BEA WebLogic Enterprise Security™®

Programming Security for Web Services

Product Version: 4.2
Revised: August 12, 2005

Copyright

Restricted Rights Legend

Trademarks or Service Marks

1. Introduction to the Web Services Security Service Module

2. Web Services Interfaces

CHAPTER 1

Introduction to the Web Services Security Service Module

This document provides an introduction to WebLogic Enterprise Security Web Services product and describes interfaces clients use to interact with it.

This section covers the following topics:

About This Document

This section covers the following topics:

Audience for This Guide

It is assumed that the reader understands Web Services technologies and has a general understanding of the Microsoft Windows or UNIX operating systems being used. This document is intended for the following audiences:

- Web Services Developers—Developers who are programmers who focus on developing Web services applications and who work with other engineering, quality assurance (QA), and database teams to implement security features. Web services developers have in-depth working knowledge of Web Services programming.
- Application Developers—Developers who focus on designing applications and work with other engineers, quality assurance (QA) technicians, and database teams to implement applications.
- Security Administrators—Administrators who are responsible for installing and configuring the WebLogic Enterprise Security products and designing policy. Security administrators work with other administrators to implement and maintain security configurations, authentication and authorization schemes, and to set up and maintain access to deployed application resources. Security administrators have a general knowledge of security concepts and the WebLogic Enterprise Security architecture.

How this Document is Organized

This document is organized as follows:

- Chapter 1 (this chapter) provides an overview of the Web Services Security Service Module and the related standards and features it supports.
- Chapter 2, "Web Services Interfaces," describes the Web Services interfaces and the methods that each interface supports.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev web site:

`http://dev2dev.bea.com`

To view the documentation for this product, select More Product Centers from the menu on the left side of the screen on the dev2dev page. From the BEA Products list, choose WebLogic Enterprise Security. The home page for this product is displayed. From the Resources menu,

choose Documentation 4.2. The home page for the complete documentation set for the product is displayed.

Related Information

The BEA corporate web site provides all documentation for BEA WebLogic Enterprise Security. Other BEA WebLogic Enterprise Security documents that may be of interest to the reader include:

- *Programming Security for Java Applications*—The document describes how to implement security in Java applications. It include descriptions of the Security Service Application Programming Interfaces and programming instructions for implementing security in Java applications.
- *BEA WebLogic Enterprise Security Administration Guide*—This document provides a complete overview of the product and includes step-by-step instructions on how to perform various administrative tasks.
- *Developing Security Providers for BEA WebLogic Enterprise Security* —This document provides security vendors and security and application developers with the information needed to develop custom security providers.
- *BEA WebLogic Enterprise Security Policy Managers Guide*—This document defines the policy model used by BEA WebLogic Enterprise Security, and describes how to import and export policy data.
- *WSDL Documentation for the Web Service Interfaces*—This document provides reference documentation for the Web Services Interfaces that are provided with and supported by this release of BEA WebLogic Enterprise Security.
- *BEA WebLogic Enterprise Security Web Server Installation*—This document describes how to install Web Server Security Service Module.
- *Javadocs for Security Service Provider Interfaces*—This document provides reference documentation for the Security Service Provider Interfaces that are provided with and supported by this release of BEA WebLogic Enterprise Security.

Overview of Web Services

Web services are a special type of service that can be shared by and used as components of distributed Web-based applications. Web services interface with existing back-end applications, such as customer relationship management systems, order-processing systems, and so on.

Traditionally, software application architecture tended to fall into two categories: huge monolithic systems running on mainframes or client-server applications running on desktops. Although these architectures work well for the purpose the applications were built to address, they are closed and cannot be easily accessed by the diverse users of the Web. Thus the software industry has evolved toward loosely coupled service-oriented applications that interact dynamically over the Web. The applications break down the larger software system into smaller modular components, or shared services. These services can reside on different computers and can be implemented by vastly different technologies, but they are packaged and transported using standard Web protocols, such as Extensible Markup Language (XML) and Hyper Text Transfer Protocol (HTTP), thus making them easily accessible by any user on the Web.

This concept of services is not new. Remote Method Invocation (RMI), Component Object Model (COM), and Common Object Request Broker Architecture (CORBA) are all service-oriented technologies. However, applications based on these technologies must be written using that particular technology, often as implemented by a particular vendor. This requirement typically hinders widespread acceptance of such services on the Web. To solve this problem, web services are defined to share the following properties that make them easily accessible from heterogeneous environments:

- Web services are accessed over the Web.
- Web services describe themselves using an XML-based description language.
- Web services communicate with clients (both end-user applications or other web services) through XML messages that are transmitted by standard Internet protocols, such as HTTP.

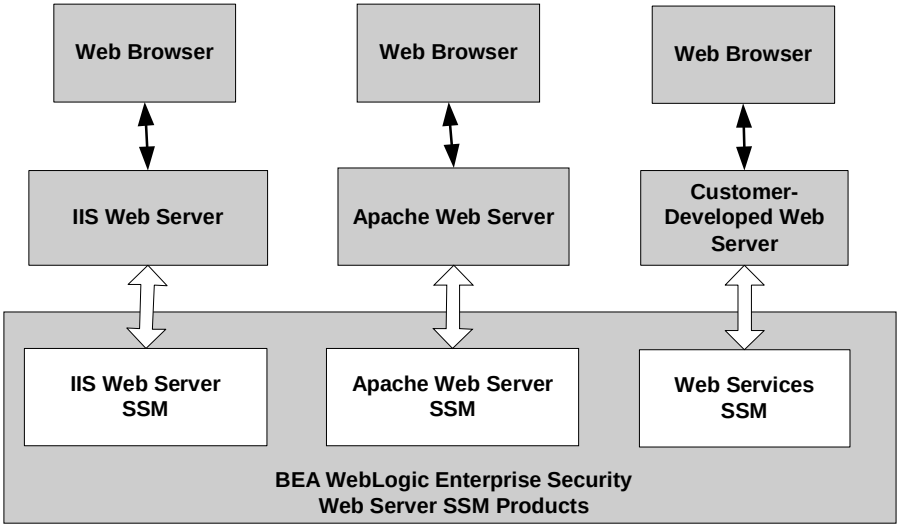
Product Overview

The BEA WebLogic Enterprise Security provides three Web Server products: the IIS Web Server Security Service Module (SSM), the Apache Web Server SSM, and the Web Services SSM (see

Figure 1-1). This document only describes the Web Services SSM and the security service application programming interfaces (APIs) that it supports.

