_SAML_ENCRYPT_ALG_KEY	■ rsa-1_5 (default) ■ rsa_oaep	Static	Indicates the encryption algorithm to use to encrypt the symmetric encryption key.
TXM_WSSEC_ENCRYPT_ALG_DATA or TXM_WSSEC_SAML_ENCRYPT_ALG_DATA	■ tripledes-cbc (default) ■ aes128-cbc ■ aes256-cbc ■ aes192-cbc	Static	Indicates the encryption algorithm to use to encrypt the data element or elements that have been specified using TXM_WSSEC_ENCRYPT[_SAML]_ELEMENT variables.

Response Attribute Variables for Handling WS-Security Headers

The following table describes response attribute variable name/value pairs that can be configured to tell the SiteMinder WSS Agent how to handle the mustUnderstand attribute in WS-Security request , or to remove the mustUnderstand or the entire security token from messages passed to the recipient web service.

Variable Name	Variable Value	Attribute Type	Meaning
TXM_WSSEC_MUST_UNDERSTAND (optional)	■ True ■ False ■ Not set (default)	Static	Determines how the SiteMinder WSS Agent behaves with respect to the mustUnderstand attribute when consuming and producing messages. For specifics of the behavior of this variable when consuming and producing headers, see TXM_WSSEC_MUST_UNDERSTAND Response Variable Effect Detail (see page 74).
TXM_WSSEC_REMOVE (optional)	■ True ■ False (default)	Static	Determines whether the SiteMinder WSS Agent removes WS-Security headers from messages. ■ If True, the WS-Security header for a specified actor/role (or the default header if no actor/role is defined) is removed from the inbound document. This setting effectively scrubs credentials. ■ (Default) If False, no change is made to the request document.

More information:

TXM_WSSEC_MUST_UNDERSTAND Response Variable Effect Detail (see page 74)

TXM_WSSEC_MUST_UNDERSTAND Response Variable Effect Detail

The following table describes the exact behavior of the SiteMinder WSS Agent when consuming and producing WS-Security tokens with different values of the TXM_WSSEC_MUST_UNDERSTAND response variable name/value pair.

TXM_WSSEC_MUST_UNDERSTAND Value	Behavior when Consuming WS-Security Tokens	Behavior when Producing WS-Security Tokens
Not set	If present in the WS-Security header, leaves mustUnderstand="1".	Places mustUnderstand="1" in the generated WS-Security header
False	If present in the WS-Security header, removes mustUnderstand="1"	Does not place mustUnderstand="1" in the generated WS-Security header

TXM_WSSEC_MUST_UNDERSTAND Value	Behavior when Consuming WS-Security Tokens	Behavior when Producing WS-Security Tokens
True	If present in the WS-Security header, leaves mustUnderstand="1".	Places mustUnderstand="1" in the generated WS-Security header

WS-Security Response Examples

The following examples show how you can use WS-Security responses.

Example 1

This example shows how to create a response that generates a Username and Password Digest token and uses the enterprise private key to digitally sign the message's SOAP envelope.

The following table shows the response attributes you must add to the response (all attributes are of type WebAgent-WS-Security-Token):

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_TOKEN_TYPE	password	Static
TXM_WSSEC_USER_PASSWORD	userpassword	User Attribute
TXM_WSSEC_SIGNATURE	all	Static

Example 2

This example shows how to create a response that generates an X509v3 token and uses the enterprise private key to digitally sign the message's SOAP envelope.

The following table shows the response attributes you must add to the response (all attributes are of type WebAgent-WS-Security-Token):

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_TOKEN_TYPE	X509	Static

Example 3

This example shows how to create a response that generates a SAML assertion token using the holder-of-key subject confirmation method, retrieving the subject's public key from an associated user store.

The following table shows the response attributes you must add to the response (all attributes are of type WebAgent-WS-Security-Token):

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_TOKEN_TYPE	SAML	Static
TXM_WSSEC_SAML_AFFILIATE	affiliate1	Static
TXM_WSSEC_SAML_SIG _REQUIRED	hk	Static
TXM_WSSEC_SAML_USER_CERT_SRC	User_Store	Static
TXM_WSSEC_SAML_USER_CERT	usercertificate	User attribute

Example 4

This example shows how to create a response that encrypts an incoming document and deliver the encrypted document to the web service.

The response generates a SAML assertion token using the sender vouches subject confirmation method and encrypts the SAML assertion and message body. The token and other related information are placed in a WS-Security header identified by the SOAP actor/role samlrole.

The SAML assertion and the message body are encrypted using the public key certificate found in the WS-Security header with the role pubkeyrole. The rsa-1_5 algorithm should be used to encrypt the symmetric encryption key; the tripledes-cbc algorithm should be used to encrypt the assertion and body data.

The document should be signed before encryption; the document and assertion should also be signed with a sender-vouches signature.

The following table shows the response attributes you must add to the response (all attributes are of type WebAgent-WS-Security-Token):

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_TOKEN_TYPE	SAML	Static

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_SAML_AFFILIATE	affiliate2	Static
TXM_WSSEC_SAML_ROLE	samlrole	Static
TXM_WSSEC_SAML_SIG _REQUIRED	sv	Static
TXM_WSSEC_SAML_ENCRYPT_PUB_KEY _ROLE	pubkeyrole	Static
TXM_WSSEC_SAML_ENCRYPT_ALG_KEY	rsa-1_5	Static
TXM_WSSEC_SAML_ENCRYPT_ALG_DAT A	tripledes-cbc	Static
TXM_WSSEC_SAML_ENCRYPT_ELEMENT	Assertion	Static
TXM_WSSEC_SAML_ENCRYPT_ELEMENT	Body	Static
TXM_WSSEC_SAML_ENCRYPT_OR_SIGN _FIRST	sign	Static

Example 5

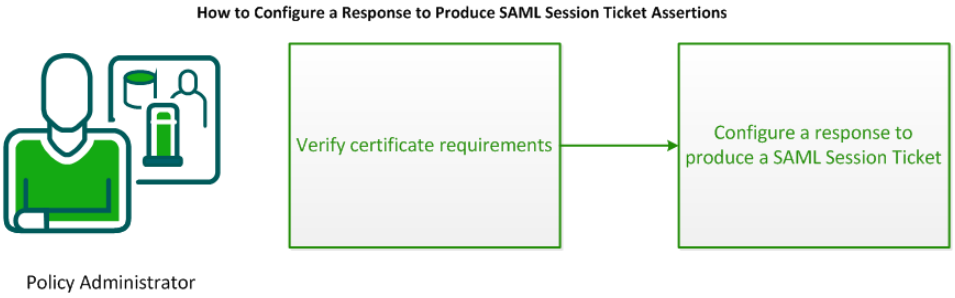
This example shows how to create a response that decrypts an incoming encrypted message and passes it to the associated web service in a message with a SAML assertion token.

Variable Name	Variable Value	Attribute Type
TXM_WSSEC_TOKEN_TYPE	SAML	Static
TXM_WSSEC_SAML_AFFILIATE	affiliate2	Static
TXM_WSSEC_SAML_ENCRYPT _DECRYPT	yes	Static

How to Configure Responses to Produce SAML Session Tickets

To configure CA SiteMinder® Web Services Security to produce SAML Session Tickets, create a response and associate it with resources in a web service authorization policy. This response data is used by the SiteMinder WSS Agent to generate SAML Session Tickets. The SiteMinder WSS Agent then delivers the SAML Session Ticket to the protected web service.

Use variable types, if needed, to pass data back to the web service. Variables are resolved by the Policy Server at run time, when it generates the response.



