Oracle® Fusion Middleware

Understanding Oracle Web Services Manager

12*c* (12.2.1.3.0)
E98747-01
August 2018

ORACLE

Oracle Fusion Middleware Understanding Oracle Web Services Manager, 12*c* (12.2.1.3.0)

E98747-01

Primary Authors: Anupam Das, Showvik Roychowdhuri

Contents

Preface

Audience ix
Documentation Accessibility ix
Related Documents ix
Conventions x

What's New in This Guide

New and Changed Features for 12c (12.2.1.3.0) xi
New and Changed Features for 12c (12.2.1.2.0) xi
New and Changed Features for 12c (12.2.1.1.0) xi

1 Introducing Oracle Web Services Manager

1.1 Overview of Oracle Web Services Manager 1-1
1.2 Overview of Oracle Web Services Manager Features 1-2
1.3 Overview of Oracle Web Service Manager Architecture 1-3

2 Understanding Web Service Security Concepts

2.1 About Web Service Security 2-2
2.2 Understanding Transport-level and Application-level Security 2-3
2.3 Understanding Authentication 2-4
2.3.1 About Digest Authentication 2-5
2.4 Understanding Authorization 2-5
2.5 Overview of Message Protection 2-6
2.5.1 Understanding Message Protection 2-6
2.5.2 About Message Encryption 2-7
2.5.3 About Message Signing (XML Signature) 2-8
2.6 Overview of the Roles of Keys and Certificates in Security and Authentication 2-8
2.6.1 About Private Keys and Certificates 2-9
2.6.2 Understanding How Different Security Policies Use Private Keys and Certificates 2-11

2.6.2.1 Overview of Message Protection Policy Types 2-11
2.6.2.2 Overview of Authentication Token Policy Types 2-13
2.6.3 How OWSM Locates Keystore and Key Passwords for the JKS Keystore 2-15
2.6.4 About Private Keys and Certificates Configuration for SSL Policies 2-16
2.6.5 About Setting up Private Keys and Certificates for Message Protection Policies 2-18
2.6.5.1 Understanding Sample Basic Configuration 2-18
2.6.5.2 About Advanced Setup Considerations 2-18
2.7 Understanding How OWSM Uses the Credential Store 2-19
2.8 Understanding Security Policies 2-20
2.9 Overview of Security Tokens 2-21
2.9.1 Understanding Security Tokens 2-21
2.9.2 About the Username Token 2-22
2.9.3 About the X.509 Certificate 2-22
2.9.4 About the Kerberos Token 2-22
2.9.5 About the SAML Token 2-22
2.10 Understanding Secure Attachments 2-24
2.11 Overview of Secure Conversation 2-24
2.11.1 About Secure Conversation 2-24
2.11.2 Overview of WS-SecureConversation Usage 2-25
2.11.2.1 When to Use WS-Secure Conversation 2-25
2.11.2.2 Benefits of WS-SecureConversation 2-26
2.11.2.3 About WS-SecureConversation With WS-ReliableMessaging 2-27
2.11.3 WS-SecureConversation Architecture 2-27
2.11.4 When to Use WS-SecureConversation 2-29
2.11.5 When To Use Re-Authentication 2-30
2.11.6 About Setting the Bootstrap Mode 2-30
2.11.7 Overview of Persistence 2-30
2.11.7.1 About Default Domain-Wide Persistence Implementation 2-31
2.11.7.2 About Client- and Web Service-Specific Persistence Implementation 2-31
2.12 Overview of the Kerberos Protocol 2-31
2.12.1 Understanding the Kerberos Protocol 2-31
2.12.2 Understanding Credential Delegation in Kerberos 2-32
2.12.3 Understanding Kerberos and SPNEGO 2-33
2.12.4 About Kerberos and WS-SecureConversation Derived Keys 2-34
2.13 Understanding Web Services Addressing 2-34
2.14 Understanding Web Services Trust 2-35
2.15 Understanding Web Services ReliableMessaging 2-36
2.16 Overview of Fine-Grained Authorization Using Oracle Entitlements Server 2-37
2.16.1 References for OES Reading 2-37

2.16.2 Overview of OES Integration 2-38
2.16.2.1 OES Integration: The Big Picture 2-38
2.16.2.2 Data Masking 2-39
2.16.2.3 About XACML Obligations 2-41
2.16.2.4 Overview of OES Fine- and Coarse-Grained Authorization 2-41
2.16.3 About OWSM OES Policies 2-44
2.16.4 Overview of Resource Mapping and Naming 2-45
2.16.4.1 Resource Mapping and Naming 2-45
2.16.4.2 Example of OES Policies 2-46
2.16.5 How Attributes Are Processed 2-48
2.16.6 About the Guard Element 2-50
2.17 Overview of Personally Identifiable Information 2-51
2.17.1 Overview of PII Data 2-51
2.17.1.1 About PII Data 2-51
2.17.1.2 About the PII Security Policy 2-52
2.17.2 Example of How PII Data is Protected 2-53
2.17.3 About PII Policy XPath Expressions 2-54
2.17.4 When to Use the PII Policy 2-55
2.17.4.1 Single SOA Composite Use Case 2-56
2.17.4.2 Oracle Service Bus Proxy Service to Business Service Use Case 2-57
2.17.4.3 PII at the JCA Binding Use Case 2-57
2.17.5 Who Should Have Access to the PII 2-59
2.17.6 About Additional Considerations for Unmarshalling 2-59
2.18 Understanding OAuth 2.0 for REST and SOAP Services and Clients 2-60
2.19 Understanding REST APIs for Managing Credentials and Keystores 2-60

3 Understanding the OWSM Policy Framework

3.1 Overview of OWSM Policy Framework 3-1
3.1.1 About OWSM Policy Framework Components 3-1
3.1.2 Understanding OWSM Agent and Policy Manager Interaction 3-2
3.1.3 About OWSM Agent and Policy Manager Characteristics 3-4
3.1.4 Understanding the OWSM Agent and Policy Manager Request Flow 3-4
3.1.5 About OWSM Configuration Artifacts 3-4
3.2 Understanding Web Service Policies 3-5
3.3 Overview of Building Web Service Policies Using Policy Assertions 3-7
3.3.1 About Building Web Service Policies Using Policy Assertions 3-7
3.3.2 About Defining Multiple Policy Alternatives (OR Groups) 3-9
3.4 Understanding Policy Subjects 3-10
3.5 Overview of Attaching Policies to Policy Subjects 3-12
3.5.1 About Attaching Policies to Policy Subjects 3-13

3.5.2 About Direct Policy Attachment 3-13
3.5.3 Overview of Global Policy Attachments Using Policy Sets 3-13
3.5.3.1 Understanding Global Policy Attachments Using Policy Sets 3-14
3.5.3.2 About Subject Types and Scope of Resources 3-15
3.5.3.3 Understanding Typical Uses for Global Policy Attachments 3-15
3.6 Understanding How Policies are Executed 3-15
3.7 About OWSM Predefined Policies and Assertion Templates 3-17
3.8 About Overriding the Security Policy Configuration 3-18
3.9 About Recommended Naming Conventions for Documents Created in WSM Repository 3-18

A Web Service Security Standards

A.1 Security Standards A-1

List of Figures

1-1	Security Provided by OWSM Agents	1-1
1-2	Components of OWSM Architecture	1-4
2-1	OWSM Keystore Configuration for Message Protection	2-16
2-2	STS Token Exchange	2-36
2-3	Masking Sensitive Data	2-40
2-4	Fine-Grained Authorization	2-42
2-5	Coarse-Grained Authorization	2-44
2-6	PII Encryption in Oracle Service Bus	2-53
2-7	Single SOA Composite Use Case	2-56
2-8	JCA Adapter PII Use Case	2-58
3-1	OWSM Policy Framework Leverages OPSS and Oracle WebLogic Server Security	3-2
3-2	OWSM Agent and Policy Manager Interaction	3-3
3-3	Policy Containing Assertions	3-8
3-4	Example Policy With Two Assertions	3-8
3-5	Policy Interceptors Acting on Messages Between a Client and Web Service (SOAP)	3-16
3-6	Identifying the Different Parts of a Policy Name	3-19

List of Tables

1-1	Components of OWSM Architecture	1-4
2-1	Determining Resource String	2-46
2-2	Resource String Example	2-46
2-3	Attribute Types Supported for OES Policies	2-48
3-1	Policy Categories	3-5
3-2	Policy Subjects and Resource Scopes	3-10
A-1	Web Services Standards and Specification URLs	A-1

Preface

This section describes the intended audience, how to use this guide, and provides information about documentation accessibility.

Audience

This guide is intended for:

- System and security administrators who administer Web services and manage security
- Application developers who are developing Web services and testing the security prior to deployment of the Web services
- Security architects who create security policies.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware Web services documentation set:

- *Administering Web Services*
- "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*
- *Developing Extensible Applications for Oracle Web Services Manager*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing JAX-WS Web Services for Oracle WebLogic Server*
- *Developing JAX-RPC Web Services for Oracle WebLogic Server*
- *Developing Oracle Infrastructure Web Services*

- *Interoperability Solutions Guide for Oracle Web Services Manager*
- *Developing SOA Applications with Oracle SOA Suite*
- *Developing for Oracle WebCenter Portal*
- *Securing WebLogic Web Services for Oracle WebLogic Server*
- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding WebLogic Web Services for Oracle WebLogic Server*
- *Understanding Web Services*
- *Use Cases for Securing Web Services Using Oracle Web Services Manager*
- *WebLogic Web Services Reference for Oracle WebLogic Server*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide

The following topics introduce the new and changed features of Oracle Web Services Manager (OWSM) and other significant changes that are described in this guide, and provides pointers to additional information.

New and Changed Features for 12c (12.2.1.3.0)

This topic contains the New and Changed Features for Release 12c (12.2.1.3.0).

Minor updates, such as fixes or corrections, were made to this document.

New and Changed Features for 12c (12.2.1.2.0)

This topic contains the New and Changed Features for Release 12*c* (12.2.1.2.0).

Minor updates, such as fixes or corrections, were made to this document.

New and Changed Features for 12c (12.2.1.1.0)

This topic contains the New and Changed Features for Release 12*c* (12.2.1.1.0).

Minor updates, such as fixes or corrections, were made to this document.

1

Introducing Oracle Web Services Manager

Oracle Web Services Manager (OWSM) provides a policy framework to manage and secure Web services consistently across your organization. It provides capabilities to build, enforce, run and monitor Web service policies, such as security, reliable messaging, MTOM, and addressing policies. OWSM can be used by both developers, at design time, and system administrators in production environments.
For more information, refer to the following sections:

- Overview of Oracle Web Services Manager
- Overview of Oracle Web Services Manager Features
- Overview of Oracle Web Service Manager Architecture

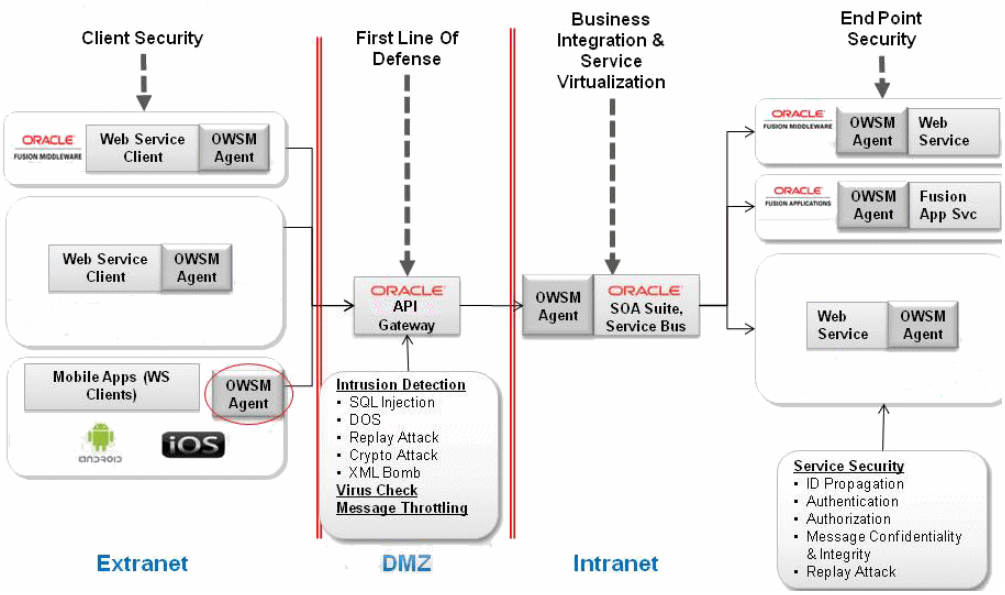
For definitions of unfamiliar terms found in this and other books, see the Glossary.