For a description of the IIS and Apache Web Server SSMs, see *Web Server Installation.*

Figure 1-1 Web Server Product Components



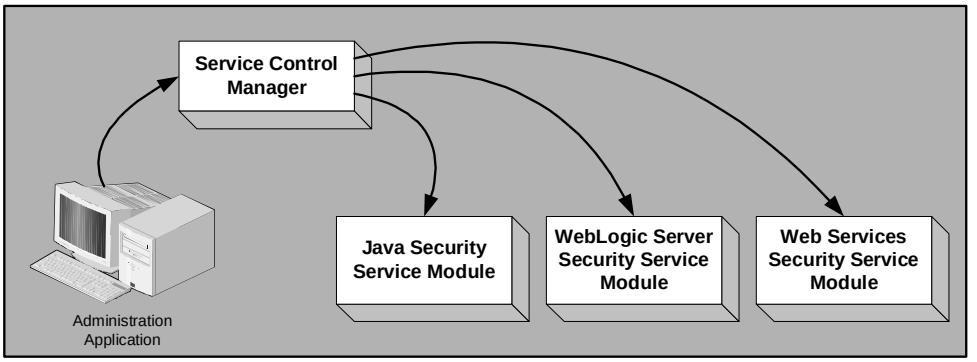
The following topics provide more information on the these products:

- "Web Server Product Environment" on page 1-5
- "Web Services Security Service Module" on page 1-7
- "Client Trust Model" on page 1-8
- "Deployment Model" on page 1-9
- "Usage Model" on page 1-10

Web Server Product Environment

Figure 1-2 shows the major components that make up the BEA WebLogic Enterprise Security product environment.

Figure 1-2 WebLogic Enterprise Security Product Environment



- Administration Application

  The Administration Application allows you to configure, deploy, and manage multiple Security Service Modules in a distributed environment. While the modules consume configuration data and then service security requests accordingly, the Administration Application allows you to configure and deploy security configuration information to the modules and modify that information as needed.

- Service Control Manager

  The Service Control Manager (SCM) is an essential component of the configuration provisioning mechanism and a key component of a fully-distributed security enforcement architecture. A SCM is a machine agent that exposes a provisioning, or deployment, interface to the Administration Application to facilitate the management of a potentially large number of distributed Security Service Modules (SSMs). The SCM can receive and store meta-data updates, both full and incremental, initiated by the Administration Application.

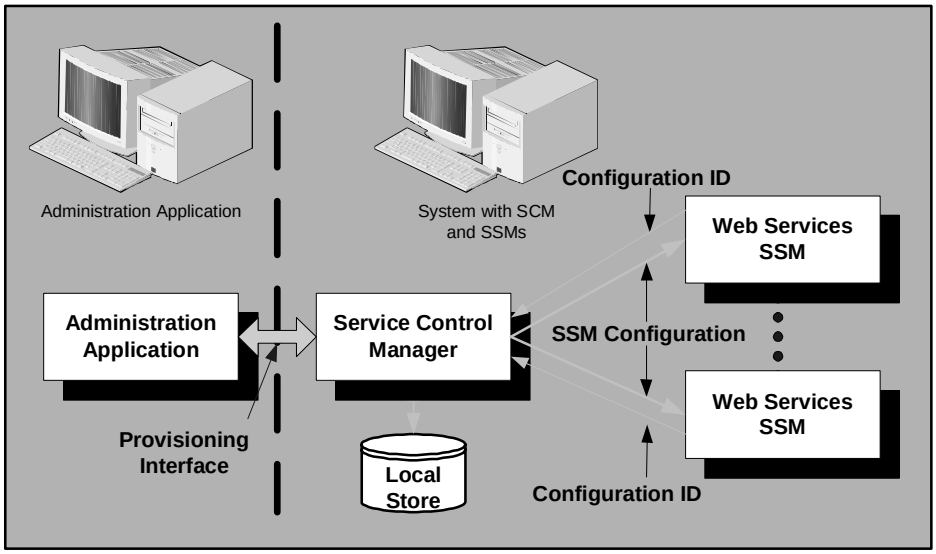
  The Administration Application uses this provisioning mechanism to distribute configuration data to each created instance of a SSM where it is consumed locally (see Figure 1-3). Each instance of a SSM is assigned a unique configuration ID, which is registered with the SCM when the SSM is enrolled. The SCM uses the configuration ID when distributing and updating configuration data to each SSM instance to ensure that the correct data is distributed.

- Web Services Security Service Module

  The Web Services Security Service Module (SSM) is used to adapt web servers to the WebLogic Enterprise Security infrastructure so that web server resources can be protected

by a custom security configuration. You define and deploy the security configuration using the Administration Application. You can only configure one instance of the Web Services SSM on a single machine, however, the number of machines in your network on which you can configure Web Services SSMs is unlimited (see Figure 1-3). After you deploy the initial security configuration to a Web Server SSM, it does not require any additional communication with the Service Control Manager (SCM) to perform runtime security functions. However, the SCM does maintain communication with each Web Services SSM instance so that it can distribute, or deploy, full and incremental security configuration updates.

Figure 1-3 Deploying Security Configuration Data



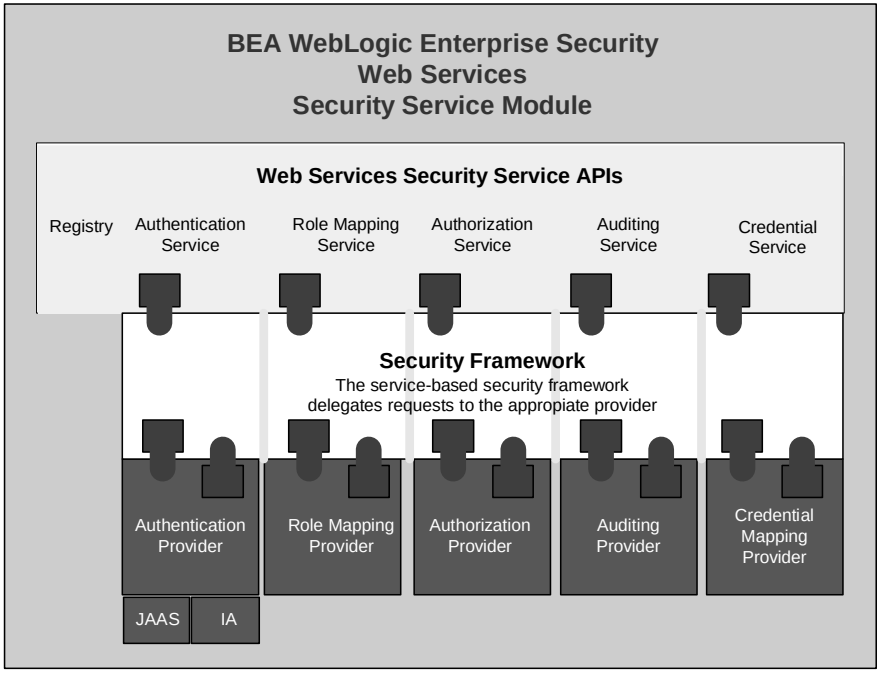
Web Services Security Service Module

The Web Services Security Service Module (SSM) provides six security service APIs: Registry, Authentication, Authorization, Auditing, Role Mapping, and Credential Mapping (see Figure 1-4). These APIs can be used to developed web services clients to access the WebLogic Enterprise Security infrastructure and use it to make access control decisions for users attempting to access web server application resources. Once the web services client is implemented, it uses the Web Services SSM (which incorporates the Security Services APIs, the Security Framework, and the configured security providers) to make access control decisions for the web server to which it is connected. Then you can use the WebLogic Enterprise Security Administration

Application to configured and deploy a security configuration to protect the web server application resources. Thus, the Web Services SSM enables security administrators and web developers to perform security tasks for applications running on a web server. Additionally, you can use the Web Services SSM to add information provided by the Security Framework (such as roles and response attributes) to the HTTP requests handled by the protected web server applications.