To configure responses to produce WS-Security Headers for outgoing messages, do the following procedures:

1. Verify certificate requirements (see page 78).
2. Configure a response to produce SAML Session Tickets (see page 78).

More information:

Review Supported Authentication Schemes for Producing Different WS-Security Header Types (see page 61)

Verify Certificate Requirements

If the public keys used in assertions are going to be stored in the user directory, define an attribute in your directory to store these public keys, and make sure it is available to the Policy Server.

Note: This is not required if the public key is included in the client's submitted XML document or obtained from a certificate over the SSL link.

Configure a Response to Produce a SAML Session Ticket

To define the properties of the SAML Session Ticket you want CA SiteMinder® Web Services Security to produce, create a SAML Session Ticket response in an application security policy.

Note: If you are using the domain security model, create a SAML Session Ticket response in the web service domain.

Follow these steps:

1. Create an application object or modify an existing application object that defines the security policy for a web service.
2. Click the Response tab.
3. Click Create Response.

   The Create Application Response pane opens.
4. Type the response name in the General section.
5. Add response attributes that define the properties of the SAML Session Ticket, by doing the following steps:
   a. Click Create Response Attribute. The Create Response Attribute pane that opens
   b. Select the WebAgent-SAML-Session-Ticket response attribute type from the Attribute drop-down list in the Attribute Type section.
   c. Select the attribute type (Static, User Attribute, DN Attribute, or Active Response) in the Attribute Kind section.

      The fields on the Attribute Fields group box are updated to match the specified attribute type.
   d. Specify a required name/value pair (listed in the following sections) in the Attribute Fields section. Enter values directly in the Variable Name and Variable Value fields or populate those fields with valid values from the Select a Name and Select a Value drop-down lists.
   e. Specify Cache Value or Recalculate value every ... seconds on the Attribute Caching group box.
   f. Click Submit.

      The Create Response Attribute Task is submitted for processing, and the response attribute is added to the Attribute List on the Create Response Attribute pane.
6. Create further response attributes as required.
7. Click OK.

   The Create Response Task is submitted for processing and you are returned to the Responses tab.

More information:

(Mandatory) Response Attribute Variable for Specifying the Generated WS-Security Token Type (see page 67)
Response Attribute Variables for Encrypting/Decrypting WS-Security Messages (see page 72)
Response Attribute Variables for Handling WS-Security Headers (see page 74)

SAML Session Ticket Response Attribute Variables

The following table lists the response attribute variable name/value pairs specific to the WebAgent-SAML-Session-Ticket-Variable attribute. You can use these variables to build assertions.

Note: You can configure other response variables with the SAML Session Ticket attribute; however, they are ignored by CA SiteMinder® Web Services Security for the assertion and are handled as standard response attributes by CA SiteMinder®.

Variable Name	Variable Value	Attribute Kind	Meaning
TXM_SAML_Location (required)	■ Envelope_Header (default) ■ HTTP_Header ■ Cookie_Header	Static	Instructs the SiteMinder WSS Agent to insert the assertion into the SOAP envelope message header, an HTTP header, or a cookie header. If Envelope_Header is the value, the client must provide an XML message for the assertion. If HTTP_Header is the value, an HTTP header named tmsamlsessionticket is added to the HTTP headers delivered to the web service If Cookie_Header is the value, the assertion is inserted into a cookie named tmsamlsession and returned to the caller in an HTTP Set-Cookie header. The cookie can also be read by the web service application at the URI protected by CA SiteMinder® Web Services Security. **Note:** Do not attempt to place more than one signature in a cookie—a 4 KB limit on the size of cookies that can be returned by the SiteMinder WSS Agent results in *no* cookie being generated if it would be greater than 4KB.

Variable Name	Variable Value	Attribute Kind	Meaning
TXM_Force_Logon (optional)	Yes or No	Static	Forces the client to authenticate using the authentication scheme for the target realm. This variable is useful if a client tries to get an assertion when logging on with only a cookie. The client is allowed access to the web service, but does not receive an assertion because the client has only a cookie. To inform the user that they have to logoff and then get rechallenged to obtain the assertion, the web service can be set up to redirect the client to a log-off URI. The user can then come back to the web service and be challenged again to obtain the assertion. **Note:** To find out how to set up a log-off URI, see the *Web Agent Configuration Guide*.
TXM_Issuer (optional)	URI	Static	Indicates the issuer of the assertion. Value is placed in the issuer URI field in the generated assertion. If the assertion is sent to a third party, the third party can use this variable to validate the assertion by sending it back to the specified URI.
TXM_Namequalifier (optional)	Domain name	Static	Indicates the domain name of the subject of the assertion.
TXM_Sign (optional)	Yes or No	Static	Tells the SiteMinder WSS Agent to sign the SOAP document payload with the private key dynamically generated by the Policy Server. **NOTE**: If you use this variable, do not use the TXM_Public_Key variable.
TXM_Sign_Assertion (optional)	Yes or No	Static	Tells the SiteMinder WSS Agent to sign the assertion that is part of the SOAP document. This ensures that no one can alter the assertion.

Variable Name	Variable Value	Attribute Kind	Meaning
TXM_Public_Key (optional)	■ XMLDSIG ■ Client_Cert ■ User_Store	Static	Tells the SiteMinder WSS Agent where to get the public key that it binds to the session ticket. **XMLDSIG**—Tells SiteMinder WSS Agent to get the key from the document with the digital certificate. (Web service must be protected by the XML Digital Signature authentication scheme.) **Client_Cert**—Indicates the client certificate sent over the SSL connection **User_Store**—Tells SiteMinder WSS Agent to get the key from the user store. **Note:** Do not use this variable with TXM_Sign.
TXM_User_Cert LDAP user directories only (optional)	usercertificate This value is the most common for LDAP user directories. If you have used a custom naming scheme for your LDAP directory, the value will be different.	User Attribute	Specifies the LDAP query string that the SiteMinder WSS Agent uses to retrieve the public key from the user store. This variable is required when TXM_Public_Key is set to User_Store.

Note: Do not use the SAML assertion, XML Body, XML Agent, and XML Envelope Header variables that you can choose from the Variables policy object in a policy domain. These variables are for use exclusively in policy expressions, not with the SAML Session Ticket response.

Enter these variables by typing the name and value in the appropriate fields in the Response Attribute dialog.

SAML Session Ticket Response Examples

You can use assertion variables to help the SiteMinder WSS Agent build the assertion.

Example 1

If the web service is protected by the XML-DSIG authentication scheme, create an attribute that extracts the client's public key from the certificate and adds it to the SAML assertion. To instruct the SiteMinder WSS Agent to get the public key from the digital certificate, enter the variable TXM_Public_Key with the value XMLDSIG.