1.1 Overview of Oracle Web Services Manager

Oracle Web Services Manager (OWSM) provides business agility to respond to security threats and security breaches by allowing policy changes to be enforced in real time without the need to interrupt the running business processes.

As shown in Figure 1-1, OWSM provides the "first mile security" via client agents for securing Web service clients, and "last mile security" via server agents securing Web services. If your Web services are accessible only from inside the corporate intranet, they typically still require authentication and authorization. In addition, auditing is often required to address regulatory compliance.

Figure 1-1 Security Provided by OWSM Agents



OWSM allows for policy-driven centralized management of Web services with local enforcement. OWSM provides a policy framework to manage and secure Web services consistently across your organization.

The benefits of this policy driven approach include:

- Allows security to be declarative and externalized.
- Provides business agility to respond to security threats and security breaches by allowing policy changes to be enforced in real time without the need to interrupt the running business processes.
- Avoids the need for developers to understand security specifications and security implementation details.

OWSM allows you to:

- Centrally define and store declarative policies applied to the multiple Web services.
- Locally enforce policies through configurable agents.
- Monitor run time security events such as failed authentication or authorization.

You can use OWSM to secure the following categories of Oracle Web services:

- Oracle Infrastructure web services—SOA, Application Development Framework (ADF and WebCenter), Oracle Service Bus, and Oracle Enterprise Scheduler services
- Java EE web services—SOAP (JAX-WS) and RESTful (JAX-RS) web services

Companies worldwide are actively deploying service-oriented architectures (SOA) using Web services, both in intranet and internet environments. While Web services offer many advantages over traditional alternatives (for example, distributed objects or custom software), deploying networks of interconnected Web services still presents key challenges, particularly in terms of security and administration.

1.2 Overview of Oracle Web Services Manager Features

OWSM includes an extensive array of policy and management features. Security standards supported and specific tasks performed using OWSM is discussed in the following section.

Following OWSM features are included:

- Policy Management:
  - Global and direct policy attachment.
  - Policy attachment at design time and post-deployment.
  - Ability to attach/detach multiple policies to a Web service or client.
  - Auto-select of client policies.
  - Identity propagation across multiple Web services.
  - Policy advertisement in WSDL.
- Monitoring/Management:
  - Centralized management, auditing and reporting.
  - Policy versioning and rollback.

  - Performance management, including metrics for service, port, and operation, policy dependencies per port, number of security violations, number of invocations, and more.
  - Policy export and import.
  - Policy impact analysis.
- Security standards supported:
  - A broad range of security standards is supported, as described in Table A-1.
  - Pre-defined, reusable policies, including security, reliability, addressing, management and MTOM policies.
  - Custom policy extensions.

OWSM supports policy attachment at both design time and post-deployment, which provides capabilities for both developers and system administrators:

- Developers can attach OWSM policies from the Oracle JDeveloper context menu and property inspector. For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.
- System administrators can leverage OWSM through the Oracle Enterprise Manager Fusion Middleware Control and WLST. They can centrally define policies using the OWSM Policy Manager and enforce OWSM polices locally at run time.

Examples of specific tasks that you can perform using OWSM include the following:

- Handle WS-Security (for example, encryption, decryption, signing, signature validation, and so on).
- Define authentication and authorization policies against an LDAP directory.
- Generate standard security tokens (such as SAML tokens) to propagate identities across multiple Web services used in a single transaction.
- Segment policies into different namespaces by creating policies within different folders.
- Examine log files.

1.3 Overview of Oracle Web Service Manager Architecture

The Oracle Web Services Manager (OWSM) agent, policy manager, and repository are the main components in the OWSM architecture.

Figure 1-2 illustrates the interaction among the main OWSM components and the Oracle Fusion Middleware Control console.

> **Note:**
>
> A subset of OWSM policies are supported for RESTful Web services, as described in Which OWSM Policies Are Supported for RESTful Web Services? in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. The subset does not include all of the policy interceptor types shown in Figure 1-2.

Figure 1-2 Components of OWSM Architecture

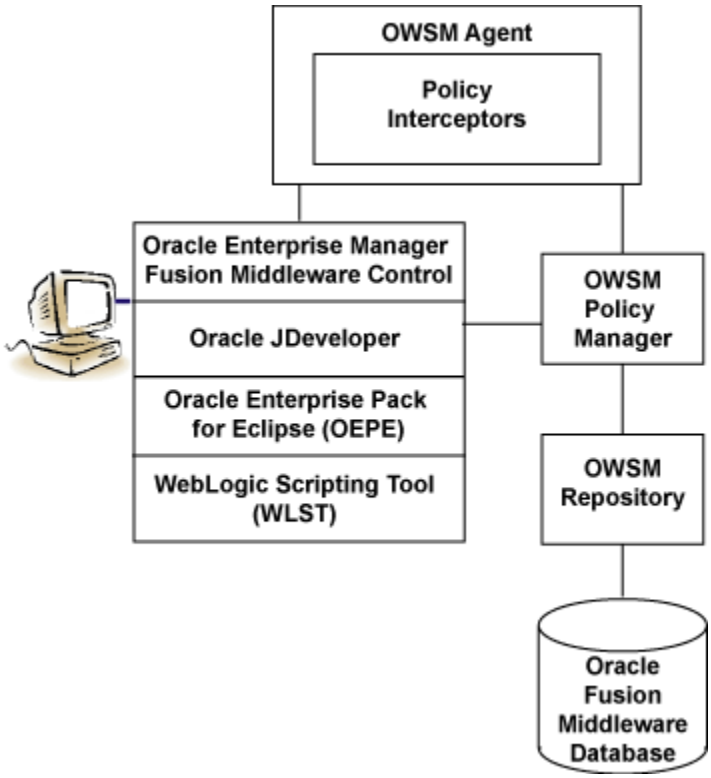


Table 1-1 describes the components of OWSM shown in Figure 1-2, and highlights their use in the figure.

Table 1-1 Components of OWSM Architecture

OWSM Component	Description
Oracle Enterprise Manager Fusion Middleware Control	Enables administrators to access OWSM's functionality to manage, secure, and monitor Web services.
Oracle JDeveloper	Provides a full-featured Java IDE that can be used for end-to-end development of Web services. Using visual and declarative tools, developers can build Oracle SOA, ADF, WebCenter, and WebLogic Java EE Web services, automatically deploy them to an instance of Oracle WebLogic Server, and immediately test the running Web service. Alternatively, JDeveloper can be used to drive the creation of Web services from WSDL descriptions. JDeveloper is Ant-aware. You can use this tool to build and run Ant scripts for assembling the client and for assembling and deploying the service. For more information, see the Oracle JDeveloper online help.
WebLogic Scripting Tool (WLST)	Enables administrators to view and configure Web services, and manage Web service policies from the command line.

Table 1-1 (Cont.) Components of OWSM Architecture

OWSM Component	Description
OWSM Policy Manager	Reads/writes the policies, including predefined and custom policies from the OWSM Repository.
OWSM Agent	Manages the enforcement of policies via the Policy Interceptor Pipeline.
Policy Interceptors	Enforces policies. For more information, see "Understanding How Policies are Executed".
OWSM Repository	Stores OWSM metadata, such as policies, policy sets, assertions templates, and policy usage data. The OWSM Repository is available as a database (for production use) or as files in the file system (for development use in JDeveloper).
Oracle Fusion Middleware Database	Provides database support for the OWSM Repository.

Subsequent chapters of this document describe conceptual information about the OWSM policy framework and security concepts. This document also includes a section on the security standards for Oracle Infrastructure Web Services.

The companion documents *Securing Web Services and Managing Policies with Oracle Web Services Manager* and *Administering Web Services* describe how to secure and administer Web services using OWSM, respectively.

2

Understanding Web Service Security Concepts

Web services security encompasses a number of requirements, such as authentication, authorization, and message protection.
Web Services Security concepts are described in the following sections:

- About Web Service Security
- Understanding Transport-level and Application-level Security
- Understanding Authentication
- Understanding Authorization
- Overview of Message Protection
- Overview of the Roles of Keys and Certificates in Security and Authentication
- Understanding How OWSM Uses the Credential Store
- Understanding Security Policies
- Overview of Security Tokens
- Understanding Secure Attachments
- Overview of Secure Conversation
- Overview of the Kerberos Protocol
- Understanding Web Services Addressing
- Understanding Web Services Trust
- Understanding Web Services ReliableMessaging
- Overview of Fine-Grained Authorization Using Oracle Entitlements Server
- Overview of Personally Identifiable Information
- Understanding OAuth 2.0 for REST and SOAP Services and Clients
- Understanding REST APIs for Managing Credentials and Keystores

> **Note:**
>
> A subset of OWSM authentication and authorization policies are supported for RESTful web services, as described in OWSM Policies Are Supported for RESTful Web Services in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
>
> This section primarily describes web services over SOAP.

2.1 About Web Service Security

Web services with its nature of loosely coupled connections and its use of open access like HTTP, adds a new set of requirements to the security landscape.

Because of its nature (loosely coupled connections) and its use of open access (mainly HTTP), SOA implemented by web services adds a new set of requirements to the security landscape.

Key components of web service security are:

- **Authentication**—Verifying that the user is who she claims to be. A user's identity is verified based on the credentials presented by that user, such as:

  1. Something one has, for example, credentials issued by a trusted authority such as a passport (real world) or a smart card (IT world).
  2. Something one knows, for example, a shared secret such as a password.
  3. Something one is, for example, biometric information.

  Using a combination of several types of credentials is referred to as "strong" authentication, for example using an ATM card (something one has) with a PIN or password (something one knows). See "Understanding Authentication" for more information.

- **Authorization (or Access Control)**—Granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute. See "Understanding Authorization" for more information.

- **Confidentiality, privacy**—Keeping information secret. Accesses a message, for example a web service request or an email, as well as the identity of the sending and receiving parties in a confidential manner. Confidentiality and privacy can be achieved by encrypting the content of a message and obfuscating the sending and receiving parties' identities. See "Overview of Message Protection" for more information.

- **Integrity, non repudiation**—Making sure that a message remains unaltered during transit by having the sender digitally sign the message. A digital signature is used to validate the signature and provides non-repudiation. The timestamp in the signature prevents anyone from replaying this message after the expiration. For more information, see "Overview of Message Protection".

Web services security requirements also involve credential mediation (exchanging security tokens in a trusted environment), and service capabilities and constraints (defining what a web service can do, under what circumstances).

In many cases, web services security tools such as OWSM rely on Public Key Infrastructure (PKI) environments. A PKI uses cryptographic keys (mathematical functions used to encrypt or decrypt data). Keys can be private or public. In an asymmetric cipher model, the receiving party's public key is used to encrypt plaintext, and the receiving party's matching private key is used to decrypt the ciphertext. Also, a private key is used to create a digital signature by signing the message, and the public key is used for verifying the signature. Public-key certificates (or certificates, for short) are used to guarantee the integrity of public keys.

Web services security requirements are supported by industry standards both at the transport level (Secure Socket Layer) and at the application level relying on XML frameworks.

For more information about the specifications and standards supported by web services, see Web Service Security Standards.

> **Note:**
>
> Oracle has been instrumental in contributing to emerging standards, in particular the specifications hosted by the OASIS Web Services Secure Exchange technical committee.

Oracle Web Services Manager (OWSM) is designed to define and implement web services security in heterogeneous environments, including authentication, authorization, message encryption and decryption, signature generation and validation, and identity propagation across multiple web services used to complete a single transaction.