Figure 1-4 shows the components of the Web Services SSM.

Figure 1-4 Web Services SSM Components



For a description of the Web Services Security Service APIs, see "Web Services Interfaces" on page 2-1

Client Trust Model

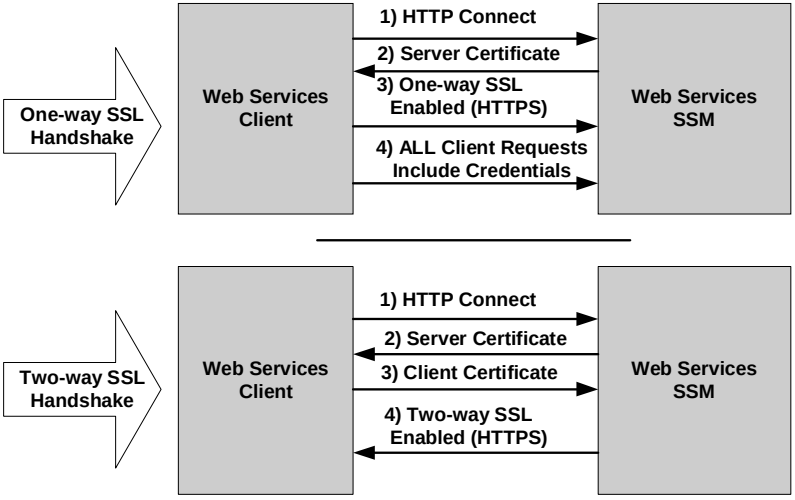
To protect messages in transit between the client and the Web Services SSM, a channel security protocol (SSL 3.0 or TLS 1.0) is used for all communication between the two. The Web Services SSM supports both one-way and two-way SSL (see Figure 1-5). To establish an SSL connection

with one-way SSL, only the server is required to present a digital certificate. With two-way SSL authentication, both the client and server must present digital certificates before the SSL connection is enabled between the two. Thus, with two-way SSL, the Web Services SSM not only authenticates itself to the client to complete the SSL handshake (which is the minimum requirement for certificate authentication), but it also requires authentication from the requesting client. The client first authenticates the server's certificate, and then the server authenticates the client's certificate. The client certificate must be signed by a recognized certificate authority (CA).

As mentioned, Web services clients can communicate with the Web Services SSM over one-way or two-way SSL connections. However, when the client communicates over a one-way SSL, each client request be accompanied by a valid client name/password credentials pair, a signed SAML assertion, or a WLES cookie. Figure 1-5 shows the Web Services SSM client trust model.

For more information on authentication, see "Authentication Service Interface" on page 2-4.

Figure 1-5 Client Trust Model

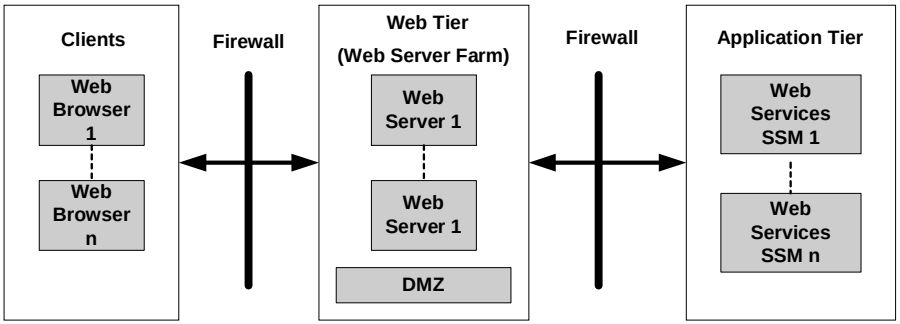


Deployment Model

Figure 1-6 shows how you typically deploy instances of Web Services Security Service Modules (SSM) to protect application resources. The web servers (the web services clients) are located in the web tier, which is in the Demilitarized Zone (DMZ), and are protected from unwanted traffic on the Internet by a firewall. The DMZ is created by using two firewalls. The Web Services SSMs

are located in the application tier behind the second firewall for added security. The Web Services SSM supports Secure Sockets Layer (SSL) communication so all traffic between the web servers and the SSM is encrypted.

Figure 1-6 Typical Web Services Deployment Model

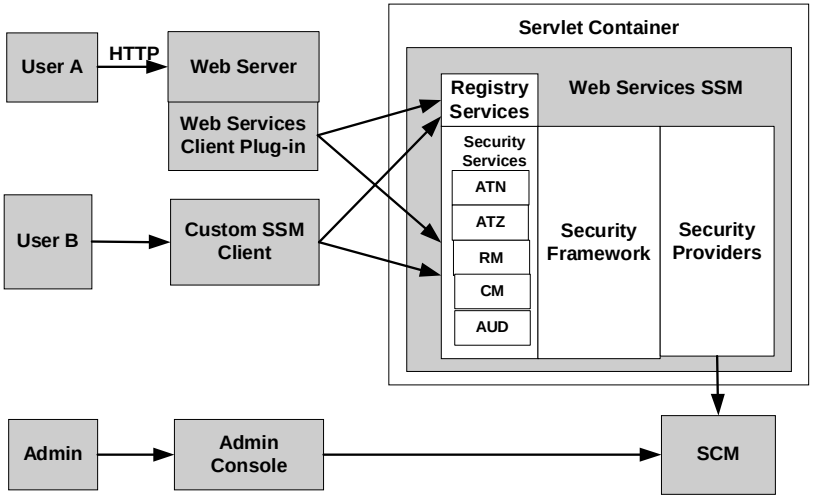


Usage Model

Once the Web Services client has established an SSL connection with the Web Services SSM, it can issue requests on behalf of users for access to the web resources protected by the Web Services SSM (see Figure 1-7).

- User A uses web servers with web services client plug-ins to access the Registry Service and locate specific instances of SSMs and a specific security service on that SSM instance. The Web servers use that information to make requests access to application resources on behalf of users. BEA provides two web server products that have built-in web services clients, the IIS Web Server SSM and the Apache Web Server SSM.
- User B uses a custom SSM client, which has a user-written web services client, to access the Registry Service and in turn the specific SSM security service. Because communication between the custom SSM client and the Web Services SSMs is placed on the network, the Web Services SSMs can be located any where on the network. Their location is not restricted to the SSMs on the local machine.
- The Admin user uses the administration console to the distribute the initial security configuration and all updates over the network. If the Administration Server is disconnected from the Web Services SSMs, the SSMs continue to function.

Figure 1-7 Usage Model



The interaction between the client and the Web Services SSM is as follows (see Figure 1-7):

1. The client communicates with the Web Services SSM and establishes an SSL connection.
1. The client issues a user request that includes the following:
   - A user identity credential (a username/password pair, a signed SAML 1.1 assertion, or WLES cookie)
   - The resource to which the user wants access and the action to be performed
   - The Authentication Service and SSM's Configuration ID
2. The Web Services SSM specified by the Configuration ID queries the local Registry Service to retrieve the fully qualified URL for the endpoint of the authentication service. If the authentication service exists on the local machine, the SSM returns the URL to the client. If it does not exist, the request is ignored locally and serviced by a remote Registry Service. If the Configuration ID is omitted from the client request, the URL for the default authentication service on the local machine is returned.
3. The client uses the URL to resubmit the request to the Web Services SSM Authentication Service.