The following table shows the properties of the primary response attribute:

Field	Value
Attribute	WebAgent-SAML-Session-Ticket-Variable
Attribute Kind	Static
Variable Name	TXM_Public_Key
Variable Value	XMLDSIG

If the public key is coming from the user directory, two response attributes are required. The properties of the first required response attribute would be as follows:

Field	Value
Attribute	WebAgent-SAML-Session-Ticket-Variable
Attribute Kind	User Attribute
Variable Name	TXM_User_Cert
Variable Value	usercertificate

The properties of the second required response attribute would be as follows:

Field	Value
Attribute	WebAgent-SAML-Session-Ticket-Variable
Attribute Kind	Static
Variable Name	TXM_Public_Key
Variable Value	User_Store

Example 2

To ensure that the assertion is placed in the SOAP envelope message header, the properties of the required response attribute would be as follows:

Field	Value
Attribute	WebAgent-SAML-Session-Ticket-Variable
Attribute Kind	Static
Variable Name	TXM_SAML_Location

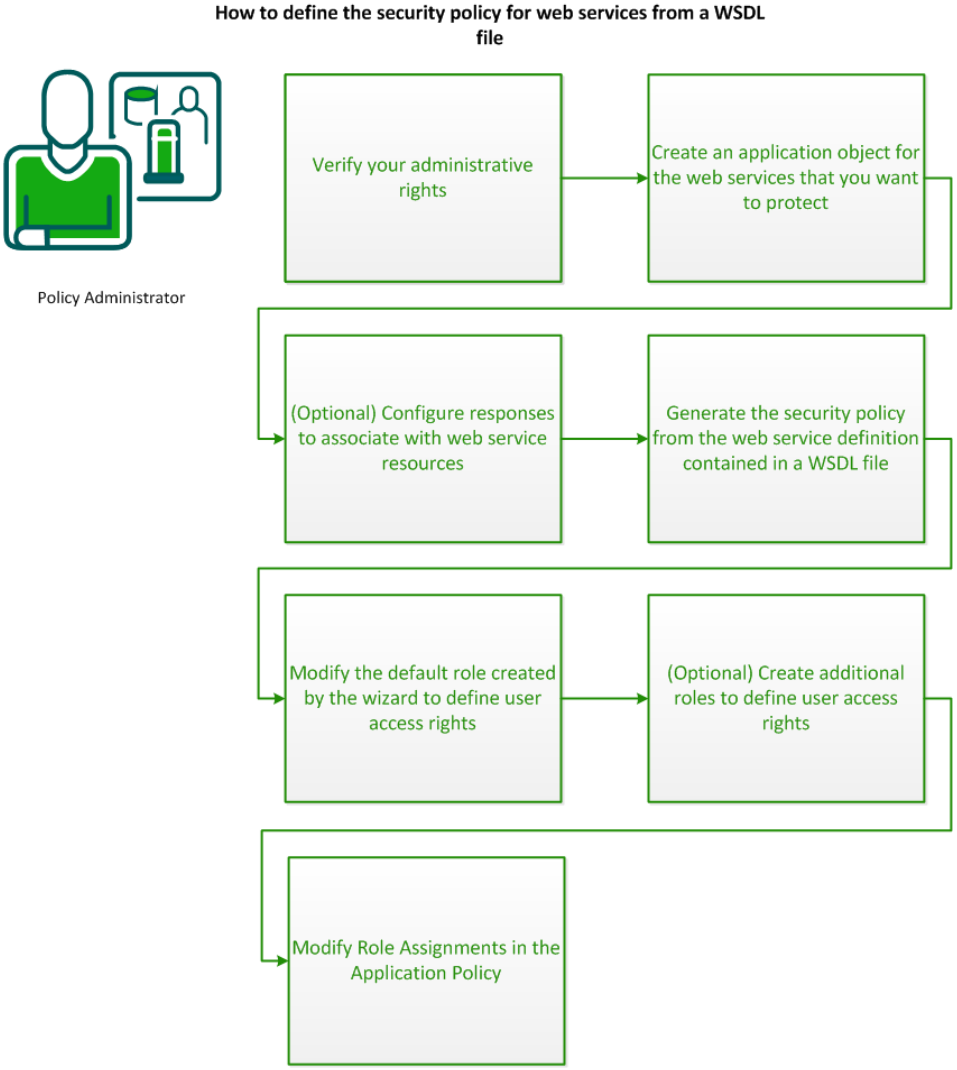
Field	Value
Variable Value	Envelope_Header

Chapter 4: How to Define the Security Policy for One or More Related Web Services from a WSDL File

To protect web services in your organization, you create application security policies. These policies define the resources you want protected and specify who is allowed access to the protected application.

Application objects provide an intuitive method of defining a complete security policy for one or more related web services. Application objects associate resources with user *roles* to specify entitlement policies that determine what web service users can access what web service application resources. Roles identify the set of users who have access to a resource or group of resources in terms of a named or unnamed expression.

This scenario describes how a policy administrator defines the security policy for web service resources from their associated Web Service Definition Language (WSDL) files.



To define the security policy for one or more related web services from a WSDL file, do the following procedures:

1. Verify your administrative rights (see page 87).
2. Create an application object for the web service resources that you want to protect (see page 87).
3. (Optional) Configure responses to associate with web service resources (see page 89).
4. Generate the security policy from the web service definition contained in a WSDL file (see page 90).

5. Modify the default role created by the wizard to define user access rights (see page 92).
6. (Optional) Create additional roles to define user access rights (see page 93).
7. Repeat Steps 4, 5, and 6 for any additional web services defined in other WSDL files that you want to protect in the same application.
8. Modify role assignments in the security policy (see page 94).

Verify Your Administrative Rights

To implement application security policies, you require the necessary administrative rights. An administrator can be assigned the following application-related rights:

Application administration

The application administration right lets you create, modify, and delete an application and its components.

Policy administration

The policy administration right lets you define the resources, roles, and policies that are associated with an application.

If you do not have the necessary rights, contact the CA SiteMinder® superuser.

Create an Application Object for the Web Services That You Want to Protect

The application object you create for one or more related web services must specify the top-level location of the resources that you want to protect, and a directory of users who are authorized to use the resources.

To identify the application and select the directory server

1. Log in to the Administrative UI
2. Click Policies, Application
3. Click Applications.
4. Click Create Application.

   The Create Application pane opens.

5. Enter values for the fields in the General group box. Choose distinctive values that help you remember its purpose or function, as shown in the following examples:

   Name

   Name of the application

   Description

   (Optional) A description of the application.

6. In the Components group box, specify values for a default component description.

   Note: These fields are mandatory, but the component they define is not used; component definitions for your web services will be created from their WSDL files.

   Agent Type

   Web Agent

   Agent

   Any SiteMinder WSS Agent.

   Resource Filter

   *

7. Accept the defaults for the remaining settings.
8. In the User Directories group box, click Add/Remove.

   The Choose User Directories dialog opens.

9. Select one or more directories that contain the the users that you want to be access the web service resources then click the right arrow to move the selected directory or directories from the Available members column to the Selected Members column.
10. Click OK.

    You return to the General tab.

11. Click Submit.

    The application is identified and the directory selected.

(Optional) Configure Responses to Associate With Web Service Resources

To include a response (for example, to generate WS-Security headers) in the application security policy you generate from a WSDL file, first configure the response.

Follow these steps:

1. Log in to the Administrative UI
2. Open the application object that defines the security policy for web service resources in an editable state.
3. Click the Response tab.
4. Click Create Response.

   The Create Application Response pane opens.
5. Type the name of the response in the General group box.
6. Click Create Response Attribute to create a response attribute, then complete the following steps on the Create Response Attribute pane that opens:
   a. Select a response attribute type from the Attribute drop-down list in the Attribute Type section.

      To configure a response to produce WS-Security headers, select WebAgent-WS-Security-Token. To configure a response to produce SAML Session Tickets, select WebAgent-SAML-Session-Ticket-Variable.

   b. Select an attribute type in the Attribute Kind (one of Static, User Attribute, DN Attribute, and Active Response) section.

      The fields on the Attribute Fields group box are updated to match the specified attribute type.

   c. Complete the fields in the Attribute Fields section to specify required Variable Name/Variable Value pairs.

      Note: For WebAgent-SAML-Session-Ticket-Variable and WebAgent-WS-Security-Token attributes, you can either enter values directly in the Variable Name and Variable Value fields or populate those fields with valid values from the Select a Name and Select a Value lists that appear.

d. Specify Cache Value or Recalculate value every ... seconds on the Attribute Caching group box.

e. Click Submit.