The following summarize the web service security requirements:

- Use transport security to protect the communication channel between the web service consumer and web service provider.
- Use message-level security to ensure confidentiality by digitally encrypting message parts; integrity using digital signatures; and authentication by requiring username, X.509, or SAML tokens.

2.2 Understanding Transport-level and Application-level Security

Security concepts can be divided into transport level and application level security. Transport-level security secures the communications channel between applications.

An example of a transport-level security protocol is Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS), the Internet Engineering Task Force (IETF) officially standardized version of SSL. This is the most widely used transport-level data-communication protocol providing:

- Authentication (the communication is established between two trusted parties).
- Confidentiality (the data exchanged is encrypted).
- Message integrity (the data is checked for possible corruption).
- Secure key exchange between client and server.

SSL provides a secure communication channel, however, when the data is not "in transit," the data is not protected. This makes the environment vulnerable to attacks in multi-step transactions. (SSL provides point-to-point security, as opposed to end-to-end security.)

SSL can be used in three modes:

- No authentication: Neither the client nor the server authenticates itself to the other. No certificates are sent or exchanged. In this case, only confidentiality (encryption/decryption) is used.
- One-way authentication (or server authentication): Only the server authenticates itself to the client. The server sends the client a certificate verifying that the server is authentic. This is typically the approach used for Internet transactions such as online banking.
- Two-way authentication (or bilateral authentication): Both client and server authenticate themselves to each other by sending certificates to each other. This approach is necessary to prevent attacks from occurring between a proxy and a web service endpoint.

SSL uses a combination of secret-key and public-key cryptography to secure communications. SSL traffic uses secret keys for encryption and decryption, and the exchange of public keys is used for mutual authentication of the parties involved in the communication.

Application-level security complements transport-level security. Application-level security is based on XML frameworks defining message confidentiality, integrity, authenticity (also known as message protection); message structure; trust management and federation. These components of application-level security are described in greater detail in the following sections, "Overview of Message Protection", "Understanding Authentication", and "Understanding Authorization".

2.3 Understanding Authentication

Authentication is verifying that the user is who they claim to be based on the credentials.

A user's identity is verified based on the credentials presented by that user, such as:

- Something one has, for example, credentials issued by a trusted authority such as a digital certificate, standard Security Assertion Markup Language (SAML) token, or Kerberos token.
- Something one knows, for example, a shared secret such as a password.
- Something one is, for example, biometric information.

Using a combination of several types of credentials is referred to as "strong" authentication, for example using an ATM card (something one has) with a PIN or password (something one knows).

SAML is one of the most interesting security tokens because it supports both authentication and authorization. SAML is an open framework for sharing security information on the Internet through XML documents. SAML includes three parts:

- SAML Assertion—How you define authentication and authorization information.
- SAML Protocol—How you ask (SAML Request) and get (SAML Response) the assertions you need.
- SAML Bindings and Profiles—How SAML assertions ride "on" (Bindings) and "in" (Profiles) industry-standard transport and messaging frameworks.

The full SAML specification is used in browser-based federation cases. However, web services security systems such as OWSM only use SAML assertions. The protocol

and bindings are taken care of by WS-Security and the transport protocol, for example HTTP.

SAML assertions and references to assertion identifiers are contained in the WS-Security Header element, which in turn is included in the SOAP Envelope Header element (described in the WS-Security SAML Token Profile). The SAML security token is particularly relevant in situations where identity propagation is essential.

2.3.1 About Digest Authentication

OWSM supports digest based authentication in username-token authentication policies. Digital Authentication is an authentication mechanism in which a web application authenticates itself to a web service by sending the server a digest, which is a cryptographic hash of the password, nonce, and timestamp.

When using digest authentication:

1. The client makes an un-authenticated request to the web service, and the server sends a response with a digest authentication challenge indicating that it supports digest authentication.
2. The client generates a nonce and sends it to the service along with a timestamp, digest, and username. The digest is a cyptographic hash of the password, nonce, and timestamp.
3. The server generates the hash itself from the password (retrieved from the service store), nonce and timestamp (from the message), and if the generated hash matches the hash in the request, the request is allowed.

The advantage of digest authentication is it is resistant to replay attacks. The implementation maintains a cache of used nonces/timestamps for a specified period of time. All requests with a timestamp older than the specified timestamp are rejected as well as any requests that use the same timestamp/nonce pair as the most recent timestamp/nonce pair still in the cache. WebLogic Server stores this cache in a database.

2.4 Understanding Authorization

Authentication is the first step of determining whether a user should be given access to a web service. After the user is authenticated, the second step is to verify that the user is authorized to access the web service.

Authorization (also known as access control) is granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute.

Authorization enables you to determine what operations authenticated clients can access.There are three basic approaches to authorization:

- Role-based—Role-based security is based on the notion that a set of identities, known as principals, can be grouped into roles, and then a policy can be applied to each of the roles.
- Identity based—Identity Model enables you to manage claims and policies in order to authorize clients. With this approach, you can verify claims contained within the authenticated users' credentials. These claims can be compared with the set of authorization policies for the Windows Communication Foundation (WCF) service.

Depending on the claims provided by the client, the service can either grant or deny access to the operation or resources. Identity Model is useful for fine-grained authorization and is most beneficial when using issue token authentication.

- Resource based—Individual resources are secured by using Windows access control lists (ACLs).

2.5 Overview of Message Protection

Message Protection is about the process of encrypting data, maintaining confidentiality of the messages and message signing.

The following topics describe message protection in detail:

- Understanding Message Protection
- About Message Encryption
- About Message Signing (XML Signature)

2.5.1 Understanding Message Protection

Message protection encompasses two concepts, **message confidentiality** which involves keeping the data secret and **message integrity** by digitally authorizing the message.

Message confidentiality involves keeping the data secret, as well as the identities of the sending and receiving parties. Confidentiality is achieved by encrypting the content of messages and obfuscating the identities of the sending and receiving parties.The sender uses the recipient's public key to encrypt the message. Only the recipient's private key can successfully decrypt the message, ensuring that it cannot be read by third parties while in transit.The web service's base64-encoded public certificate is published in the WSDL for use by the web service client, as described in "Using the Service Identity CertificateExtensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Message integrity is achieved by having an authority digitally sign the message. Digital signatures are used to authenticate the sender of the SOAP message and to ensure the integrity of the SOAP message (that is, to ensure that the SOAP message is not altered while in transit).

When a digital signature is applied to a SOAP message, a unique hash is produced from the message, and this hash is then encrypted with the sender's private key. When the message is received, the recipient decrypts the hash using the sender's public key.

> **Note:**
>
> Generally, the recipient does not need to have the sender's public key in its keystore to validate the certificate. It is sufficient to have the root certificate in the keystore to verify the certificate chain. However, if the sender's public key is not present in the message, as in the case of the Thumbprint and SerialIssuer mechanisms, the sender's public key must be in the recipient's keystore.

This serves to authenticate the sender, because only the sender could have encrypted the hash with the private key. It also serves to ensure that the SOAP message has not been tampered with while in transit, because the recipient can compare the hash sent with the message with a hash produced on the recipient's end.

The message-protection assertion templates and predefined policies can be used to protect request and response messages by doing the following:

- Signing messages
- Encrypting messages
- Signing *and* encrypting messages
- Decrypting messages
- Verifying signatures
- Decrypting messages *and* verifying signatures

2.5.2 About Message Encryption

The XML encryption specification describes a process for encrypting data and representing the result in XML.

Specifically, XML encryption defines:

- How digital content is encrypted and decrypted.
- How the encryption key information is passed to a recipient.
- How encrypted data is identified to facilitate encryption.

An XML document may be encrypted as a whole or in part.

The following example illustrates credit card data represented in XML.

```
<PaymentInfo xmlns="http://www.example.com/payment">
  <CreditCard>
    <Name>John Smith</Name>
    <CreditCardNumber>4012 8888 8888 1881</CreditCardNumber>
    <Limit>5000</Limit>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

The following example illustrates the same XML snippet with the credit card number encrypted and represented by a cipher value.

```
<PaymentInfo xmlns='http://www.example.com/payment">
  <CreditCard>
    <Name>John Smith</Name>
    <CreditcardNumber>
      <EncryptedData xmlns="http://www..." Type="http://www...">
        <CipherData>
          <CipherValue>A23B4...5C56</CipherValue>
        </CipherData>
      </EncryptedData>
    <Limit>5000</Limit>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
```

```
    </CreditCard>
  </PaymentInfo>
```

2.5.3 About Message Signing (XML Signature)

The XML Signature specification describes signature processing rules and syntax. XML Signature binds the sender's identity (or "signing entity") to an XML document. The document is signed using the sender's private key; the signature is verified using the sender's public key.

Signing and signature verification can be done using asymmetric or symmetric keys. XML Signature also ensures non-repudiation of the signing entity, that is, it provides proof that messages have not been altered since they were signed.

A signature can apply to a whole document or just part of a document, as shown in the following example.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<!-- The signedInfo element allows us to sign any portion of a
 document -->
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www..."/>
    <SignatureMethod Algorithm="http://www..."/>
    <Reference URI="#Body">
      <DigestMethod Algorithm="http://www..."/>
      <DigestValue>o+jtqlieRtF6DrUb...X8O9M/CmySg</DigestValue>
    </Reference>
  </SignedInfo>
  <!-- Following is the result of running the algorithm over the
  document. If changes are made to the document, the SignatureValue is
  changed. The security application verifies the SignatureValue,
  extracts the X.509 cert and uses it to authenticate the user -->
  <SignatureValue>oa+ttbsvSFi...EtRD2oNC5</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <!-- Following is the public key that matches the private key
      that signs the document -->
      <RSAKeyValue>
        <Modulus>5TT/oolzTiP++Ls6GLQUM8xoFFrAlZQ...</Modulus>
        <Exponent>EQ==</Exponent>
      </RSAKeyValue>
    </KeyValue>
    <!-- Following is the certificate -->
    <X509Data>
      <X509Certificate>wDCCAXqgAwIBAgI...</X509Certificate>
    </X509Data>
  </KeyInfo>
</Signature>
```

2.6 Overview of the Roles of Keys and Certificates in Security and Authentication

Before configuring your web services, you need to determine the type of private keys and certificates required, and the names for the keys and keystores. Then you can set up your environment accordingly.

Before you can use any message protection security policies or message protection and authentication with SSL security policies, you need to set up your keystores and truststores. Note that authentication-only security policies do not require keys. For more information about authentication policies, see Understanding the OWSM Policy Framework .

The keystore contains the entities private keys and certificates associated with those private keys. A truststore contains certificates from a Certificate Authority (CA), or other entities that this entity trusts. The keystore and the truststore can be maintained together in a common store, for instance with Oracle Web Services Manager (OWSM).

For more information, refer to the following topics:

- About Private Keys and Certificates
- Understanding How Different Security Policies Use Private Keys and Certificates
- How OWSM Locates Keystore and Key Passwords for the JKS Keystore
- About Private Keys and Certificates Configuration for SSL Policies
- About Setting up Private Keys and Certificates for Message Protection Policies

2.6.1 About Private Keys and Certificates

Private keys, digital certificates, and trusted certificate authorities establish and verify server identity and trust.

SSL uses public key encryption technology for authentication. With public key encryption, a public key and a private key are generated for a server. Data encrypted with the public key can only be decrypted using the corresponding private key and data verified with a public key can only have been signed with the corresponding private key. The private key is carefully protected so that only the owner can decrypt messages that were encrypted using the public key.

The public key is embedded in a digital certificate with additional information describing the owner of the public key, such as name, street address, and e-mail address. A private key and digital certificate provide identity for the server.

The data embedded in a digital certificate is verified by a certificate authority and digitally signed with the certificate authority's digital certificate. Well-known certificate authorities include Verisign and Entrust.net. The trusted certificate authority (CA) certificate establishes trust for a certificate.