4. The SSM calls the Authentication Service, uses the user credentials to authenticate the user. If authentication succeeds, the SSM returns an identity assertion token that is an internal representation of the user's identity to the web services client. If the authentication fails, the SSM returns an AuthenticationFaliure SOAP Fault.
5. The client includes the identity assertion token in the user request and resubmits it to the SSM. This request includes the following:
   - An identity assertion token that is an internal representation the user's identity on the Web Services SSM
   - The resource to which the user wants access and the action to be performed
   - The Authorization Service and SSM's Configuration ID
6. The Web Services SSM specified by the Configuration ID queries the local Registry Service to retrieve the fully qualified URL for the endpoint of the Authorization Service.
7. The SSM uses the Authorization Service and the associated Role Mapping Service to determine whether the user has been granted the privileges required to access to the specified resource and perform the requested action. The SSM answers the question "Is this user allowed to perform the particular action on specified resource?" To make the access decision, the SSM compares the roles granted to the user to the policy written to protect the requested resource. If none of the user's roles match the roles allowed to access the requested resource, or if the resource does not have any policy to protect it, access is denied.
8. If access is allowed, the user is granted access to the resource and the request is serviced. If access is denied, the Web Services SSM returns an AuthorizationFaliure SOAP Fault to the client.

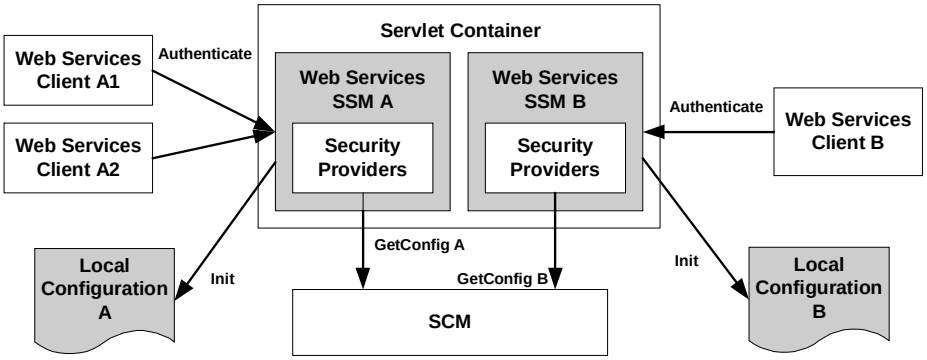
Product Features

The Web Services Security Service Module (SSM) has the following features:

- **Standalone Deployment**

  Each instance of a Web Services SSM is made accessible to clients via a separate SOAP endpoint with a unique URL. Further, each security service is deployed as a separate component inside the hosting process, with each service using disparate configuration entry to identify and initialize itself. Figure 1-8 shows how each Web Services SSM is deployed and initialized in an environment where multiple Web Services SSMs are deployed.

Figure 1-8 Multiple Web Services SSM Deployments



- **Credentials Protection**

  Identity assertions must be signed by a trusted entity. An assertion that is not signed or signed by an unknown authority is rejected and the processing stopped. The digital signature is attached to the identity assertion and covers the entire assertion.

- **SOAP 1.1 Support**

  The Web Services SSM supports SOAP 1.1 as the message format for web services clients invoking operations on the Web Services SSM.

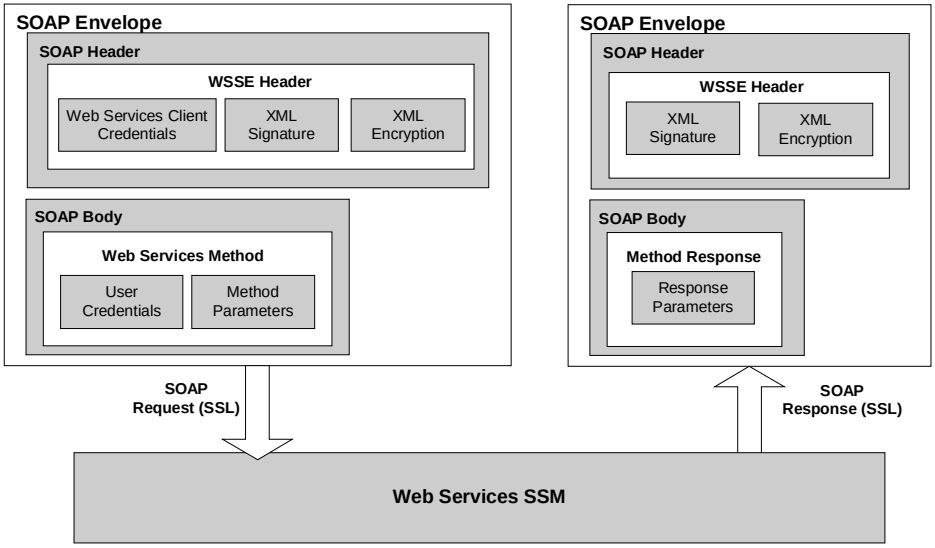
- **Support for Both RPC-Oriented and Document-Oriented Operations**

  Web Services SSM operations can be either RPC-oriented (SOAP messages contain parameters and return values) or document-oriented (SOAP messages contain documents.).

- **XML Structures Follow Industry Standards**

  The XML structures used for transmitting identity assertion tokens and other credential types follow existing industry standards, particularly those defined by the OASIS WS-Security Standard v1.0. See Figure 1-9 for the general format of the SOAP messages exchanged between the web services client and the Web Services SSM.

Figure 1-9 SOAP Message Format



- Secure Web Services security service APIs

  Access to the Web Services SSM security service APIs is restricted to clients that have been authenticated.

- **SOAP Interoperability**

  The Web Services security service APIs are able to interoperate with the most commonly used toolkits. At the minimum, the APIs are fully interoperable with web services clients that are written using the gSOAP v2.2.3 Framework.

- **Channel and Message Protection**

  In order to protect messages in transit, the Web Services SSM supports a channel security protocol (SSL 3.0 or TLS 1.0) for all communication that takes place between the SSM and its clients.

- **Support for One-way and Two-Way SSL Client Connections**

  The Web Services SSM always authenticates itself to client using its X.509 site certificate.

A client presents its certificate as part of a two-way SSL handshake with the Web Services SSM servlet container. The client's identity, contained in the SSL certificate, is subsequently used for client authentication.

Note: Any certificate authority (CA) used to generate the client certificate must be manually added to the Web Services SSM `trust.jks` keystore if it is not already listed there.

- **Automatic Restart of Security Services**

  The Web Services SSM uses the WebLogic Enterprise Security process monitoring and management mechanism to automatically restarted a security service if it is found to be unresponsive. Upon restart, the service is initialized using the latest configuration and automatically resumes its normal operation.

- **Event Auditing Capabilities**

  The Web Services SSM relies on the WebLogic Enterprise Security auditing capabilities to provide a file-based, audit logging facility with configurable audit log filename. Any service or request-related failures produce an audit trail. Additionally, the following Web Service SSM instance lifecycle events produce audit trails:

  – Initialization complete, ready to serve requests

  – Shutdown initiated

  – Restarted after a process crash

- **Configuration Stored Locally**

  The Web Services SSM relies only on the local configuration for its operation. The local configuration includes all necessary information to start-up Web Services SSM process, identify its instance, and set up a two-way SSL connection to the local Service Control Manager (SCM) process.