The Create Response Attribute Task is submitted for processing, and the response attribute is added to the Attribute List on the Create Response Attribute pane.

7. Create further response attributes as required.
8. Click OK.

The Create Response Task is submitted for processing and you are returned to the Responses tab.

More information:

How to Configure Responses to Produce WS-Security Headers (see page 59)
How to Configure Responses to Produce SAML Session Tickets (see page 77)

Generate the Security Policy from the Web Service Definition Contained in a WSDL File

After you create the application object, you generate the security policy to protect web service resources from their associated WSDL file.

Follow these steps:

1. Log in to the Administrative UI
2. Click Policies, Application.
3. Click Secure Web Services from WSDL.

The Secure Web Services from WSDL: Select Application pane appears.

4. Select the application to secure from the Choose an Existing Application list.
5. Click Next.

The Secure Web Services from WSDL: Input WSDL pane appears.

6. Specify whether you want to open a WSDL file that resides on your local system or at a specific URL by selecting the File or URL option, and identifying the file accordingly as follows:

- If you chose FILE, click Browse and navigate to its location.
- If you chose URL, type the URL.in the Enter the WSDL URL field. For example,

http://example.com/WSDL/my-wsdl.wsdl

7. Click Next.

   The Secure Web Services from WSDL: Define Policies pane appears, displaying a selectable table of the web services (ports) defined in the WSDL file.

8. Define the web service or services to protect in the Define Web Service Protection Policy table:

   - Select the web service or services that you want to protect in the Port Name column.
   - Assign the SiteMinder WSS Agent that will protect each protected web service from the Agent list.
   - Assign an authentication scheme to use to protect each web service from the Authentication Scheme list.
   - (Optional) Choose a response to bind to a web service from the Response list.

9. (Optional) Set the Propagate Authentication Scheme of Web Service to all its operations option to apply the authentication scheme you assigned to protect each web service to all of its constituent operations.

10. Click on a web service entry in the Port Name column to drill down to see its constituent operations in the Define Web Service Protection Policy table and select individual operations to protect, authentication schemes to use, and optionally, response bindings.

    (To return to the top-level WSDL view, click the All Web Services link at the top-left corner of the table.)

11. When your policy definitions are complete, click Next.

    The Secure Web Services from WSDL: Summary pane opens, displaying a summary of the components, subcomponents, and resources that will be created according to your selections.

12. If the summary is correct, click Finish.

    The Administrative UI creates component and resource definitions corresponding to your settings for all specified web service ports and operations, a default application role (that defines *no* user access), and a security policy that binds that default role with resources.

    However, if you assigned different authentication schemes to a web service port and any of its operations, you must manually create a resource definition for that web service port:

    a. Click Policies, Application, Modify Application.

       The Modify Application pane opens

    b. Specify search criteria, and click Search.

       A list of applications that match the search criteria opens.

c. Select your application from the list, and click Select.

   The Modify Object: *Name* pane opens.

d. Click on the Resources Tab.

e. Choose the appropriate entry for the web service port from the Select a context root pulldown. No resources should be listed.

f. Click Create.

   The Create Application Resource pane opens.

   Specifiy a name for the resource, accept the default resource filter (/*) and select the ProcessSOAP and ProcessXML Web Agent actions.

g. Click OK.

h. Click Submit.

The Administrative UI creates component and resource definitions corresponding to your settings for all specified web service ports and operations, a default role (that defines *no* user access), and a security policy that binds that default role with resources.

The web services you chose to protect are now secure. No access requests will be authorized until you modify the default role to define access privileges or create more roles and bind them to resources in the authorization policy.

Note: You can repeat this procedure to add the resources from multiple WSDL files to the same application. However, the Secure Web Services from WSDL operation is only intended for *initial* generation of policy objects from a particular WSDL file; if a web service changes or you must enable other operations from a previously loaded WSDL file you must delete the previously created application or edit it manually.

Modify the Default Role Created By the Wizard to Define User Access Rights

Roles associate resources with groups of users must be created.The Secure Web Services from WSDL wizard creates a default role that allows no access when it secures web services from a WSDL file. You must modify this role to define a group of users that can access a resource to which the role is assigned.

To create a new role

1. Log in to the Administrative UI
2. Click Policies, Application,
3. Click Applications.
4. Click Modify Application.

   The Modify Application pane opens
5. Specify search criteria, and click Search.

   A list of applications that match the search criteria opens.
6. Select your application from the list, and click Select.

   The Modify Object: *Name* pane opens.
7. Click the Roles tab.
8. Click the Edit button beside the default role.
9. Ensure the Create a new object of type Role button is selected, and then click OK.

   The Modify Role pane opens.
10. Define the groups, organizations, and user attribute expressions that define the members of the role by making selections in the Users Setup group box.
11. Click OK.

    The role is modified.

Create Additional Roles to Define User Access Rights

Roles associate resources with groups of users must be created. The Secure Web Services from WSDL wizard creates a default role which is assigned to all resources in when it secures web services from a WSDL file. If required, you can create additional roles.

To create a new role

1. Log in to the Administrative UI
2. Click Policies, Application
3. Click Applications.
4. Click Modify Application.

   The Modify Application pane opens
5. Specify search criteria, and click Search.

   A list of applications that match the search criteria opens.

6. Select your application from the list, and click Select.

   The Modify Object: *Name* pane opens.

7. Click the Roles tab.
8. Click Create Role.
9. Verify that the Create a new object of type Role button is selected, and then click OK.
10. Enter a name and optionally, a description for the role.
11. Specify whether the role applies to All Users or Selected Users in the configured user directories.

    Note: The Users Setup and Advanced sections do not apply when the All Users option is set and are no longer displayed.

12. Define the groups, organizations, and user attribute expressions that define the members of the role by making selections in the Users Setup group box.
13. Click OK.

    The role is created.

14. Repeat steps 8 through 13 for each additional required role.

Modify Role Assignments in the Security Policy

The Secure Web Services from WSDL wizard generates an application security policy that binds the web service resources specified in a WSDL to a default. You can modify this policy to change the roles assigned to resources to allow different groups of users to access different resources protected by the application.

Follow these steps:

1. Log in to the Administrative UI
2. Click Policies, Application
3. Click Applications.
4. Click Modify Application.
5. Specify search criteria, and click Search.

   A list of applications that match the search criteria opens.

6. Select your application from the list, and click Select.

   The Modify Object: *Name* pane opens.

7. Click the Policies tab.

   The Policies pane opens and displays a table listing the configured resources and available roles. This table lets you quickly see which roles can be granted access to which resources.

8. Place or remove checks in the role column to set the required role assignments for each web service resource.

   For example, if you had a human resources application that secures a web service for benefits management and another for performance appraisals and separate roles for employees and managers, you could:

   a. Check the Employees role beside the rows of resources that protect the benefits management operations to create a policy that allows employees to manage their benefits.

   b. Check the Managers beside the rows of resources that protect the performance appraisals to create a policy that allows only managers to access the performance appraisals web service.

9. Click Submit.

   Security policies are created for each role assigned.

Note: If you need to edit resources or roles, you must make the changes on the respective tabs and not on the Policies pane.

Chapter 5: Configure Security Policies Using Domain-based Policy Management

This section contains the following topics:

Domain-based Policy Management Overview

Domain-based policy management using policy domains and domain objects allows you to perform manual configuration of security policies for web service resources. Domain-based policy management is required to create policies that implement content-based authorization using variables.