An application participating in an SSL connection is authenticated when the other party evaluates and accepts the application's digital certificate. Web browsers, servers, and other SSL-enabled applications generally accept as genuine any digital certificate that is signed by a trusted certificate authority and is otherwise valid. For example, a digital certificate can be invalidated because it has expired or the digital certificate of the certificate authority used to sign it expired. A server certificate can be invalidated if the host name in the digital certificate of the server does not match the URL specified by the client.

The different types of trusted certificates that you can use in your environment, along with the benefits and disadvantages of each, are as follows:

- **Self-signed certificates** — A self-signed certificate is a certificate that is signed by the entity creating it.

  Benefits:

  - Easy to generate because you can do it yourself, for example, using the keytool command for the JKS keystore as described in "Generating Private Keys and Creating the Java Keystore" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
  - Can be used in production as long as you use only a new certificate that you have generated.

  Disadvantages:

  - Self-signed certificates can quickly become unmanageable if you have many clients and services that need to communicate with each other. For example, if you have three clients communicating with two services, you need to generate a private key and self-signed certificate for both services, and then import the two certificates into the truststore of all three clients.

- **Demonstration Certificate Authority (CA) signed certificates**— WebLogic Server includes a set of demonstration private keys, digital certificates, and trusted certificate authorities that are for development only.

  Benefits:

  - Easy to use because they are available and configured for use in the default WebLogic Server installation in a development environment.

  Disadvantages:

  - Should never be used in a production environment. The private key of the demo certificate CA is available to all installations of WebLogic Server, therefore each installation can generate a demo CA signed certificate using the same key. As a result, you cannot trust these certificates.

- **Internal CA signed certificates** — An internal CA signed certificate is a certificate that you issue yourself using an internal CA that you can setup for your intranet. This type of certificate can be used if your services are mostly internal only.

  Benefits:

  - You have complete control over the certificate issuance process because you create the certificates yourself.You can control to whom the certificates are issued, how long the certificates remain valid, and so on. For example, if you are issuing certificates to your partners, you can issue them only to partners in good standing.

  Disadvantages:

  - You need to ensure that all clients have the internal CA root certificate imported into their truststore.

- **External CA signed certificates** — An external CA signed certificate is a certificate that has been issued by a reputable CA such as Verisign and Entrust.net. This type of certificate should be used if your services are external facing.

  Benefits:

  - In most cases, clients are already set up to trust these external CAs. Therefore, those clients do not have to modify their truststore.

  Disadvantages:

  - You do not have any control over the certificate issuance process.

2.6.2 Understanding How Different Security Policies Use Private Keys and Certificates

OWSM security policies that require the use of private keys address two aspects: message protection and authentication.

- Message protection encompasses two concepts, **message confidentiality** and **message integrity**. Message confidentiality involves keeping the data secret and is achieved by encrypting the content of messages. Message integrity ensures that a message remains unaltered during transit by having the sender digitally sign the message.
- Authentication involves verifying that the user is who they claim to be. A user's identity is verified based on the credentials presented by that user.

The predefined OWSM policies that are included with your installation support various options for message protection and authentication. These options are described in the following sections.

> **Note:**
>
> The naming convention used for OWSM policies identifies the type of options being used. For example, the policy `oracle/wss10_username_token_with_message_protection_service_policy` is a message protection service policy that uses the wss10 web services standard and requires a username_token for authentication. For more information about policy naming conventions, see "About Recommended Naming Conventions for Documents Created in WSM Repository".

For more information, refer to the following topics:

- Overview of Message Protection Policy Types
- Overview of Authentication Token Policy Types

2.6.2.1 Overview of Message Protection Policy Types

SSL, wss11, and wss10 message protection policies are supported in Oracle Web Service Manager.

- About SSL Policies
- About wss11 Policies
- About wss10 Policies

2.6.2.1.1 About SSL Policies

Policies that include the SSL option, such as `oracle/wss_saml_or_username_token_over_ssl_service_policy`, use one-way SSL for message protection.

When using policies of this type, you need to do the following:

- On the service side, set up private keys at the SSL termination point as described in "About Private Keys and Certificates Configuration for SSL Policies".
- On the client side, set up the truststore to trust the service keys.

The private key is used to protect the messages for the SSL handshake, at which time the client and service agree on a shared session key. After the SSL handshake, the private key is not used, and all traffic between the client and the service are signed and encrypted using the shared session key.

For information on how to configure SSL, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.6.2.1.2 About wss11 Policies

Policies of this type use WS-Security 1.1 for message protection.

When using wss11 policies, you need to do the following:

- On the service side, set up private keys and define as the Encryption Key Alias in the OWSM Keystore Configuration screen. For details see "Configuring the OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- On the client side, you need to configure the client-side trust by obtaining the server's certificate in one of the following ways:
  - Use the service's public certificate published in the WSDL using the Service Identity Certificate extension as described in "Using the Service Identity Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You also need to import either the server certificate itself, or the root certificate from the CA that issued the server certificate, into the client truststore. You can choose any alias name for the server certificate.
  - Import the server certificate into the client keystore using any alias you choose, and specify that alias using the `keystore.recipient.alias` property using a configuration override when you attach the policy. For this method you need to import the actual server certificate, you cannot import the CA root certificate.

For each request, the following occurs:

1. The client creates a symmetric key, encrypts this symmetric key with the service's public key as configured with Encryption Key Alias, and then encrypts and signs the whole message with the symmetric key.
2. When the service receives the message, it decrypts the encrypted key first, and then decrypts and verifies the whole message.
3. The web service then uses the same symmetric key to encrypt and sign the response that it sends back to the client.

2.6.2.1.3 About wss10 Policies

Policies of this type use WS-Security 1.0 for message protection.

When using wss10 policies, you need to do the following:

- Set up private keys on both the client and service side. On the client side, you need to set a signature key alias, and on the service side you need both an encryption key alias and signature key alias. Note that you can normally use the same key for both.
- On the client side, you need to configure the client-side trust by obtaining the server's certificate in one of the following ways:
  - Use the service's public certificate published in the WSDL using the Service Identity Certificate extension as described in "Using the Service Identity Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You also need to import either the server certificate itself, or the root certificate from the CA that issued the server certificate, into the client truststore. You can choose any alias name for the server certificate.
  - Import the server certificate into the client keystore using any alias you choose, and specify that alias using the `keystore.recipient.alias` property using a configuration override when you attach the policy. For this method you need to import the actual server certificate, you cannot import the CA root certificate.
- On the service side, you need to configure the service to trust the client, either by importing these certificates directly, or importing the CA that issued these certificates.

Similar to the wss11 option, the client creates a symmetric key, and then encrypts the symmetric key with the service's public key. The difference, however, is that it only uses this symmetric key for encrypting the message; it doesn't use it for signing the message. Instead, the client signs the request message with its own private signature key as defined by the Signature Key alias, and the service signs the response with its private signature key.

2.6.2.2 Overview of Authentication Token Policy Types

Supported tokens for authentication are username, Kerberos, X.509 Certificate, SAML sender vouches, SAML bearer, and SAML HOK tokens from STS.

For information on how to configure authentication, see "Configuring Authentication" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **Note:**
>
> In the following sections, "signature key alias" means different things in different contexts:
>
> - In SAML sender vouches policies, the signature key alias means the key used to sign the SAML assertion. This proves the authenticity of the SAML assertion, and SAML Login module will then assert the user specified in the SAML assertion.
> - In wss10 policies, the signature key alias means the key used to sign the request and response message to prevent them from being tampered over the wire.
> - In X.509 authentication policies, the signature key alias means the key used to authenticate a particular end user.

For more information, refer to the following topics:

- About the Username Token
- About the Kerberos Token
- About the X.509 Certificate Token
- About the SAML Sender Vouches Token
- About SAML Bearer and SAML HOK Tokens from an STS

2.6.2.2.1 About the Username Token

A username token carries basic authentication information such as a username and password. When a username token is used with an authentication-only policy, no private keys are used. When used in a policy that includes authentication and message protection, the keys required for message protection are required.

2.6.2.2.2 About the Kerberos Token

A Kerberos token is comprised of a binary authentication and session token. When a kerberos token is used with an authentication-only policy, no private keys are used. When used in a policy that includes authentication and message protection, the keys required for message protection are required.

2.6.2.2.3 About the X.509 Certificate Token

Request messages are signed with the end user's signature key. On the client side you need to configure a signature key alias with the end user's signature key.

2.6.2.2.4 About the SAML Sender Vouches Token

In SAML sender vouches, the client signs the SAML token with its own private signature key.

Use the SAML sender vouches token with each of the message protection options as follows:

- With SSL: SAML sender vouches requires two-way SSL. Therefore, you need to set up an SSL client-side private key, and corresponding trust certificate on the service side. If your SSL terminates before WebLogic Server, such as in the Oracle HTTP Server or in the Load balancer, you must configure these layers to propagate the client certificate all the way to WebLogic Server.
- With wss11: Normally wss11 does not need a client-side signature key. However, when you use wss11 with SAML, you must set up a signature key on the client side, and configure it using the signature key alias. You must also add this client certificate or its issuer to the service's truststore.
- With wss10: There is no additional setup to use SAML. The regular client signature key that is used for signing the request is also used for signing the SAML token.

> **Note:**
>
> Be very cautious when using the SAML signature key. It is a very powerful key as it enables the client side to impersonate any user. Consider configuring the server side to limit the number of SAML signers that is accepts, by setting up a Trusted DN list. For information about setting up a trusted DN, see "Configuring SAML Trusted Issuers and DN Lists Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.6.2.2.5 About SAML Bearer and SAML HOK Tokens from an STS

For these options, the client does not construct the SAML token. Instead it is STS that constructs and signs the SAML token.

When using tokens from an STS, you must add the STS's certificate or its issuer to the service's truststore. Optionally, you can configure the STS in the Trusted DN list.

2.6.3 How OWSM Locates Keystore and Key Passwords for the JKS Keystore

OWSM expects JKS keystore and key passwords to be in the Credential Store Framework (CSF).

The working of JKS Keystore and key passwords is shown below:

> **Note:**
>
> For information about the OPSS Keystore Service, see "Managing Keys and Certificates with the Keystore Service" in *Securing Applications with Oracle Platform Security Services*.

- A JKS keystore file is protected by a keystore password.
- A keystore file consists of zero or more private keys, and zero or more trusted certificates. Each private key has its own password, (although it is common to set the key passwords to be the same as the keystore password). OWSM needs to know both the keystore password and key password.
- The CSF consists of many *maps*, each with a distinct name. OWSM only uses the map `oracle.wsm.security`.
- Inside each map is a mapping from multiple csf-key entries to corresponding credentials. A csf-key is just a simple name, but there can be many different types of credentials. The most common type of credential is a password credential which is primarily comprised of a username and a password.

  OWSM refers to the following csf-keys for the JKS keystore inside the `oracle.wsm.security` map:

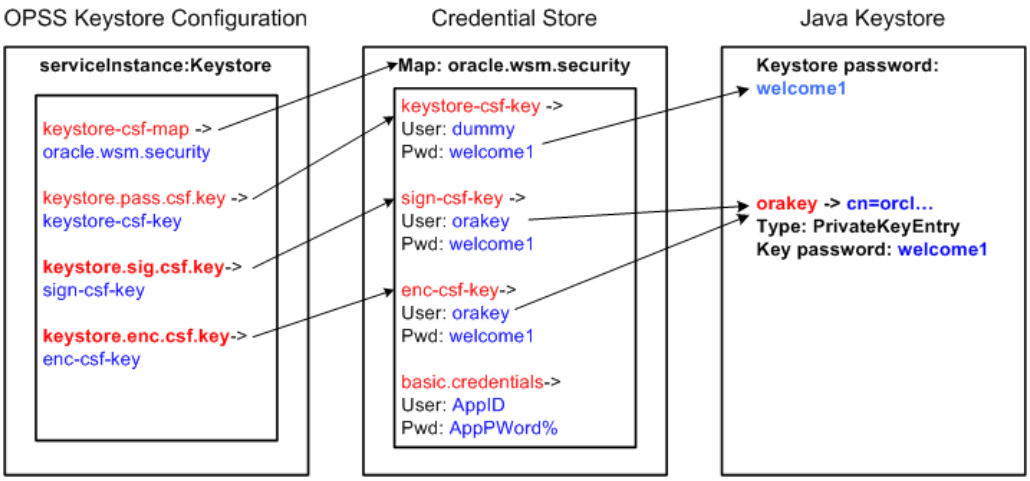
  – `keystore-csf-key` - This key should contain the keystore password. The username is ignored.