- **SOAP Message Handlers to Access SOAP Messages**

  A SOAP message handler accesses the SOAP message and its attachment in both the request and response of the Web Services SSM. You can create handlers in the web services client that invoke the Web Services SSM.

Supported Web Services Standards

The Web Services Security Service Module supports the following standards:

- "SOAP" on page 1-16
- "WSDL 1.1" on page 1-16

SOAP

SOAP (Simple Object Access Protocol) is a lightweight XML-based protocol used to exchange information in a decentralized, distributed environment. The WebLogic Enterprise Security Web Services Security Service Module implements SOAP 1.1.

The protocol consists of:

- An envelope that describes the SOAP message. The envelope contains the body of the message, identifies who should process it, and describes how to process it.
- A set of encoding rules for expressing instances of application-specific data types.
- A convention for representing remote procedure calls and responses.

This information is embedded in a Multipurpose Internet Mail Extensions (MIME)-encoded package that can be transmitted over HTTP or other Web protocols. MIME is a specification for formatting non-ASCII messages so that they can be sent over the Internet.

The Web Services application programming Interface (API) is exposed to external clients, which may or may not be developed using the same SOAP stack implementation. Since SOAP specifications allow combinations of different communication modes (RPC versus. Document, encoded versus. literal), the exposed public interface is able to interoperate with the most commonly used toolkits. At the minimum, the Web Services public SOAP interface is fully interoperable with clients written using the gSOAP v2.2.3 framework.

For more information on SOAP, see SOAP 1.1 at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ and the SOAP 1.2 Recommendation at: http://www.w3.org/TR/soap/.

WSDL 1.1

The Web Services Description Language (WSDL) is an XML-based specification that describes a web service. A WSDL document describes web service operations, input and output parameters, and how a client application connects to the web service.

For more information, see http://www.w3.org/TR/wsdl.

CHAPTER 2

Web Services Interfaces

To develop web services clients, you use the Web Services Security Service Module (SSM) application programming interfaces (APIs) developed by BEA Systems.

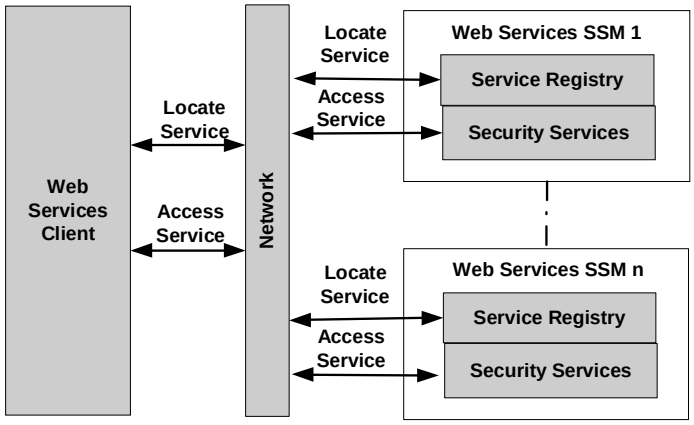
These interfaces are described in the following sections:

Registry Service Interface

To map security service type and SSM configuration and SOAP endpoints, the Web Services SSM includes a Registry Service. This service maps the SSM Configuration ID and the service type to the Web Services SSM and SOAP endpoints for the security services. The Registry Service makes it possible to distinguish between multiple instances of Web Server and Web Services SSMs on the same machine or on remote machines on the network as well as between the supported service types (auditing, authentication, authorization, credential mapping, and role mapping). Registry Service operations on a particular machine are limited to the local machine

(see Figure 2-1). On each Web Services SSM host machine, the Registry Service has a static address.

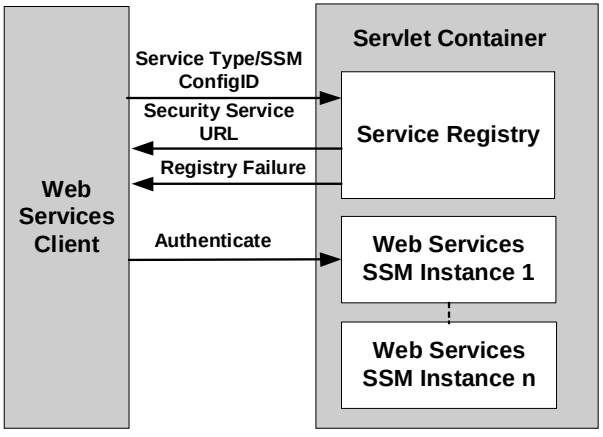
Figure 2-1 Registry Service Function



Registry Process

The Registry Service process is as follows (Figure 2-2 fig):

Figure 2-2 Registry Process



1. If a web services client asks the Registry Service about the existence of a particular security service type/Configuration ID combination on the local host, the Registry Service responds with a true or false answer.
2. If a web services client asks the registry for the URL of a valid security service type /Configuration ID combination for the local host, the Registry Service returns the fully qualified URL of the endpoint for the requested service type that is provided by the SSM configuration as defined by the Configuration ID.
3. If a web services client asks the registry for the URL of an valid security service type but does not supply a Configuration ID, the Registry Service assumes the default value for Configuration ID and returns the fully qualified URL of the endpoint for the requested service.
4. If a web services client asks the registry for a URL for a security service type/Configuration ID combination that has an invalid security service type for Web Services SSM identified by the Configuration ID, the Registry Service returns a RegistryFailure SOAP Fault.

Note: The Registry Service does not provide lifecycle management functions for the Web Services.

For more information on the Registry Service interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Registry Service Methods

The Registry Service supports two methods: `locateService()` and `doesServiceExist()`. Both methods accept the requested service type and SSM Configuration ID of the Web Services SSM that provides the service. For the `locateService()` method, the Registry Service returns the fully qualified URL for the endpoint of the requested service. For the `doesServiceExist()` method, the service returns a Boolean value (`true` or `false`) that indicates whether the service exists and can be requested.

Methods Common to All Web Services Interfaces

The following methods are supported by all Web Services interfaces, except for the Registry Service interface.

- `getServiceType()`—This method takes an empty request and returns a structure that contains the service. The Web Services SSM supports five security service types: authentication, auditing, authorization, credential mapping, and role mapping.

- `getServiceVersion()`—This method takes an empty request and returns a structure that contains the version of the service.
- `isCompatible()`—This method accepts service version information in its request and returns compatibility information. You use this method to determine whether the version of the service interface specified in the web services client is compatible with the current version of the service interface in the instance of the Web Services SSM.

Authentication Service Interface

The Authentication Service provides security functions to an application so as to establish, verify, and transfer a person or process identity. Thus, the Authentication Service provides two main security functions: authentication and identity assertion.

The Authentication Service accepts user credentials and/or identity assertion tokens and verifies that they match the user identity stored in the existing user profile. The following types of identity assertion tokens are supported:

- Username/password
- Signed Security Assertion Markup Language (SAML) 1.1 assertions
- ALES cookie

The Extensible Markup Language (XML) structures used by the Authentication Service for transmitting identity assertion tokens and other credential types follow the existing industry standards, particularly those defined by the OASIS Web Service—Security Standard v1.0.