How to Identify a Web Service Resource by Agent, Realm, and Rule

The Resource field in a CA SiteMinder® Web Services Security rule specifies the resource that is the subject of the rule. The complete resource specification (shown by the Effective Resource field on the Rule dialog box) is a concatenation of the values of the Agent, the Resource Filter of the parent realm (or realms in a nested realm environment), and the Resource field of the rule itself:

[*agent*] [*realm_resource_filter*] [*rule_resource*]

agent

Specifies a SiteMinder WSS Agent that monitors a server or gateway that contains one or more realms of protected web service resources.

realm_resource_filter

Specifies a string that specifies the resources covered by the realm. If the realm is a top-level realm, specify the resources relative to the server that serves up the files or application. If the realm is nested, specify the resources relative to the parent realm.

rule_resource

Specifies a string or regular expression that specifies the resources to which the rule applies. Specify the resources relative to the realm containing the resource. You can use wildcards (for example, "*") to broaden the specification of a rule.

How a SiteMinder WSS Agent for Web Servers Identifies Web Service Resources

By default, the SiteMinder WSS Agent for Web Servers identifies a web service being requested by extracting the binding URL and name of the web service and concatenating them as follows:

[*agent*] [/*web_service_URL*] [/*web_service_name*]

However, the SiteMinder WSS Agent for Web Servers can be configured to perform fine-grain resource identification, in which case it additionally identifies the web service operation being requested:

[*agent*] [/*web_service_URL*] [/*web_service_name*] [/*web_service_operation*]

How Other SiteMinder WSS Agent Types Identify Web Service Resources

This topic describes how the following SiteMinder WSS Agent types identify web service resources:

- SiteMinder WSS Agent for IBM WebSphere
- SiteMinder WSS Agent for Oracle WebLogic
- SiteMinder Agent for JBoss SiteMinder WSS Agent Security Interceptor

If a request is received over HTTP(S) transport, these SiteMinder WSS Agent types identify the web services being requested by extracting the binding URL, the name of the web service, and the name of the web service operation and concatenating them as follows:

[*agent*] [/*web_service_URL*] [/*web_service_name*] [/*web_service_operation*]

If a request is received over JMS transport, these SiteMinder WSS Agent types identify the web services being requested by extracting the JMS queue or topic name and the name of the web service operation and concatenating them as follows

[*agent*] [/*queue_or_topic_name]* [/*web_service_operation*]

Resource Identification Policy Examples

Coarse-Grain Resource Identification Over HTTP Example

Say you want to protect a resource with the following properties.

- The resource is hosted on an IIS web server on host **soap** in domain **example.com** that is protected by a SiteMinder WSS Agent called **MySoaAgent**.
- MyIISSoaAgent is configured to provide coarse-grain resource identification
- The resource is accessible over HTTP transport
- Web service URL is **services/soap2**.
- Web service name is **ExampleSearchService**.
- ExampleSearchService provides two operations:
  - **KeywordSearchRequest**
  - **PowerSearchRequest**

To protect ExampleSearchService, configure the following:

- A realm for ExampleSearchService with Resource Filter value "/services/soap2/ExampleSearchService"
- A single rule in the ExampleSearchService realm with Resource value "*"

Fine-Grain Resource Identification Over HTTP Example

Say you want to protect a resource with the following properties.

- The resource is hosted on an IBM WebSphere Application Server on host **soap** in domain **example.com** that is protected by a SiteMinder WSS Agent called **MyWSSoaAgent**.
- MyWSSoaAgent provides fine-grain resource identification
- The resource is accessible over HTTP transport
- Web service URL is **services/soap2**.
- Web service name is **ExampleSearchService**.
- ExampleSearchService provides two operations:
  - **KeywordSearchRequest**
  - **PowerSearchRequest**

To protect ExampleSearchService, configure the following:

- A realm for ExampleSearchService with Resource Filter value "/services/soap2/ExampleSearchService"
- One rule in the ExampleSearchService realm for each operation:
  - A rule for the KeywordSearchRequest operation with Resource value "/KeywordSearchRequest"
  - A rule for the PowerSearchRequest operation with Resource value "/PowerSearchRequest"

Fine-Grain Resource Identification Over JMS Example

Say you want to protect a resource with the following properties.

- The resource is hosted on BEA WebLogic Server on host **soap** in domain **example.com** that is protected by a SiteMinder WSS Agent called **MyWebLogicSoaAgent**.
- MyWebLogicSoaAgent provides fine-grain resource identification
- The resource is accessible over JMS transport
- JMS queue name is **ExampleQueue**
- Web service name is **ExampleSearchService**.
- ExampleSearchService provides two operations:
  - **KeywordSearchRequest**
  - **PowerSearchRequest**

To protect ExampleSearchService, configure the following:

- A realm for ExampleSearchService with Resource Filter value "/ExampleQueue"
- One rule in the ExampleSearchService realm for each operation:
  - A rule for the KeywordSearchRequest operation with Resource value "/KeywordSearchRequest"
  - A rule for the PowerSearchRequest operation with Resource value "/PowerSearchRequest"

Unprotected Realms, Rules, and Policies

By default a realm is created in a protected state. In most cases, you should use protected realms instead of changing a realm to an Unprotected state. In a protected realm, all resources are protected against access. To allow access, a rule must be defined, then included in a policy.

When you create a realm in an unprotected state, you must configure rules before CA SiteMinder® Web Services Security protects the resources in the realm. If you create a rule for resources in the unprotected realm, only the specified resources are protected. Once the resource is protected, the rule must be added to a policy to allow users to access the resource. You may want to use an unprotected realm if only a subset of the resources in a realm need to be protected from unauthorized access.

The following is an example of the actions required when setting up an Unprotected realm:

Action	Protection State
Create unprotected realm called Realm1 with the Resource Filter: /dir.	Resources contained in /dir and subdirectories are not protected.
Create Rule1 in Realm1 for the resource: getCachedQuote.asp.	The /dir/getCachedQuote.asp resource is protected, but the rest of the contents of /dir are not protected.
Create Policy1 and bind Rule1 and User1 to the Policy.	User1 can access /dir/getCachedQuote.asp. All other users cannot access the protected file.

Guided Example: Create Security Policies from a WSDL File

Deployed web services are typically described in an associated Web Services Description Language (WSDL) file. One way of getting started creating security policies using tradtional policy management, especially in terms of creating realms, rules, and the resource mappings they define, is to work from the WSDL file associated with a deployed web service.

Follow these steps:

1. Parse the WSDL file for the web service you want to secure. Look for <service> elements. A <service> element contains the web service <port> elements which need to be secured. The name attribute of a <port> element identities the port type (and hence contains a reference to a <portType> element). A <port> element also contains the binding URL which refers to the URL where the web service is located. The web service port is protected by creating a realm whose Resource Filter is the combination of the binding URL and port name.

In the following snippet from ExampleSearch.wsdl, the web service port to secure is ExampleSearchPort. This port is bound to the URL http://api.example.com/*search/beta2.*

```
<service name="ExampleSearchService">
   <port name="ExampleSearchPort" binding="typens:ExampleSearchBinding">
     <soap:address location="http://api.example.com/search/beta2"/>
   </port>
 </service>
```

2. To protect the ExampleSearchPort web service, create a realm named ExampleSearchRealm whose Resource Filter is /search/beta2/. Choose a SiteMinder WSS Agent and authentication scheme with which to secure this realm as appropriate.
3. Repeat Step 2 (create a realm) for every <port> element contained within the <service> element in the WSDL file.
4. Look for all the web service operations that are available under the above web service port by looking for the <portType> element whose name matches the name of the <port> element.