- `enc-csf-key` - This key should contain the encryption key alias as the username, and the corresponding key password.
- `sign-csf-key` - This key should contain the signature key alias as the username, and the corresponding key password.

In addition to these csf-keys, you should add a csf-key entry for every new private key that you want OWSM to use, for example when you want to specify signature and encryption keys in configuration overrides.

Figure 2-1 illustrates the relationship between the JKS keystore configuration in the OPSS, the `oracle.wsm.security` map in the credential store, and the OWSM Java keystore.

Figure 2-1 OWSM Keystore Configuration for Message Protection



As shown in the figure:

- The `keystore.csf.map` property points to the OWSM map in the credential store that contains the CSF aliases. In this case `keystore.csf.map` is defined as the recommended name `oracle.wsm.security`, but it can be any value.
- The `keystore.pass.csf.key` property points to the CSF alias `keystore-csf-key` that is mapped to the username and password of the JKS keystore. Only the password is used; username is redundant in the case of the keystore.
- The `keystore.sig.csf.key` property points to the CSF alias `sign-csf-key` that is mapped to the username and password of the private key that is used for signing in the JKS keystore.
- The `keystore.enc.csf.key` property points to the CSF alias `enc-csf-key` that is mapped to the username and password of the private key that is used for decryption in the JKS keystore.

2.6.4 About Private Keys and Certificates Configuration for SSL Policies

You can configure keys and trust on the client and service side to use SSL policies.

- **Service-side configuration:** For SSL security policies, you must setup the private keys at the SSL termination point. These termination points typically consist of one of the following:
  - Java EE container, such as WebLogic Server. For configuration details, see "Configuring Keystores for SSL" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
  - Oracle HTTP Server, if you have configured it as a Web proxy between the client and WebLogic Server. For configuration details, see "Configuring SSL on Oracle HTTP Server" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
  - Load balancer, if you have a load balancer in front of WebLogic Server or Oracle HTTP Server.

  > **Note:**
  >
  > With SSL you can only have one private key per server, so if there are multiple web services running on the same server, they all use the same private key. This SSL private key needs to be generated with the same DN as the host name, although for testing purposes, you can turn off the host name verifier on the client side.

  Sample basic configuration: Use the demonstration digital certificates, private keys, and trusted CA certificates that are included with WebLogic Server. These keys and certificates are provided for development use only and should not be used in a production environment.

  Advanced configuration: In a production environment, use an internal or external CA.
- **Client-side configuration:** On the client side, you need to import the server certificates into the client truststore. If the server side is using self-signed certificates, you need to include them directly. If the server side is using certificates that are signed using a CA, import the CA root certificate into the client truststore. Note that each type of web service client has a different client truststore:
  - For Java EE (WebLogic) web services, you need to import the keys into the WebLogic Server trust store. The demonstration CA certificate is already present in the WebLogic Server truststore.
  - For Oracle Infrastructure web services you need to specify the truststore using javax.net.ssl* system properties, or specify it in the connection. For details, see "Configuring SSL for a Web Service Client" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
  - For SOA composite applications, you need to specify the truststore using the javax.net.ssl* property as described in "Configuring SOA Composite Applications for Two-Way SSL Communication" in *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
  - For asynchronous web services, you need to configure the truststore as described in "Configuring SSL for Asynchronous Web Services" in *Developing Oracle Infrastructure Web Services*.

2.6.5 About Setting up Private Keys and Certificates for Message Protection Policies

For OWSM message protection security policies, you must setup your private keys in the OWSM keystore.

There is a single OWSM keystore per domain, and it is shared by all web services and clients running in the domain. This keystore contains both private keys and trust certificates. The JDK cacerts file is not used by OWSM.

The following sections describe a basic OWSM keystore configuration and an advanced configuration.

- Understanding Sample Basic Configuration
- About Advanced Setup Considerations

2.6.5.1 Understanding Sample Basic Configuration

The easiest way to set up the OWSM keystore is to create a single self-signed private key and use it for the entire domain. When you create the private key and keystore, you specify a name and a password for the keystore, for example `default-keystore.jks` as the keystore name if you are using a JKS keystore, and *`Password`* as the password for the keystore. You also specify an alias name and password to use when referring to the private key, for example `orakey` as the alias name and *`Password`* as the key password. You can use the same key and alias for both the signature key alias and the encryption key alias, and the same password for both the keystore and the alias. You do not need to add any trusted certificates, as certificates associated with private keys are automatically considered as trusted.

Once you have created the keys and keystore, you need to provide the keystore password, and alias names and passwords to OWSM. You can do so using either Fusion Middleware Control or WLST.

The procedures in "Generating Private Keys and Creating the Java Keystore" and "Configuring the OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* describe how to setup this basic configuration for a JKS keystore using the names and passwords specified in this example. In your own environment, you should use names and passwords that are appropriate for your configuration.

As long as your client and server are on the same domain, this set up is sufficient to work with most of the policies. That is, you can use any wss10 or wss11 policies with or without SAML.

If you have multiple related domains that share a common JPS root, you can copy this keystore file to all the domains. By doing so, all the related domains will share this single key for all encryption and signing.

2.6.5.2 About Advanced Setup Considerations

The simplest way to set up message protection security is to have a single private key for all web services in the domain.

For more sensitive web services, you need to configure each web service to use its own distinct private encryption key. These private keys need to exist in the OWSM keystore. Ensure that each one uses a different alias name, for example `ServiceA`, and `ServiceB`, and that you add the aliases to the credential store. When you attach a policy to the service, you need to use a configuration override to indicate the specific alias name that the web service requires, otherwise it will use the default alias that you configured for the domain.

The procedure in "Adding Keys and User Credentials to the Credential Store" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* describes how to add these sample aliases to the credential store.

You should also use trusted certificates issued by an internal or external CA, instead of self-signed certificates, because it is much easier to manage the trusted CA certificates. Be sure, however, to set up the SAML signers Trusted DN list, as described in "Configuring SAML Trusted Issuers and DN Lists Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. This is especially important if you import external CA certificates into the OWSM Keystore, otherwise any user with a certificate will be able to sign a SAML token and impersonate any user.

2.7 Understanding How OWSM Uses the Credential Store

The Credential Store Framework (CSF) provides a way to store, retrieve, and delete credentials for a Web Service and other applications.

OWSM uses the CSF to manage the credentials in a secure form by retrieving the following information:

- Alias names and passwords for keys in the Java keystore

  For details about how OWSM uses the credential store to look up alias names and passwords from the Java keystore, see "How OWSM Locates Keystore and Key Passwords for the JKS Keystore".

- Usernames and passwords used for authentication

  Suppose, for example, that you have a web service that accepts a username token for authentication. If you create a web service client to talk to this web service, you need to configure the web service client with a username and password that can be sent to the web service. You store this username and password in the credential store (using either Fusion Middleware Control or WLST) and assign it a csf key.

  For example, the `oracle/wss_username_token_client_policy` policy includes the `csf-key` property, with a default value of `basic.credentials`. To use the `wss_username_token_client_policy`, you should create a new password credential in the CSF using the credential name `basic.credentials`, and the username and password with which the client needs to connect. If you have two web service clients that use this same client policy, these clients can either share the same password credential, which defaults to `basic.credentials`, or each one can have its own credential. In the latter case, you need to create two password credentials in the CSF, for example `App1.credentials` and `App2.credentials`, for Client1 and Client2 respectively. For Client1, you set the csf-key configuration override to `App1.credentials`, and for Client2, you set the csf-key property to `App2.credentials`. For more information, see "Overriding Policy Configuration Properties" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. Note

that in both cases, the username and password must represent valid users in the OPSS identity store.

An OPSS CSF can store a username and password. A generic credential can store any credential object.

The OPSS CSF configuration is maintained in the `jps-config.xml` file in the *`domain-home`*`/config/fmwconfig` directory.

When you configure the OWSM keystore using Fusion Middleware Control, as described in "Configuring the OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, the aliases and passwords that you specify are securely stored in the credential store. If, however, you add other aliases to the keystore, or you need to add authentication credentials for a client, you need to ensure that they are configured and stored in the credential store.

2.8 Understanding Security Policies

WS-SecurityPolicy is part of the Web Services Secure Exchange (WS-SX) set of specifications hosted by OASIS (in addition to WS-SecurityPolicy, the WS-SX technical committee defines two other sets of specifications: WS-Trust and WS-SecureConversation, described later in this chapter).

WS-SecurityPolicy defines a set of security policy assertions used in the context of the WS-Policy framework. WS-SecurityPolicy assertions describe how messages are secured on a communication path. Oracle has contributed to the OASIS WS-SX technical committee several practical security scenarios (a subset of which is provided by OWSM 12*c*). Each security scenario describes WS-SecurityPolicy policy expressions.

WS-SecurityPolicy *scenarios* describe examples of how to set up WS-SecurityPolicy policies for several security token types described in the WS-Security specification (supporting both WS-Security 1.0 and 1.1). The subset of the WS-SecurityPolicy scenarios supported by OWSM 12*c* represents the most common customer use cases. Each scenario has been tested in multiple-vendor WS-Security environments.

To illustrate WS-SecurityPolicy, let's use a scenario supported by OWSM: UsernameToken with plain text password. As mentioned earlier, Username token is one of the security tokens specified by WS-Security. This specific scenario uses a policy that says that a requester must send a password in a Username token to a recipient who has authority to validate that token. The password is a default requirement for the WS-Security Username Token Profile 1.1.

This scenario is only recommended when confidentiality of the password is not an issue, such as a pre-production test scenario with dummy passwords.

```
<wsp:Policy>
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:UsernameToken/>
    </wsp:Policy>
  </sp:SupportingTokens>
</wsp:Policy>
```