To ensure secure handling of credentials, all credentials presented to the Web Services Security Service Module must satisfy the following requirements:

- All SAML 1.1 identity assertions must be signed by a trusted entity, as determined by the SAML IdentityAsserter. The signature must be attached to the identity assertion and cover the entire assertion. An identity assertion that is not signed or signed by an unknown authority is rejected and the processing is stopped.
- The caller should verify the validity of server credentials prior to invoking the Web Service. In particular, when a X.509 certificate is used as an identity assertion, the caller should verify its validity by examining its expiration date, certification path, and revocation status.

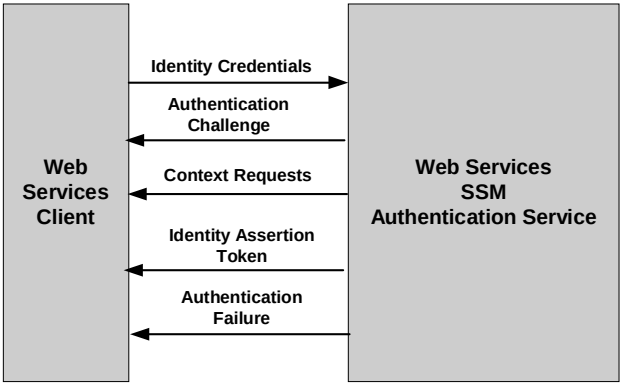
The Web Services SSM only accepts clear-text passwords. However, the credentials presented are always be protected by SSL, either one-way or two-way. The Web Services SSM returns credentials to clients over SSL as well.

The SOAP authentication interface enables the Web Services SSM to return challenges to clients if they fail to provides the information necessary to complete the authentication process. In such cases, the client can respond with the requested information. In order to avoid expensive round trips, however, the web services client should pass in all available credentials information with the initial SOAP request.

Authentication Process

The authentication process is as follows (see Figure 2-3):

Figure 2-3 Authentication Service Process



1. The web services client connects to the Web Services SSM over HTTP and the Web Services SSM responds by presenting a digital certificate to prove its identity to the client. The Web Services SSM authenticates itself to the web services client using its server X.509 certificate. If the Web Services SSM presents a certificate that is not valid (for instance, the certificate has expired or has a subject name mismatch), the client should break the connection.
2. The web services client verifies the SSMs digital certificate and submits a certificate signed by a recognized certificate authority (CA) to the Web Services SSM.
3. The Web Services SSM verifies the clients certificate and establishes an SSL connection.
4. The web services client collects user credentials from the user and submits those credentials to Web Services SSM to login. Web services clients present either of the following with each login request:
   – A valid service name/password pair or a signed SAML assertion

   – An ALES cookie

5. The Web Services SSM checks to verify that the client credentials match the credentials stored in the user's profile.
6. If the credentials match, the SSM returns an identity assertion token to the web services client in behalf of the user.
7. If the credentials do not match, the Web Services SSM submits the credentials to the Authentication Service for checking. The Authentication Service does one of two things:
   – Validates the user credentials, grants the login, and returns a success message to the Web Services SSM.
   – Invalidates the user credentials, blocks the login, and returns a failure message to the Web Services SSM.
8. If an authentication decision cannot be made with the parameters included in the initial client request, the Authentication Service returns a context request to the web services client indicating which parameters are missing.
9. If authentication is successful, the Web Services SSM returns the default identity assertion token representing the user. The type of the default identify assertion token is configurable. If the type is not specified, SAML assertions are used.
10. If authentication fails, the Web Services SSM returns an `AuthenticationFailure` SOAP Fault to the caller.

For more information on the Authentication Service Interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Authentication Service Methods

To support authentication functions the authentication provides the following methods:

- "authenticate() Method" on page 2-7
- "assertIdentity() Method" on page 2-8
- "isAssertionSupported() Method" on page 2-8
- "validateIdentity() Method" on page 2-9

authenticate() Method

This method accepts any credential type supported by the authentication provider or a response to an earlier authentication challenge, and, optionally, the type of requested identity assertion that represents the identity and application context of the authenticated user. In response, it returns either the requested identity assertion token, an authentication challenge, or additional context requests, if a challenge is required by the specific authentication provider or the authentication protocol.

Note: In addition to the identity assertion types, the Web Services SSM supports user credentials in form of usernames with passwords.

Table 2-1 describes the `authenticate()` method parameters.

Table 2-1 authenticate() Method Parameters

Parameter	Description
`AuthenticationChallenge`	In response to an authentication challenge, specifies the identity assertion token requested by the authentication challenge.
`IdentityCredential`	Specifies the identity credential that is to be used to authenticate the caller.
`AssertionCredentialType`	Specifies the identity assertion token type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.
`Context`	Specifies the application context in which authentication is being requested.

assertIdentity() Method

This method accepts any supported identity assertion type or a response to an earlier authentication challenge, and, optionally, the type of requested identity assertion that represents the identity and application context of the authenticated user. In response, it returns either the requested identity assertion token, an authentication challenge, or additional context requests, if required by the specific authentication provider or the authentication protocol.

Table 2-2 describes the `assertIdentity()` method parameters.

Table 2-2 assertIdentity() Method Parameters

Parameter	Description
`AuthenticationChallenge`	Specifies the information requested by the authentication challenge.
IdentityAssertion	Specifies the identity assertion.
`AssertionCredentialType`	Specifies the identity assertion token type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.
`Context`	Specifies the application context in which authentication is being requested.

isAssertionSupported() Method

This method accepts an identity assertion token type that represents the authenticated user's identity. It returns a Boolean value (`true` or `false`) to indicate whether this token is supported by this instance of the Web Services SSM.

Table 2-3 describes the `isAssertionSupported()` method parameters.

Table 2-3 isAssertionSupported() Method Parameters

Parameter	Description
`AssertionCredentialType`	Specifies the identity assertion token type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.

validateIdentity() Method

This method accepts any supported identity assertion type that represents the identity of the authenticated user. It returns a structure with a Boolean value (`true` or `false`) that indicates the authenticity of the token.

Table 2-4 describes the `validateIdentity()` method parameter.

Table 2-4 validateIdentity() Method Parameters

Parameter	Description
IdentityAssertion	Specifies the identity assertion token.

Authorization Service Interface

The Authorization Service is a service that allows an application to determine if a specific identity is permitted to access a specific resource. This decision may then be enforced in the application directly at the policy enforcement point.