   In the following snippet, the three web service operations to secure are doGetCachedPage, doSpellingSuggestion and doExampleSearch. All these three operations are children of ExampleSearchPort which has been secured by the realm named ExampleSearchRealm.

```
<portType name="ExampleSearchPort">
  <operation name="doGetCachedPage">
    <input message="typens:doGetCachedPage"/>
    <output message="typens:doGetCachedPageResponse"/>
  </operation>

  <operation name="doSpellingSuggestion">
    <input message="typens:doSpellingSuggestion"/>
    <output message="typens:doSpellingSuggestionResponse"/>
  </operation>

  <operation name="doExampleSearch">
    <input message="typens:doExampleSearch"/>
    <output message="typens:doExampleSearchResponse"/>
  </operation>

</portType>
```

5. To configure fine-grain authorization policies, you must secure every child <operation> element. Create a rule under the ExampleSearchRealm realm for each operation with the following properties:

   Resource Filter: "/*Web Service Operation Name*"

   Action: Post, ProcessSOAP and ProcessXML Web Agent actions

6. Create a policy containing the rules you created for every web service operation in the WSDL; assign users to the policy, as required.

Chapter 6: (Optional) Configure Variables To Use in Message-based Authorization Policies

This section contains the following topics:

eTelligent Rules

Use eTelligent Rules to define variables that enable fine-grained access-control criteria that are known as policy expressions.

Policy expressions are implemented as policy attributes. They include operators and customer-defined variables that are evaluated at runtime, when a user attempts to access a protected resource on a web site.

Variables can store local information that is within the enterprise or remote information that a web service providess.

The variables that eTelligent Rules provides are available in the Administrative UI. You can define variable objects and can incorporate them into policy logic through policy expressions. You can also include variables in CA SiteMinder® response objects.

eTelligent Rules Benefits

eTelligent rules provides the following benefits:

- Reduce complexity and eliminate the need for custom code.

  The CA SiteMinder® administrator defines authorization access in policy expressions, using graphical tools rather than application code. It is not necessary to integrate and reconcile backend business application access control information, because that information is centralized in the CA SiteMinder® Policy Server.

- Use business data dynamically in security policies.

  Defining access control to secure resources is based on local user information and incoming information, such as the amount of a purchase order that a user places an order.

- Combine various types of information for authorization decisions.

  Web browser forms data, user-context data (stored locally in the Policy Server), and remote data (obtained through a service bureau) can be flexibly combined in policy expressions.

- Make transactional decisions online.

  It is not necessary to go back to a backend business application each time authorization is required to access a protected resource.

- Rely on XML-based third-party security data.

  eTelligent Rules use a standard XML protocol to communicate with trusted service bureaus, thus increasing the choice of web services providers.

- Use Boolean logic.

  CA SiteMinder® security administrators define policy expressions using variables together with logical operators.

- Minimize the number of policies required.

  Due to the use of policy expressions that are based on logic, fewer policies are necessary, thus keeping policy administration to a minimum.

How to configure eTelligent Rules

To configure eTelligent Rules, do the following tasks:

1. Configure the following eTelligent Rules properties files:
   - JVMOptions.txt
   - LoggerConfig.properties
2. Configure variables.
3. Configure policy expressions that use the eTelligent Rules variables.

More information:

Variables Overview (see page 109)
eTelligent Rules Properties Files (see page 107)

eTelligent Rules Properties Files

The following properties files are required for eTelligent Rules:

- JVMOptions.txt

  This file is required to configure the JVM for eTelligent Rules. The installed location of this file is: *policy_server_home*/config/

- LoggerConfig.properties

  This file is required to configure logging for eTelligent Rules. The installed location of this file is:

  policy_server_home/config/properties

More information:

JVMOptions.txt File (see page 107)
Modify the LoggerConfig.properties File (see page 108)

JVMOptions.txt File

The JVMOptions.txt file contains the settings that the Policy Server uses when creating the Java Virtual Machine that is used to support eTelligent Rules.

If you encounter errors that are related to missing classes, you can modify the classpath directive in the JVMOptions.txt file. For complete information about the settings that are contained in the JVMOptions.txt file, see your Java documentation.

Modify the LoggerConfig.properties File

On the Policy Server, the LoggerConfig.properties file allows you to specify logging features that are used when you start the SiteMinder service from a command line. The properties that are contained in this file are not used when the service is started from the Policy Server Management Console. Modify this file to obtain more output for debugging purposes.

The following shows an example of a LoggerConfig.properties file.

```
// LoggingOn can be Y, N
LoggingOn=Y

// LogLevel can be one of LOG_LEVEL_NONE, LOG_LEVEL_ERROR,
LOG_LEVEL_INFO, LOG_LEVEL_TRACE
LogLevel=LOG_LEVEL_TRACE

// If LogFileName is set Log output will go to the file named
LogFileName=affwebserv.log

// AppendLog can be Y, N.  Y means append output to LogFileName if
specified
AppendLog=Y

// AlwaysWriteToSystemStreams can be Y, N.
// Y means log messages are written to System.out
// or System.err regardless of what the logger streams are
// set to.  If the logger streams are set to System.out
// or System.err log messages will be written multiple times.
// This facilitates logging messages to System.out/System.err
// and a file simultaneously.
AlwaysWriteToSystemStreams=N

// DateFormatPattern can be any valid input to java.text.DateFormat
constructor.
// See the Java documentation for java.text.DateFormat for details
// If not specified, the default format for the default locale is used
DateFormatPattern=MMMM d, yyyy h:mm:ss.S a
```

The settings in this file are:

LoggingOn

Enables or disables logging. Set this parameter to Y to enable logging. Set this parameter to N to disable logging.

LogLevel

Indicates the level of detail that is contained in logs. The LogLevel can be one of the following values:

LOG_LEVEL_NONE

No messages are logged.

LOG_LEVEL_ERROR

Only records error messages.

LOG_LEVEL_INFO

Records error messages and warnings.

LOG_LEVEL_TRACE

Records error messages, warnings, and general processing information that is useful for tracking problems.

LogFileName

If LogFileName is set, all log output goes to the file named in this parameter.

AppendLog

Indicates whether logging information is appended to an existing file at startup or if a new file is created at startup. To append output to the file specified in the LogFileName parameter, set this parameter to Y. To create a new file at startup, set this parameter to N.

AlwaysWriteToSystemStreams

To log messages to System.out or System.err regardless of what the logger streams are set to, set this parameter to Y. If the logger streams are set to System.out or System.err, log messages are written multiple times. This facilitates logging messages to System.out/System.err and a file simultaneously.

DateFormatPattern

DateFormatPattern can be any valid input to java.text.DateFormat constructor. See the Java documentation for java.text.DateFormat for details.

If not specified, the default format for the default locale is used.

Variables Overview

The Policy Server resolves variable objects to a value which you can incorporate into the authorization phase of a request. The value of a variable object is the result of dynamic data and is evaluated at runtime. Variables provide a flexible tool for expanding the capabilities of policies and responses.

Variable Types

The following types of variables are available:

- Static Variables (see page 110)
- Request Context Variables (see page 110)
- User Context Variables (see page 111)
- Form Post Variables (see page 111)
- Web Services Variables

Static Variables

Static variables consist of a simple name/value pair of a particular type, such as string, boolean, and others. The key benefit of a static variable is to implement good programming practices. Instead of repeating the value of a constant each time it's used in a policy, a static variable provides a single piece of data that can be used throughout multiple policies.

Request Context Variables

Each request that is processed establishes a request context. This context identifies the following properties:

Action

Indicates the type of action that is specified in the request, such as GET or POST.

Resource

Indicates the requested resource, such as /directory_name/.