An example of a message that conforms to the above stated policy is shown below.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="...">
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1" xmlns:wsse="...">
      <wsse:UsernameToken>
        <wsse:Username>Smith</wsse:Username>
        <wsse:Password Type="http://docs.oasis open.org...>
           Password
        </wsse:Password>
        <wsse:Nonce EncodingType="...#Base64Binary">qB...</wsse:Nonce>
        <wsu:Created>2008-01-02T00:01:03Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <Oracle xmlns=http://xmlsoap.org/Oracle>
      <text>EchoString</text>
    </Oracle>
  </soap:Body>
</soap:Envelope>
```

The example above contains a <Nonce> element and a <Created> timestamp, which, while optional, are recommended to improve security of requests against replay and other attacks. A nonce is a randomly generated (unique) number. The timestamp can be used to define the amount of time the security token is valid.

2.9 Overview of Security Tokens

Web Services Security (WS-Security) specifies SOAP security extensions that provide confidentiality using XML Encryption and data integrity using XML Signature.

WS-Security also includes profiles that specify how to insert different types of binary and XML security tokens in WS-Security headers for authentication and authorization purposes.

- Understanding Security Tokens
- About the Username Token
- About the X.509 Certificate
- About the Kerberos Token
- About the SAML Token

2.9.1 Understanding Security Tokens

Different kinds of Web services security tokens are explained in the following section.

Web services security supports the following security tokens:

- Username—defines how a web service consumer can supply a username as a credential for authentication). For more information, see About the Username Token.
- X.509 certificate—a signed data structure designed to send a public key to a receiving party. For more information, see About the X.509 Certificate.
- Kerberos ticket—a binary authentication and session token. For more information, see About the Kerberos Token.

- Security Assertion Markup Language (SAML) assertion—shares security information over the Internet through XML documents. For more information, see About the SAML Token.

2.9.2 About the Username Token

The username token carries basic authentication information.

The `username-token` element propagates username and password information to authenticate the message.

2.9.3 About the X.509 Certificate

An X.509 digital certificate is a signed data structure designed to send a public key to a receiving party. A certificate includes standard fields such as certificate ID, issuer's Distinguished Name (DN), validity period, owner's DN, owner's public key, and so on.

Certificates are issued by certificate authorities (CA). A CA verifies an entity's identity and grants a certificate, signing it with the CA's private key. The CA publishes its own certificate which includes its public key.

Each network entity has a list of the certificates of the CAs it trusts. Before communicating with another entity, a given entity uses this list to verify that the signature of the other entity's certificate is from a trusted CA.

2.9.4 About the Kerberos Token

Kerberos token is a cross-platform authentication and single sign-on system. The Kerberos protocol provides mutual authentication between two entities relying on a shared secret (symmetric keys).

Kerberos uses the following terminology:

- A Principal is an identity for a user (i.e., a user is assigned a principal), or an identity for an application offering Kerberos services.
- A Realm is a Kerberos server environment; a Kerberos realm can be a domain name such as EXAMPLE.COM (by convention expressed in uppercase).

Kerberos involves a client, a server, and a trusted party to mediate between them called the Key Distribution Center (KDC). Each Kerberos realm has at least one KDC. KDCs come in different packages based on the operating platform used (for example, on Microsoft Windows, the KDC is a domain service). The Kerberos Token profile of WS-Security allows business partners to use Kerberos tokens in service-oriented architectures.

2.9.5 About the SAML Token

The Security Assertion Markup Language (SAML) is an open framework for sharing security information over the Internet through XML documents.

SAML was designed to address the following:

- Limitations of web browser cookies to a single domain: SAML provides a standard way to transfer cookies across multiple Internet domains.

- Proprietary web single sign-on (SSO): SAML provides a standard way to implement SSO within a single domain or across multiple domains. This functionality is provided by the Oracle Identity Federation product.
- Federation: SAML facilitates identity management (e.g., account linking when a single user is known to multiple web sites under different identities), also supported by Oracle Identity Federation.
- Web Services Security: SAML provides a standard security token (a SAML assertion) that can be used with standard web services security frameworks (e.g., WS-Security) – This is the use of SAML that is particularly relevant to web services security, fully supported by OWSM.
- Identity propagation: SAML provides a standard way to represent a security token that can be passed across the multiple steps of a business process or transaction, from browser to portal to networks of web services, also a feature supported by OWSM.

The SAML framework includes 4 parts:

- Assertions: How you define authentication and authorization information.
- Protocols: How you ask (SAML Request) and get (SAML Response) the assertions you need.
- Bindings: How SAML Protocols ride on industry-standard transport (e.g., HTTP) and messaging frameworks (e.g., SOAP).
- Profiles: How SAML Protocols and Bindings combine to support specific use cases.

In the context of WS-Security, only SAML assertions are used. The protocols and bindings are provided by the WS-Security framework. SAML is widely adopted by the industry, both for browser-based federation and federation enabled by web services flows.

SAML assertions are very popular security tokens within WS-Security because they are very expressive and can help prevent man-in-the-middle and replay attacks.

Typically, a SAML assertion makes statements about a principal (a user or an application). All SAML assertions include the following common information:

- Issuer ID and issuance timestamp
- Assertion ID
- Subject
- Name
- Optional subject confirmation (for example, a public key)
- Optional conditions (under which an assertion is valid)
- Optional advice (on how an assertion was made)

SAML assertions can include three types of statements:

- Authentication statement: issued by an authentication authority upon successful authentication of a subject. It asserts that Subject S was authenticated by Means M at Time T.
- Attribute statement: issued by an attribute authority, based on policies. It asserts that Subject S is associated with Attributes A, B, etc. with values a, b, and so on.

- Authorization decision statement (deprecated in SAML 2.0, now supported by XACML): issued by an authorization authority which decides whether to grant the request by Subject S, for Action A (e.g., read, write, etc.), to Resource R (e.g., a file, an application, a web service), given Evidence E.

SAML assertions can be embedded (i.e., a SAML assertion can contain another SAML assertion). SAML assertions can be signed (using XML Signature) and/or encrypted (using XML Encryption).

2.10 Understanding Secure Attachments

OWSM policies support two mechanisms to secure attachments: Packaging SOAP messages with attachments (SwA), and Message Transmission Optimization Mechanism (MTOM).

Packaging SOAP messages with attachments (SwA) has become common for any data that cannot be placed inside SOAP Envelope. The primary SOAP message can reference additional entities as attachments or attachments with MIME headers. For more information, see "Securing SwA Attachments" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Using MTOM, binary content can be sent as a MIME attachment, which reduces the transmission size on the wire. The binary content is semantically part of the XML document. Attaching an MTOM policy ensures that the message is converted to a MIME attachment before it is sent to the web service or client. See "MTOM Attachment Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for more information.

2.11 Overview of Secure Conversation

OWSM implements the Web Services Trust (WS-Trust 1.3) and Web Services Secure Conversation (WS-SecureConversation 1.3) specifications, which together provide secure communication between web services and their clients.

Secure conversation is described in the following topics:

- About Secure Conversation
- Overview of WS-SecureConversation Usage
- WS-SecureConversation Architecture
- When to Use WS-SecureConversation
- When To Use Re-Authentication
- About Setting the Bootstrap Mode
- Overview of Persistence

2.11.1 About Secure Conversation

The Web Services Secure Conversation Language (WS-SecureConversation) specification defines mechanisms for establishing and sharing security contexts or any credentials, and deriving keys from established security contexts (or any shared secret).

The Web Services Secure Conversation Language (WS-SecureConversation) specification (`http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc`) defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 and Web Services Trust Language (WS-Trust) to provide secure communication across one or more messages.

OWSM includes policies for which WS-SecureConversation is enabled by default, as described in "Which Policies Support WS-SecureConversation?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You may find that using the preconfigured WS-SecureConversation policies makes your security tasks easier to view at a glance and manage.

In addition, the OWSM security policies described in "Which Policies Support WS-SecureConversation?" include a configuration setting that allows you to enable and configure WS-SecureConversation for that policy.

2.11.2 Overview of WS-SecureConversation Usage

WS-SecureConversation is used in specific scenarios.The benefits of using WS-SecureConversation is explained in the following section.

The following topics describe WS-SecureConversation and its combination with WS-ReliableMessaging.

- When to Use WS-Secure Conversation
- Benefits of WS-SecureConversation
- About WS-SecureConversation With WS-ReliableMessaging

2.11.2.1 When to Use WS-Secure Conversation

There are two primary reasons you might want to use WS-SecureConversation: performance and security.

WS-Security, the standard employed by the OWSM security policies, provides the basic mechanism for securing messages.

However, without WS-SecureConversation, a client using an OWSM security policy (for example, `oracle/wss11_username_with_message_protection`) that exchanges multiple messages must repeatedly authenticate itself and perform expensive asymmetric operations such as key exchanges in each request.

To securely exchange multiple messages, a client and a web service typically require a security context in which to exchange the messages. WS-SecureConversation provides just such a context. It adds a "handshake" process, which allows a web service and its client to authenticate to each other and to establish a shared security context. The security context is shared by the client and web service for the lifetime of a communication session. This context contains a shared secret key that can be used to secure subsequent messages between the client and service, and can improve performance by avoiding repeated key exchanges in multi-message exchange scenarios.

Enabling secure conversation means that there is no need to repeatedly exchange the keys and authenticate each time.

Consider the following sequence:

1. When the first request is made by the client, the handshake happens between the client and the web service.
2. The client authenticates itself to the service as defined in the bootstrap policy using the WS-Trust protocol.
3. The web service returns the secure context token (SCT) containing the binary secret that is used for subsequent requests to secure messages during the communication session.

> **Note:**
>
> The authentication mechanism required by the web service does not change, merely the frequency with which the authentication operation is performed.

2.11.2.2 Benefits of WS-SecureConversation

For more information on the benefits of using WS-SecureConversation, see the comparison of Wss11, Wss10, or SSL OWSM policies.

WS-SecureConversation provides different benefits depending on whether you are using Wss11, Wss10, or SSL OWSM policies, as follows:

- Wss11 — Wss11 scenarios involve one or two asymmetric cryptographic operations in the request depending on the policy. When WS-SecureConversation is in use, the authentication and asymmetric cryptographic operations are done only once at bootstrap time and the subsequent application requests will use the SCT to secure the messages. The SCT uses only symmetric cryptographic operations, which are less expensive.

  For example, when a "username with message protection" policy is enabled with WS-SecureConversation, the bootstrap policy uses the username token for authentication and Wss11 for message protection. However, subsequent messages do not involve any authentication and the messages are protected by the SCT.
- Wss10 — Wss10 scenarios involve four asymmetric cryptographic operations in the request and response messages. When WS-SecureConversation is in use, the authentication and asymmetric cryptographic operations are done only once at bootstrap time and the subsequent application requests use the SCT to secure the messages. The SCT uses only symmetric cryptographic operations, which are less expensive.

  For example, when a "username with message protection" policy is enabled with WS-SecureConversation, the bootstrap policy uses the username token for authentication and Wss10 for message protection. However, subsequent messages do not involve any authentication and the messages are protected by the SCT.
- SSL — With SSL scenarios, communication happens via SSL throughout the session. At the bootstrap time, authentication happens. Subsequent requests use the SCT to sign the timestamp and no authentication token is sent. Signing the timestamp by SCT proves that the request is sent by the authenticated client.

  In these scenarios, SSL is used for message protection; signing the timestamp with the SCT is used for authentication.

For example if WS-SecureConversation is enabled for a "username over SSL" policy, the bootstrap policy uses the username token for authentication and SSL for message protection. However, subsequent messages will also use SSL but will contain a timestamp signed by the SCT instead of the username token.

2.11.2.3 About WS-SecureConversation With WS-ReliableMessaging

A particularly important use of WS-SecureConversation is to provide security for WS-ReliableMessaging (WS-RM) policies. WS-RM benefits from the use of secure conversation to prevent sequence attacks.

As explained in the WS-ReliableMessaging specification (http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html), because reliable messaging sequences are expected to exchange a number of messages, it is recommended that a security context be established by using the WS-Trust and WS-SecureConversation mechanisms for protecting sequences.

Therefore, you should attach a WS-SecureConversation-enabled security policy with your WS-RM policy.

2.11.3 WS-SecureConversation Architecture

The WS-SecureConversation specification defines extensions that build on Web Services Security (WS-Security) and Web Services Trust Language (WS-Trust).

The specification (http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc) is the best source of information about the WS-SecureConversation architecture, features, and functions.

At a minimum, you should be familiar with the following concepts:

- WS-Trust — As described in Web Services Trust Language (WS-Trust) in *Understanding WebLogic Web Services for Oracle WebLogic Server*, the Web Services Trust Language (WS-Trust) specification defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 to provide a framework for requesting and issuing security tokens, and to broker trust relationships.
- Security Context — A security context is an abstract concept that refers to an established authentication state and negotiated key(s) that may have additional security-related properties.
- Security Context Token — A security context token (SCT) is a representation of the security context abstract concept, which allows a context to be named by a URI and used with WS-Security. Once the context and secret have been established (authenticated), you can then compute derived keys for each key usage in the secure context.
- Derived Keys — As described in the WS-SecureConversation specification (http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc), "A security context token implies or contains a shared secret. This secret MAY be used for signing and/or encrypting messages, but it is recommended that derived keys be used for signing and encrypting messages associated only with the security context."

  Again as described in the WS-SecureConversation specification, "Once the context and secret have been established (authenticated), the mechanisms

described in Derived Keys can be used to compute derived keys for each key usage in the secure context."

Derived keys are useful for message protection. Instead of using the same SCT across multiple requests, a different key derived from the SCT is used in each request, which improves overall security.

When you enable WS-SecureConversation for a policy, OWSM uses derived keys by default for WSS10 and WSS11. (For SSL policies, message protection is done using SSL and a derived key is not necessary.)