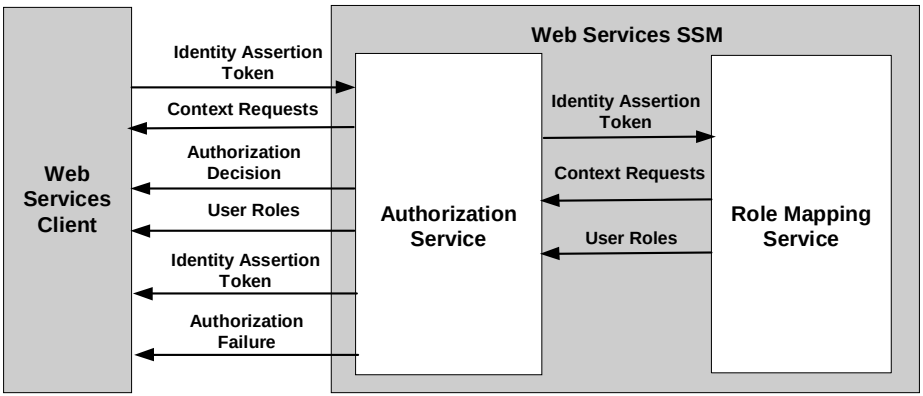
The Authorization Service is primarily based on a single method: `isAccessAllowed()`. This method accepts a supported type of user credential or an identity token, a runtime resource, and a runtime action. Optionally, this method can accept the type of the requested identity assertion token that represents the identity of the authenticated user, the application context, and direction

parameters. The `isAccessAllowed()` method requires that a valid, authenticated identity or a null identity token (representing an anonymous identity) be present when requesting an access decision.

Authorization Process

The authorization process is as follows (see Figure 2-4):

Figure 2-4 Authorization and Role Mapping Process



1. The authorization process begins when the web services client calls the Web Services SSM to answer the question "Is this user allowed to perform this particular action on the specified resource?" The Authorization Service accepts valid identity assertion tokens and checks the credential cache to verify the token. Identity assertion token types supported include ALES cookies and signed SAML 1.1 assertions.
2. If the token matches a credential in the cache, the SSM sets the user identity to the internal identity representation, which includes the user identity and the list of roles the user has been granted.
3. If the token does not match any of the credentials in the cache, the SSM calls the Authorization Service and the Role Mapping Service to determine whether the user is authorized to access resources and to get the list of roles the user has been granted. If the user is authorized, the user identity is set to the internal identity representation, which includes the user identity and the list of roles the user has been granted.

4. The Authorization Service then compares the user identity to the resource security policy to determine whether the user is in a role that has been granted the access privilege that is necessary to perform the requested action on the particular resource.
5. If an authorization decision can not be made with the parameters included in the initial client request, the Authorization Service returns a context request to the web services client indicating which parameters are missing.
6. The web services client must fill in the application context with the required parameters and re-submit the request. The Web Services SSM supports six context types: `StringValue`, `BoolValue` type, `DateTimeValue`, `TimeValue`, `IntValue`, and `IpValue`.
7. If the authorization process fails due to parameter-related problems, an `AuthorizationFailure` SOAP Fault is returned to the caller.
8. In the absence of an error, the authorization decision that is returned to the web services client contains a clear and unambiguous true or false statement that allows or disallows access to the resource in question. A negative authorization decision is a valid result and does not result in a SOAP Fault.

For more information about the Authorization Service interface, see the *WSDLdocs for Web Services Interfaces*.

Authorization Service Methods

The Authorization Service supports the following methods:

- "isAccessAllowed()" on page 2-11
- "isAuthenticationRequired() Method" on page 2-12

isAccessAllowed()

This method accepts a supported type of an identity assertion token, and a runtime resource and action structures. Optionally, it can accept type of the requested identity assertion token, (representing the authenticated user's identity), application context, and authorization direction parameters. In response, this method returns the authorization decision (optionally accompanied by the time-to-live (TTL) value), an identity Assertion token, and a list of user roles, or, if required by the authorization provider, additional context requests.

Table 2-5 describes the `isAccessAllowed()` method parameters.

Table 2-5 isAccessAllowed() Method Parameters

Parameter	Description
`Identity Assertion`	Specifies the identity assertion token type. The type specified must be supported by the credential mapping provider; otherwise, the service returns `CredentialMappingFailure` to the caller.
`RuntimeResource`	Specifies the runtime resource that the caller is requesting.
`RuntimeAction`	Specifies the runtime action that the caller is requesting.
`AssertionCredentialType`	Specifies the identity assertion token type. The type specified can be any type supported by the ALES Credential Mapping provider. The Authentication Service may also return an authentication challenge to the caller requesting other assertion token types. If the caller fails to specify an assertion token type supported by the Authentication Service even after challenges, the Authentication Service returns `CredentialMappingFailure` to the caller.
`Context`	Specifies the application context in which access to the resource is being requested.
`AZDirection`	Specifies authorization direction parameters.

isAuthenticationRequired() Method

The Authorization Service interface also supports the `isAuthenticationRequired()` method. This method accepts a runtime resource and a runtime action. It returns a Boolean value (`true` or `false`) that indicates whether authentication is require to access this resource. The web services client uses this method to test whether privileges are required to access a particular resource.

Table 2-6 describes the `isAuthenticationRequired()` method parameters.

Table 2-6 isAuthenticationRequired() Method Parameters

Parameter	Description
`RuntimeResource`	Specifies the runtime resource that the caller is requesting.
`RuntimeAction`	Specifies the runtime action that the caller is requesting.

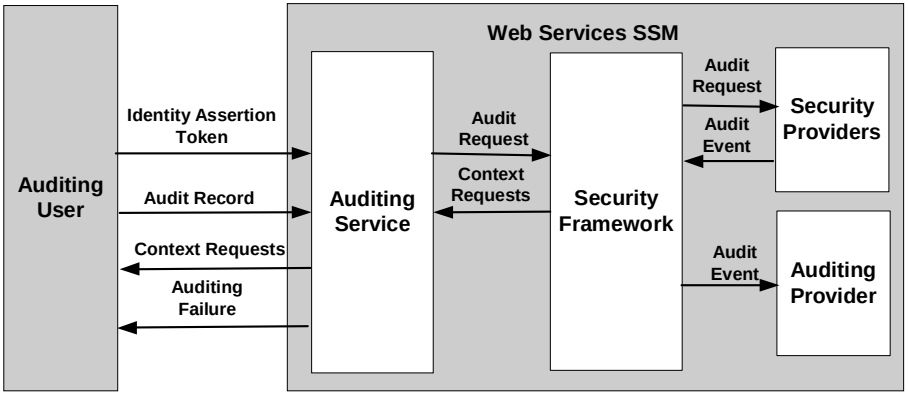
Auditing Service Interface

The Auditing Service logs events based on activity related to enterprise security. The Web Services Security Service Module (SSM) runtime uses the Auditing Service to log appropriate data when events occur. The Auditing Service is based on an event model. When something of note occurs, an auditing event is automatically logged. A user or a application that wants notification when a particular event occurs can derive a new class from the `AuditRecord` class or use the `AuditRecord` directly, as it is a named object.

Auditing Process

The auditing process is as follows (see Fig):

Figure 2-5 Auditing Process



1. During the auditing process, the Web Services SSM audit logging function supports automated, centralized logging of audit messages. To capture a particular event for notification purposes an auditing user can use the `recordEvent()` method to specify the name of the audit record to be captured, and, optionally, the identity assertion token of the auditing user, and the application context.
2. If the auditing provider requires application context that is not included in the `recordEvent()` method, the Auditing Service returns requests for additional application context.
3. If parameter-related audit logging failures occur, including passing in an invalid identity assertion token for the user, an `AuditingFailure` SOAP Fault is returned to the caller.
4. If no errors occur during the auditing process, an empty response is returned to the auditing user as an event notification and the audit event is logged by the auditing provider.