Server

Indicates the full server name that is specified in the request, such as server.example.com.

A request context variable can capture any of this information and can make it available for inclusion in a policy expression or response. The key benefit of this type of variable is to provide fine-grained request context information without any programming logic.

User Context Variables

When the Policy Server authenticates a user against an entry in a directory, a user context is created. The user context consists of information about the user directory and the contents of the directory that pertain to the authenticated user.

User context variables can be based on an attribute of a directory connection, or based on the contents of the directory. The key benefit of this type of variable is to provide flexibility in defining rules that are based on particular user context without any programming logic.

Form Post Variables

HTML forms are often used to collect information that back-end applications require. Form Post variables can be used to capture any information that is entered in an HTML form and POSTed. For example, if the business logic associated with an application requires a purchase order amount that is specified on an HTML login form, create a Form Post variable object to collect the value of the purchase order that the user supplies. The variable can then be used in policies.

Important: EJB and Servlet Agents do not support Form Post variables. Do not use Form Post variables in policies that are enforced by EJB or Servlet Agents.

The key benefit of this type of variable is that it allows the Policy Server to use POST data as a part of a policy expression rather than forcing enterprises to build security logic into HTML forms are often used to collect information that back-end applications require. Form Post variables can be used to capture any information that is entered in an HTML form and POSTed. For example, if the business logic associated with an application requires a purchase order amount that is specified on an HTML login form, create a Form Post variable object to collect the value of the purchase order that the user supplies. The variable can then be used in policies. server applications. Using HTTP POST variables results in efficient network usage between Agents and Policy Servers. The Agent extracts the HTTP variable information from the HTTP stream so that the information can be used during authorization processing by the Policy Server.

Variable Use in Policies

Variables allow you to include business logic in policies by capturing a wide range of dynamic data that can be built into policy expressions. When you define variable objects in the Administrative UI, you can use those variables in expressions in the Policy dialog on the Expression tab. You can build expressions that use multiple variable objects and boolean operators to capture complex business logic in your policies.

For example, a policy can contain an expression that requires the value of a user account type and a credit score to allow access to an application. An expression can be defined in the policy so that only users whose account type is “gold”, and whose credit score is greater than a specific value can have access to a resource. This example requires two variables, which must be combined in an expression.

Message-based Authorization Using Variables

Variables are objects that can be resolved to a value, which you can incorporate into the authorization phase of a request. The value of a variable object is the result of dynamic data and is evaluated at run time.

To make authorization decisions based on the transport header, SOAP envelope header, XML payload, or SAML assertions, you can define specific CA SiteMinder® Web Services Security variables and add them to policies in the form of policy expressions. The Policy Server can use a policy expression as an additional criterion when determining if a client should be permitted access to a web service.

Note: Variables can only be used in policy expressions when using traditional (policy domain-based) policy management. They are not available when using enterprise (application-based) policy management.

CA SiteMinder® Web Services Security provides five variables types that represent dynamic, context-sensitive data from any layer (transport, message envelope, or message body) of an XML message. All of these variables can be used in policy expressions.

- **SAML Assertion**—Lets you obtain information from SAML assertions.
- **Transport**—Lets you to obtain HTTP header values from the web service request.
- **XML Agent**—Lets you obtain information about the web server whose resources the SiteMinder WSS Agent is protecting.
- **XML Body**—Lets you obtain information from any element in the body (or payload) of an incoming XML message.
- **XML Envelope Header**—Lets you obtain information from any element in the SOAP envelope (including WS-Security headers) of an incoming XML message.

Once defined, these variables can be used in policy expressions to make authorization decisions. For example, you could define an XML body variable called ShipToZipCode that corresponds to an XML query that obtains the ship-to ZIP code from a purchase order XML document. Variables can also be used in responses.

Variable Use in Responses

Variables can be used in responses. When you define variable objects in the Administrative UI, you can use those variables in responses. The value of the response is created at runtime by the Policy Server as it resolves the value of a variable object.

Create a Variable

You create a variable to make it available for use in policies or responses. Variables are domain objects. You create them within a specific policy domain, or import them into a domain using the smobjimport tool.

More information about importing objects into policy domains exists in the *Policy Server Administration* guide.

Create a SAML Assertion Variable

SAML Assertion variables let you obtain information from any SAML assertion and use this information in policy expressions to authorize a client. The assertion may be included in a SOAP envelope or HTTP header of an incoming XML message. For example, you can create a variable that enables the Policy Server to check who issued the assertion before permitting access to a web service.

SAML assertion variables are resolved to the value of an XPath string. The string identifies an element (and optionally, an operation to perform on that element) of a SAML assertion.

Note: For more information about XPATH, see the XPATH specification available at http://www.w3.org/TR/xpath.

Follow these steps:

1. Click Policies, Domain.
2. Click Variables.
3. Click Create Variable.

   Verify that the Create a new object of type Variable option is selected.
4. Click OK.
5. Select a domain from the list and click Next.
6. Type the variable name in the Name field.
7. Select SAML Assertion from the Variable Type list.

   SAML Assertion variable settings open.

8. Specify the data type in which the value of the specified XPATH query should be returned by choosing one of the following options from the Return Type list:
   - Boolean
   - Number
   - String (the default)
9. Type in an XPath query that you want to resolve to the variable value in the Query box.
10. Optionally, set the SAML Authentication Scheme Required box if the web service is protected by the SAML Session Ticket authentication scheme.
11. If the web service is not protected by the SAML Session Ticket authentication scheme, specify whether the SiteMinder WSS Agent should look for the SAML assertion in the Envelope Header or HTTP Header by selecting the appropriate SAML Assertion Location option.
12. Click Finish.

    The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create a Transport Variable

Transport variables let you obtain HTTP header values from the web service request.

Follow these steps:

1. Click Policies, Domain.
2. Click Variables.
3. Click Create Variable.

   Verify that the Create a new object of type Variable option is selected.
4. Click OK.
5. Select a domain from the list and click Next.

6. Type the variable name in the Name field.
7. Select Transport from the Variable Type list.

   Transport variable settings open.
8. Enter information in the following fields:

   Description

   (Optional) Specifies a brief description of the variable.

   Limits: No more than 1KB.

   Return Type

   Specifies the data type in which the value of the transport header data should be returned:

   - Boolean
   - Date
   - Number
   - String (the default)

   Transport Data Name

   Specifies the name of the HTTP header (for example, SOAPAction) that will provide the value of the variable.
9. Click Finish.

   The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create an XML Agent Variable

XML Agent variables let you obtain information about the web server whose resources the XML Agent is protecting for use in policy expressions or responses.

Follow these steps:

1. Click Policies, Domain.
2. Click Variables.
3. Click Create Variable.

   Verify that the Create a new object of type Variable option is selected.
4. Click OK.
5. Select a domain from the list and click Next.

6. Type the variable name in the Name field.
7. Select XML Agent from the Variable Type list

   XML Agent variable settings open.
8. Enter information in the following fields:

   Description

   (Optional) Specifies a brief description of the variable.

   Limits: No more than 1KB.

   Property

   Specifies the XML Agent property that will provide the value of the variable:

   - Server Product Name—String representation of the web server product name—for example, iPlanet Web Server. Value is obtained from the ServerProductName Agent Configuration parameter.
   - Server Vendor—String representation of the web server vendor—for example, Sun. Value is obtained from the ServerVendor Agent Configuration parameter.
   - Server Version—String representation of the web server product version—for example, 6.0 SP2.
9. Click Finish.

   The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create an XML Body Variable

XML Body variables let you obtain information from any element in the body (or payload) of an incoming XML message for use in policy expressions and responses.