- Session management — OWSM maintains the client and server secure conversation session information based on a computed Session ID.

  On the Web server side, the Session ID is maintained based on the port used by the web service.

  Client sessions are expressed by the term "reference," which is similar in concept to a client port/binding that enables message communication, or to a SOA reference.

  In the WS-SecureConversation implementation, each client reference is a separate WS-SecureConversation session. From the perspective of a web service client request, this leads to the following outcomes:

  – Multiple requests can belong to the same reference.

  – All the requests with the same Session ID belong to the same session.

  – The state for which the Session ID is valid depends on the re-authentication setting.

  OWSM computes the Session ID at runtime for each message, and associates one or more requests to a session. OWSM uses user credentials, service information, and policy and configuration data to compute the Session ID.

  The Session ID is especially important when used with Oracle WS-RM policies, where for security and performance reasons multiple messages in an RM session are protected by the same secure conversation session.

- Inner and outer policies — In the OWSM implementation of WS-SecureConversation, a secure conversation policy has actually two policies: inner and outer. The bootstrap (inner) policy is used to obtain the token and establish the handshake between the client and the web service. The outer policy is used for application messages when making requests with the token.

  The message security settings for the outer policy are obtained from the original OWSM WS-Security policy, such as such as wss11_username_with_message_protection. The message security settings for the inner policy are then derived from the outer policy.

  In most cases, you do not need to be concerned with the details of the inner and outer policies, as OWSM handles this on your behalf. However, the OWSM WS-SecureConversation implementation provides an advanced setting that provides additional control, as described in "About Setting the Bootstrap Mode".

- Re-authentication — OWSM includes a re-authenticate control that indicates whether to create a separate session for each user or to allow users to share the same session. A user is authenticated only once whether re-authenticate is true or not.

  There is one supported use case in which the user ID might be different for each application message and therefore needs to be authenticated in each message

during the WS-SecureConversation session: ID propagation with SAML sender vouches.

Re-authentication allows multiple users to share a session. In this case, the authentication token is sent in each request because multiple users share the session. However, there is no need to exchange keys and asymmetric operations (sign, encrypt) are not performed in subsequent requests.

The state for which the Session ID is valid depends on the re-authentication setting:

– If re-authenticate is false, on the client side the Session ID is maintained for a single reference for a given user.

  On the server side, the Session ID is maintained based on the port used by the web service.

– If re-authenticate is true, on the client side the Session ID is maintained for a single reference, which may involve multiple users.

  On the server side, the Session ID is maintained based on the port used by the web service.

2.11.4 When to Use WS-SecureConversation

WS-SecureConversation is used in specific scenarios in OWSM.

You should consider using WS-SecureConversation in the following scenarios:

- You are using any OWSM WS-RM policy.
- Your web service client is protected with an OWSM security policy (for example, `oracle/wss11_username_with_message_protection`) and frequently exchanges multiple messages.

When a web service client or service are secured by OWSM and expect to be involved in multiple message exchanges, it makes sense to enable WS-SecureConversation. Enabling WS-SecureConversation provides better performance because the SCT secures subsequent messages between the client and service and you do not incur the overhead of repeated authentication and public key crypto operations.

> **Note:**
>
> For ID propagation use cases, WS-SecureConversation provides a performance benefit mainly for message protection because the authentication token is sent in each message during the session at the expense of performance.

Consider the following scenarios in which multiple message exchanges might happen and WS-SecureConversation might be useful:

- One to One — In this case, a client application invokes a particular web service multiple times on behalf of a single user.
- One to One with re-authenticate=true (identity propagation) — In this case, a client application invokes a particular web service multiple times. However, a different identity may need to be passed to the web service in each subsequent request.

A single secure conversation session is created for all users.

2.11.5 When To Use Re-Authentication

You can enable the re-authenticate control only in the case of ID propagation with SAML sender vouches policies.

You should use re-authentication when the user ID might be different for each application message. In this situation, the user is authenticated in each message.

The bootstrap is done using the client identity and the end user identity is passed in all application requests to the service. WS-SecureConversation provides a benefit mainly for message protection, because the authentication token is sent in each message during the session at the expense of performance.

By default, the re-authenticate control is not set with WS-SecureConversation, and you can enable it only when WS-SecureConversation is also enabled.

2.11.6 About Setting the Bootstrap Mode

In the OWSM implementation of WS-SecureConversation, a secure conversation policy has actually two policies: the boostrap (inner) policy, and outer policy.

The bootstrap (inner) policy is used to obtain the token and establish the handshake between the client and the web service. The outer policy is used for application messages when making requests with the token.

The message security settings for the outer policy are obtained from the original OWSM WS-Security policy, such as such as `oracle/wss11_username_with_message_protection`. The message security settings for the inner policy are then derived from the outer policy.

Therefore in most cases, you do not need to be concerned with the details of the inner and outer policies, as OWSM handles this on your behalf. However, the OWSM WS-SecureConversation implementation provides additional control.

The following Bootstrap Message Security options are available:

- Inherit From Application Setting
- Use Independent Setting:
  - Algorithm Suite
  - Include Timestamp
  - Confirm Signature
  - Encrypt Signature

2.11.7 Overview of Persistence

Two persistence implementations exist: default domain-wide persistence, and client-specific/web service-specific persistence.

The section explains the preceding persistence implementation as follows:

- About Default Domain-Wide Persistence Implementation
- About Client- and Web Service-Specific Persistence Implementation

2.11.7.1 About Default Domain-Wide Persistence Implementation

OWSM includes a default domain-wide persistence implementation that supports the Coherence Cluster and in-memory persistence providers.

The Coherence persistence provider is the default when running in WebLogic Server, for both the web service client and web service. Otherwise, the in-memory persistence provider is the default.

The level of persistence granularity for session recovery is the session object.

This persistence implementation is enabled by default and does not require any configuration.

2.11.7.2 About Client- and Web Service-Specific Persistence Implementation

Each client and web service can specify one or more (one per port) persistence providers, which can be either the Coherence provider or the in-memory provider.

To do this, use one of the mechanisms described in "Configure Persistence".

2.12 Overview of the Kerberos Protocol

Kerberos is an authentication protocol that enables computers (clients and servers) communicating over a non-secure network to prove their identity to one another in a secure manner, with the help of a trusted third party.

The following topics explain Kerberos protocol in detail:

- Understanding the Kerberos Protocol
- Understanding Credential Delegation in Kerberos
- Understanding Kerberos and SPNEGO
- About Kerberos and WS-SecureConversation Derived Keys

2.12.1 Understanding the Kerberos Protocol

In Kerberos, this trusted third party is the Key Distribution Center (KDC), which contains key information for clients and servers, called principals.

The KDC consists of two components:

- The Authentication Service (AS), which authenticates a principal with the KDC
- The Ticket Granting Service (TGS), which provides authenticated principals with tickets they can use to request services from other principals in the KDC.

OWSM supports MIT Kerberos and Microsoft Active Directory as the KDC. For information about using MIT Kerberos, see "Using MIT Kerberos" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. For information about using Microsoft Active Directory, see "Using Microsoft Active Directory with Key Distribution Center" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Here are the high-level steps involved when Kerberos is used for message security between a client principal and a server principal:

1. AS-REQ (request to Authentication Service): The client begins the authentication process by sending the user ID to the AS.
2. AS-REP (reply from Authentication Service): The AS responds with:
   - A client/TGS session key, encrypted using a hash of the user's password from the KDC
   - A Ticket Granting Ticket (TGT), encrypted using the secret key of the TGS.
3. TGS-REQ (request to Ticket Granting Service): To begin communicating with services, the client first sends the following to the TGS:
   - The TGT it received from the AS
   - The ID of the requested service
   - An authenticator, encrypted using the client/TGS session key from the AS
4. TGS-REP (reply from Ticket Granting Service): The TGS decrypts the TGT using its secret key, extracts the client/TGS session key from the decrypted TGT, and then uses this session key to decrypt the authenticator. It then responds with:
   - A client/server session key, encrypted using the client/TGS session key
   - A Service Ticket (ST), encrypted using the service's secret key
5. AP-REQ (request to application): After receiving the reply from the TGS, the client initiates contact with the service by sending it:
   - The ST it received from the TGS
   - A new authenticator, encrypted using the client/server session key from the TGS
6. AP-REP (reply from application): The service decrypts the ST using its secret key, extracts the client/server session key, and then uses this session key to decrypt the authenticator. It then extracts the timestamp from the decrypted authenticator, adds one to it, and sends this value back to the client after encrypting it using the client/server session key.
7. The client decrypts the confirmation and checks whether the timestamp is correctly updated. If so, the client can trust the server and can begin issuing service requests.

For information on configuring OWSM to support the Kerberos protocol, see "Configuring the Kerberos Login Module" and "Configuring Kerberos Tokens" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.12.2 Understanding Credential Delegation in Kerberos

Kerberos uses the credential delegation mechanism when a service needs to access another service or server in order to complete a client request. To establish such a connection, Kerberos requires the first service to be authenticated to the second service or server using the client's user account and authority level.

A common way to provide credential delegation in Kerberos is through the use of the FORWARDABLE and FORWARDED flags in Kerberos tickets, a technique called forwarded TGT. Here are a high-level steps involved in using forwarded TGT:

1. The user requests the KDC for a TGT with forwardable flag set (Forwardable TGT) by setting the KDC option named FORWARDABLE in the initial AS-REQ.

2. The client requests a FORWARDED ticket by presenting this forwardable TGT to the TGS. The client also sets the KDC Option named FORWARDED in the request (TGS_REQ) in addition to providing a set of service addresses for the new ticket.
3. More such tickets (with the FORWARDED flag set) can be obtained from the KDC by providing the FORWARDED ticket obtained in Step 2.

Here are more detailed steps that specify the message sequence:

1. The user authenticates to the KDC by sending a KRB_AS_REQ message and requests a forwardable TGT.
2. The KDC returns a forwardable TGT in the KRB_AS_REP message.
3. The user requests a forwarded TGT based on the forwardable TGT from Step 2. This is done by the KRB_TGS_REQ message.
4. The KDC returns a forwarded TGT for the user in the KRB_TGS_REP message.
5. 5. The user makes a request for a service ticket to Service 1 using the TGT returned in Step 2. This is done by the KRB_TGS_REQ message.
6. The ticket granting service (TGS) returns the service ticket in a KRB_TGS_REP message.
7. The user makes a request to Service 1 by sending a KRB_AP_REQ message, presenting the service ticket, the forwarded TGT, and the session key for the forwarded TGT.
8. To fulfill the user's request, Service 1 needs to invoke Service 2 to perform some action on behalf of the user. Service 1 uses the forwarded TGT of the user and sends that in the KRB_TGS_REQ message to the KDC, asking for a ticket to Service 2 in the name of the user.
9. The KDC returns a ticket for Service 2 to Service 1, in a KRB_TGS_REP message, along with a session key that Service 1 can use. The ticket identifies the client as the user, and not Service 1.
10. Service 1 makes a request to Service 2 using a KRB_AP_REQ, acting as the user.
11. Service 2 performs the task and responds.
12. Service 1 responds to the user's request with the response it gets from Service 2.

For information on configuring OWSM to use credential delegation, see "Configuring Credential Delegation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.12.3 Understanding Kerberos and SPNEGO

SPNEGO (Simple and Protected GSS-API Negotiation Mechanism) is a standard that enables a client and a service to negotiate a method to use for authentication. Because SPNEGO uses HTTP headers to perform the negotiation, it is especially useful in a cross-platform context such as the web, where SOAP and REST endpoints that use HTTP are common.

When Kerberos is used in SPNEGO negotiation, the Kerberos token is wrapped in the HTTP header under the auth-scheme Negotiate. The WWW-Authenticate and Authorization headers are used to communicate the SPNEGO token between the client and the service, as follows:

1. The client requests access to a protected service on the server without any Authorization header.
2. Since there is no Authorization header in the request, server responds with the status code 401 (Unauthorized) and the WWW-Authenticate header set to Negotiate.
3. The client uses the user credentials to obtain the Kerberos token and then sends it to the server in the Authorization header of the new request. For example, Authorization: Negotiate a87421000000492aa874209....
4. The server decodes the token in the Authorization header. If the context is not complete (as in the case of Mutual Authentication), the server responds with a 401 status code and a WWW-Authenticate header containing the decoded data. For example, WWW-Authenticate: Negotiate 74900a2a....
5. The client decodes this data and sends new data back to the server. This cycle continues until the security context is established.

For information on configuring OWSM to use Kerberos with SPNEGO, see "Configuring Kerberos With SPNEGO Negotiation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.12.4 About Kerberos and WS-SecureConversation Derived Keys

The Web Services Secure Conversation (WS-SecureConversation) specification includes a feature called **derived keys**, which enables parties that have already authenticated to each other to use a common secret to derive additional keys for various uses, such as signing and encrypting messages.

Moreover, the WS-SecureConversation specification defines two types of derived keys:

- Explicit derived keys, which use the `wsc:DerivedKeyToken` element to contain the token information. The `ds:KeyInfo` element then contains a reference to this information.
- Implicit derived keys, which include the token information directly in the `ds:KeyInfo` element.

When using Kerberos in a WS-SecureConversation context, you can configure OWSM to use derived keys by enabling the Use Derived Keys option in the OWSM assertions for Kerberos.

2.13 Understanding Web Services Addressing

The Web Services Addressing (WS-Addressing) specification provides transport-neutral mechanisms to address web services and messages.

In particular, the specification(`http://www.w3.org/TR/ws-addr-core/`) defines a number of XML elements used to identify web service endpoints and to secure end-to-end endpoint identification in messages.

SOAP does not provide a standard way to specify where a message is going or how responses or faults are returned. WS-Addressing provides an XML framework for identifying web services endpoints and for securing end-to-end endpoint identification in messages.

A web service endpoint is a resource (such as an application or a processor) to which web services messages are sent.

The following is an example using WS-Addressing (`wsa` is the namespace for WSAddressing):

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <S:Header>
      <wsa:MessageID>http://example.com/xyz-abcd-123</wsa:MessageID>
      <wsa:ReplyTo>
         <wsa:Address>http://example.myClient1</wsa:Address>
      </wsa:ReplyTo>
```