For more information on the Auditing Service interface and the methods it supports, see *WSDLdocs for Web Services Interfaces*.

Auditing Service Method

The Auditing Service passes the audit event to the Web Services SSM runtime. Based on its configuration, the SSM runtime routes the event to the proper auditing providers so that it can be recorded.

The Auditing Service supports a single method, `recordEvent()`. Table 2-7 describes the `recordEvent()` method parameters.

Table 2-7 recordEvent() Method Parameters

Parameter	Description
`AuditRecord`	Specifies the name of the audit record that encapsulates the logging information.
`IdentityAssertion`	Provides an identity assertion token that represents the auditing user.
`Context`	Specifies the application context in which request for an auditing record is being made.

Role Mapping Service Interface

The Role Mapping Service allows an application to extract role information about specific identities and resources within the context of the application. These roles can then be used for customizing an interface or for other purposes.

Note: Do not use roles by themselves for authorization, because many policies, allowing or disallowing access to a resource, may be written against a role. Use the Authorization Service to determine actual access rights.

The Role Mapping Service evaluates an interaction of an identity with a resource within an application context and returns a list of role names associated with the configuration of that identity. These roles can change with every resource or be static for the identity across all resources. The roles assigned to an identity are determined by the security policy.

The Web Services SSM is capable of retrieving the roles that a user may have for the given resource and action combination. The user identity is passed as an identity assertion token, instead of a Java object. To obtain roles, authenticated users must be authorized to obtain their roles for the given resource/action combination.

The Role Mapping Service requires that the application pass in a valid identity, a valid resource, and a valid action. The application context is optional and may be set to null if no context is passed in.

Role Mapping Process

The role mapping process is as follows (see Figure 2-4):

1. During the role mapping process, the Web Services SSM provides a mechanism for optionally specifying the application context to support role mapping.
2. If parameter-related role mapping failures occur, including passing in an invalid identity assertion token for the user, a `RoleMappingFailure` SOAP Fault is returned to the caller.
3. If no errors occur during the role mapping process, a list of zero or more roles, configured in the policy for the provided user/resource/action combination, is returned to the caller. An empty list is a valid response and does not result in a SOAP Fault.

For more information on the Role Mapping Service interface and the methods it supports, see *WSDLdocs for the Web Services Interfaces*.

Role Mapping Service Method

The Role Mapping Service interface supports one method, `getRoles()`. This method gets the roles for an authenticated user identity in reference to a `RuntimeResource`, `RuntimeAction`, and an optional `AppContext`.

The `getRoles()` method accepts a supported type of an identity token, and, optionally, runtime resource and action structures, and an application context. It returns either a list of user roles associated for the identity or, if such is required by the Role Mapping provider, additional context requests. If the identity provided is invalid or not properly authenticated, this method returns a SOAP fault.

Table 2-8 describes the `getRoles()` method parameters.

Table 2-8 getRoles Method Parameters

Parameter	Description
`Identity Assertion`	Specifies the identity assertion token. The token type must be supported by the Role Mapping provider; otherwise, the service returns `RoleMappingFailure` to the caller.
`RuntimeResource resource`	Specifies the runtime resource that the caller is requesting.
`RuntimeAction action`	Specifies the runtime action that the caller is requesting.
`Context`	Specifies the application context in which access to the resource is being requested.

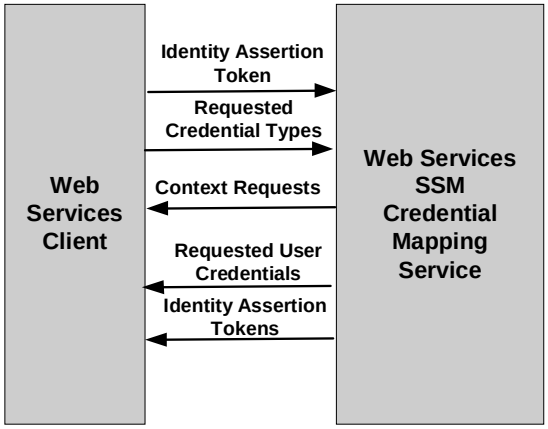
Credential Mapping Service Interface

The Credential Mapping Service allows an client application to fetch credentials of certain types that are associated with a specific identity for a specific resource. These credentials can then be used on behalf of that identity to execute some privileged function, such as logging into a database or sending e-mail.

Credential Mapping Process

The credential mapping process is as follows (see Figure 2-6):

Figure 2-6 Credential Mapping Process



1. The client presents a get credentials request to the Web Services SSM. The request includes a supported type of an identity assertion token and a list of requested credential types. Optionally, the request can include a runtime resource, runtime action, and an application context.
2. In response, the Web Services SSM returns a list of requested user credentials and identity assertion tokens or, if required by the Credential Mapping provider, additional context requests.
3. The Web Services SSM returns the following types of credentials in response to the client's request:
   - Username/password pairs
   - Signed SAML 1.1 assertions
   - ALES cookies
4. If parameter-related credential mapping failures occur, the Web Service returns a `CredentialMappingFailure` SOAP Fault to the caller.

5. If no errors occur during the credential mapping process, the Web Services SSM returns a list of zero or more credentials that are configured in the security policy for the specified user/resource/action combination. An empty list is a valid response and does not result in a SOAP Fault. Also, if some of the requested credential types are not available for specified user/resource/action combination, a SOAP Fault does not result.

For more information on the Credential Mapping Service interface and the methods it supports, see WSDLdocs *for Web Services Interfaces*.

Credential Mapping Method

The Credential Mapping Service supports one method: `getCredentials()`. This method accepts a supported type of an identity assertion token and a list of requested credential types. Optionally, this method can accept an identity assertion token that represents the identity of a different user and a runtime resource structure, which includes the requested resource and action and the application context. In response, the `getCredentials()` method returns either a list of requested user credentials, identity assertion tokens, or, if required by the ALES Credential Mapping provider, context requests.

Note: Since password credentials need to be returned in clear text to the caller in order to be usable for authentication in external systems, you should pay particular attention to providing channel and message security to protect messages in transit between clients and the Web Service. At a minimum, you must use a channel security protocol, such as SSL or TLS, for all communication.

Authenticated users are always authorized to obtain their own credentials for the given resource/action combination. However, authenticated user cannot requests credentials on behalf of another user.

Credential mapping process involves issuing new types of credentials to the specified combination of user, resource, and action. The user identity is passed as an identity assertion token, instead of a Java object.

Table 2-9 describes the `getCredentials()` method parameters.

Table 2-9 getCredentials Method Parameters

Parameter	Description
`Identity Assertion`	Specifies the identity assertion token. The type used must be supported by the Credential Mapping provider; otherwise, the service returns `CredentialMappingFailure` to the caller.
`CredentialTypes`	Specifies a list of the types of credentials being requested. The Web Services SSM supports the following types of credentials: • Username/password pairs • Signed SAML 1.1 assertions • WLES proprietary cookie
`RuntimeResource` resource	Specifies the runtime resource that the caller is requesting.
`RuntimeAction action`	Specifies the runtime action that the caller is requesting.
`AppContext context`	Specifies the application context in which access to the resource is being requested.