Specifically, XML Body variables are resolved to the value of an XPath string that identifies an element (and optionally, an operation to perform on that element) of an XML document.

Note: For more information about XPATH, see the XPATH specification available at http://www.w3.org/TR/xpath.

Follow these steps:

1. Click Policies, Domain.
2. Click Variables.
3. Click Create Variable.

   Verify that the Create a new object of type Variable option is selected.

4. Click OK.
5. Select a domain from the list and click Next.
6. Type the variable name in the Name field.
7. Select XML Body from the Variable Type list.
8. XML Body variable settings open.
9. Enter information in the following fields:

   Description

   (Optional) Specifies a brief description of the variable.

   Limits: No more than 1KB.

   Return Type

   Specifies the data type in which the value of the specified XPATH query should be returned:

   - Boolean
   - Date
   - Number
   - String (the default)

10. Do one of the following:

    - Load a schema (.xsd) file and select the element to map to the specified field name by browsing using the following procedure:

      a. Click Browse and navigate to the schema file.

      b. Click Upload XSD File.

         The schema is uploaded.

      c. Select the schema element that you want to map to the specified field name in the Select a node group box.

         The Select a node group box displays the selected schema using a standard tree-style hierarchical view. Click the plus sign (+) next to an element to expand it. Click the minus sign (-) beside an expanded element to contract it. Elements marked with an asterisk (*) are repeatable within the XML document (that is, incoming XML documents may contain multiple instances of that element).

    - Unset the Advance XPath query option and type an XPATH query defining the mapping in the XPath field.

11. Optionally, if you are working from a loaded schema in the Select a node group box, specify an XPath function (count, div, index, mod, sum) that you want to apply to a repeatable schema element, by choosing it from the Function drop-down list.

    The Function option lets you create more complex mappings by processing functions that further evaluate the XML document.

    Note: For more information about these functions, go to the XPATH specification at http://www.w3.org/TR/xpath.

12. Click Finish.

    The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create an XML Envelope Header Variable

XML Envelope Header Variables let you obtain information from any element in the SOAP envelope header (including WS-Security headers) of an incoming XML message, for use in policy expressions or responses.

Specifically, XML Envelope Header variables are resolved to the value of an XPath string that identifies a SOAP envelope header element (and optionally, an operation to perform on that element) of an XML document.

Note: For more information about XPATH, see the XPATH specification available at http://www.w3.org/TR/xpath.

Follow these steps:

1. Click Policies, Domain.
2. Click Variables.
3. Click Create Variable.

   Verify that the Create a new object of type Variable option is selected.
4. Click OK.
5. Select a domain from the list and click Next.
6. Type the variable name in the Name field.
7. Select XML Header from the Variable Type list.

   XML Header variable settings open.

8. Specify the data type in which the value of the specified XPATH query should be returned by choosing one of the following options from the Return Type list:
   - Boolean
   - Number
   - String (the default)
9. Type in an XPath query that you want to resolve to the variable value in the Query box.
10. Click Finish.

    The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create a Static Variable

You create a static variable to make it available for use in policies or responses.

Note: The value of the resolved variable must not be greater than 1 K.

To create a variable

1. Open the domain to which to you want to add a variable.
2. Click the Variables tab.

   A table lists the variables associated with the domain.
3. Click Create Variable.

   The Create Variable screen appears.
4. Verify that Create a new object is selected, and click OK.

   Variable settings open.
5. Type the variable name in the Name field.
6. Select Static from the Variable Type list.

   Static variable settings open.

   Note: Click Help for descriptions of settings and controls, including their respective requirements and limits.
7. Specify the data type and value of the variable in the Variable Information group box.
8. Click Submit.

   The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create a Request Context Variable

You create a request context variable to make it available for use in policies or responses.

Note: The value of the resolved variable must not be greater than 1 K.

To create a variable

1. Open the domain to which to you want to add a variable.
2. Click the Variables tab.

   A table lists the variables associated with the domain.
3. Click Create Variable.

   The Create Variable screen appears.
4. Verify that Create a new object is selected, and click OK.

   Variable settings open.
5. Type the variable name in the Name field.

   Note: Request Context variable names must begin with the percent character (%).

   Example: %REQUEST_ACTION
6. Select Request Context from the Variable Type list.

   Request context settings open.
7. Select the variable value from the Property list.
8. Click OK.

   The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create a User Context Variable

You create a user context variable to make it available for use in policies or responses.

Note: The value of the resolved variable must not be greater than 1 K.

To create a variable

1. Open the domain to which to you want to add a variable.
2. Click the Variables tab.

   A table lists the variables associated with the domain.
3. Click Create Variable.

   The Create Variable screen appears.

4. Verify that Create a new object is selected, and click OK.

   Variable settings open.

5. Type the variable name in the Name field.

   Note: User Context variable names must begin with the percent character (%).

   Example: %SM_USERPATH

6. Select User Context from the Variable Type list.

   User context settings open.

7. Select the portion of the user context that provides the value of the variable from the Property list.

   The return type value appears as either string or boolean depending on the value you selected from the Property list.

8. (Required for User Property and Directory Entry) Enter the name of the directory or user attribute that provides the variable value in the Property field.

9. (Required for User Property and Directory Entry) Enter the size of the buffer (in bytes) that is to store the variable in the Buffer field.

10. (Required for Directory Entry) Enter the distinguished name of the directory entry in the DN field.

11. Click Submit.

12. The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions or responses.

Create a Form Post Variable

You create a Form Post variable to make it available for use in policies.

Note: The value of the resolved variable must not be greater than 1 K.

To create a variable

1. Open the domain to which to you want to add a variable.

2. Click the Variables tab.

   A table lists the variables associated with the domain.

3. Click Create Variable.

   The Create Variable screen appears.

4. Verify that Create a new object is selected, and click OK.

   Variable settings open.

5. Type the variable name in the Name field.

6. Select Post from the Variable Type list.

   Form post settings open.

7. Enter the name of the POST variable contained in the form in the Form Field Name field.

8. Click OK.

   The variable appears in the Variables tab of the domain. The variable can now be used in policy expressions.

Configure Message-based Authorization Using an XPath Query in XmlToolkit.properties

You can configure message content-based authorization based on information in incoming XML messages by configuring variables and policy expressions to extract the required information and trigger authorization decisions based on the obtained values.

Alternatively, you can configure an XPath query in the target SiteMinder WSS Agent's XMLToolkit.properties file that extracts a value from each incoming message and incorporates it into your policy's resource value.

Note: To find out the location of the XMLToolkit.properties file for each SiteMinder WSS Agent type, see the respective SiteMinder WSS Agent Guide.

Follow these steps:

1. Open the XmlToolkit.properties file in a text editor.
2. Make the following changes in XmlToolkit.properties:
   - Add an ResourceXpathQuery parameter whose value is a valid XPath query that identifies the required XML message element, for example:

     ```
     ResourceXPathQuery=/SOAP-ENV:Envelope/SOAP-ENV:Header/method
     ```

   - (Optional) Add a NodeProperty parameter which specifies a property of an element or attribute to be returned from the XPath query and passed by the SOAP envelope handler to the XPath evaluate method.

3. Save and close the XmlToolkit.properties file.
4. Restart the target SiteMinder WSS Agent.

Notes:

- XPath query processing is namespace aware.
- The XPath query must be rooted at the document root—not at the header or body.
- You can configure only one XPath query per agent instance.
- If the XPath query fails, the target URL will be used as the resource.
- The XPath query is loaded at SiteMinder WSS Agent startup; if it is changed, you must restart the agent.

Chapter 7: Variables

Index

No index entries found.