WS-Addressing is transport-independent; that is, the request may be over JMS and the response over HTTP. WS-Addressing is used with other WS-* specifications, such as WS-Policy.

2.14 Understanding Web Services Trust

The WS-Trust 1.3 specification defines extensions to WS-Security that provide a framework for requesting and issuing security tokens, and to broker trust relationships. WS-Trust extensions provide methods for issuing, renewing, and validating security tokens.

To secure communication between a web service client and a web service, the two parties must exchange security credentials. As defined in the WS-Trust specification (`http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html`) , these credentials can be obtained from a SecurityTokenService (STS), which acts as trust broker.

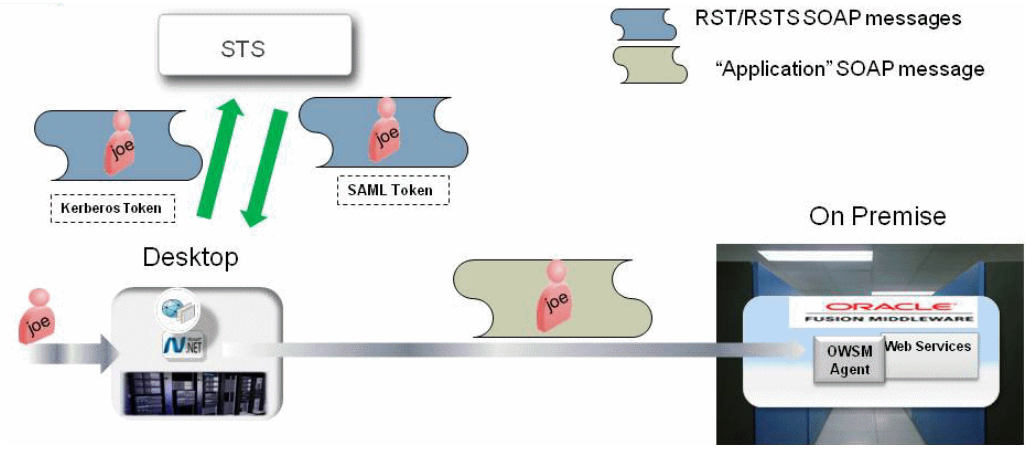
There are multiple scenarios in which you might consider using an STS, including:

- Token Exchange/Conversion — Assume that you need to exchange one kind of token for another type of token. For example, if the client has a Kerberos token but the web service requires a SAML token. You can use the STS to exchange the Kerberos token for a SAML token.
- Federation — Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With a federated identity, the individual can log in at one service provider site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity.

  For example, you might use the STS to map a client user name to the user name expected by the web service.
- Centralized Trust — The STS is trusted by both the web service client and the web service. You use this trust to provide interoperable security tokens.

Consider the token exchange scenario shown in Figure 2-2. In this scenario, a customer has a desktop application (for example, a .NET web service) that is talking to a backend web service that can accept a SAML token.

Figure 2-2 STS Token Exchange



In Figure 2-2 user "joe" logs into his desktop and a Kerberos ticket is created. When the user opens the desktop application and performs an operation, this results in a backend web service call and we want to propagate the identity of "joe" to the backend application. However the token we have is a Kerberos token on the client side and the backend Web Service only accepts a SAML token. You can use an STS to do a token conversion or token exchange.

For information on configuring OWSM to support the Kerberos protocol, see "Configuring WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.15 Understanding Web Services ReliableMessaging

WS-ReliableMessaging makes message exchanges reliable. It ensures that messages are delivered reliably between distributed applications regardless of software component, system, or network failures. Ordered delivery is assured and automatic retransmission of failed messages does not have to be coded by each client application.

Consider using reliable messaging if your web service is experiencing the following problems:

- network failures or dropped connections
- messages are lost in transit
- messages are arriving at their destination out of order

WS-ReliableMessaging considers the source and destination of a message to be independent of the client/server model. That is, the client and the server can each act simultaneously as both a message source and destination on the communications path.

For information on WS-ReliableMessaging (WS-RM), see "Using Web Services Reliable Messaging" in *Developing Oracle Infrastructure Web Services*.

2.16 Overview of Fine-Grained Authorization Using Oracle Entitlements Server

Oracle Entitlements Server (OES) is a fine-grained authorization service you can use to secure applications and services across the enterprise. It supports centralized definition of complex application entitlements and the distributed runtime enforcement of those entitlements. OES allows you to externalize entitlements and thereby remove security decisions from the application.

OWSM OES integration supports advanced authorization use cases using OES and provides the following capabilities:

- You can apply OES authorization to your SOAP-based web services. The OES authorization policy provides a grant or deny for a subject to perform a certain action on a given resource.
- OES can make grant/deny decisions based on context attributes. The context attributes could be based on information from the SOAP request message extracted using XPath statements, or they could be based on HTTP headers.
- Data masking. OWSM OES can mask (with character of your choice) certain information in the response for the web service request.

This section describes how Oracle Entitlements Server (OES) is integrated with OWSM, and how you can use OES together with OWSM for fine-grained authorization.

- References for OES Reading
- Overview of OES Integration
- About OWSM OES Policies
- Overview of Resource Mapping and Naming
- How Attributes Are Processed
- About the Guard Element

See “Configuring Fine-Grained Authorization Using Oracle Entitlements Server" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for configuration information.

2.16.1 References for OES Reading

References to many OES concepts and features is described in this section. The focus of the section is the integration with OWSM, and it does not attempt to provide an in-depth discussion of the OES concepts.

If you are not already familiar with OES, you should first refer to the Administrator's Guide for Oracle Entitlements Server and Fine Grained Authorization: Technical Insights for using Oracle Entitlements Server.

> **Note:**
>
> OWSM supports version 11.1.2.2.0 or later of OES.

2.16.2 Overview of OES Integration

You can integrate OWSM and OES. The OWSM agent checks the authorization for a protected web service based on the policies defined in OES and passes OES the autheticated subject and other attributes.

When you integrate OWSM and OES, you:

- Attach an OWSM authentication policy to your web service.
- Attach the OWSM `oracle/binding_oes_authorization_policy` or `oracle/component_oes_authorization_policy` policy, alone or in combination with the `oracle/binding_oes_masking_policy` policy as described in this section.
- Use the OES console to create authorization and data masking policies, typically with separate policies for Obligations.

> **Note:**
>
> OWSM does not expose any OES-related configuration; you use the OES console for this purpose.

For more information, refer to the following topics:

- OES Integration: The Big Picture
- Data Masking
- About XACML Obligations
- Overview of OES Fine- and Coarse-Grained Authorization

2.16.2.1 OES Integration: The Big Picture

The OWSM agent checks the authorization of a soap request for a protected web service based on the policies defined in OES. To do this, OWSM passes to OES the authenticated subject, the target resource and requested action, as well as a set of implicit attributes that are always passed in authorization requests.

In your OES policy you can define additional required values based on context attributes from the SOAP request, HTTP headers, message context properties or identity information like the subject, roles, and groups. If you configure OES to require any extra of these context attributes to make a permit/deny decision, OWSM passes them as well.

Specifically, there are two ways to contact OES for the authorization decision: a two-step method and a single-step method. You select which via the `use.single.step` attribute in `oracle/binding_oes_authorization_policy` and `oracle/component_oes_authorization_policy`.

The methods function as follows:

- In the two-step process, you must have previously identified attributes required for fine-grained authorization in the OES console and you now want OWSM to use them.

  OWSM first calls to OES to find out what attributes are needed, gets the attributes from the request payload, and then calls OES a second time to perform the actual authorization using the OES authorization policy. This means that you actually define two OES policies: one to get the needed attributes, and one for the authorization itself.

  You can also use always-passed implicit attributes, plus OES predefined attributes such as time, date, and so forth.

  This method is used for **fine-grained authorization**, as described in Overview of OES Fine- and Coarse-Grained Authorization.

- In the single-step process, OWSM makes only one call to OES to perform the authorization using the OES authorization policy. The single-step process does not require any previously-identified attributes. As with the two-step process, you can also use the always-passed implicit attributes, plus OES predefined attributes such as time, date, and so forth.

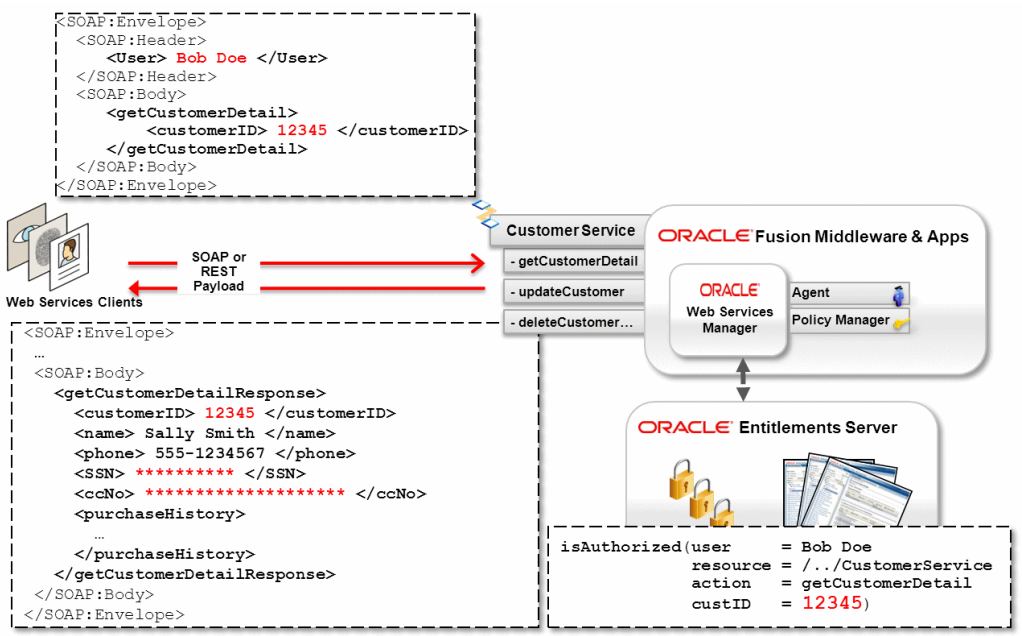
  This method can be used for **coarse-grained authorization**, as described in Overview of OES Fine- and Coarse-Grained Authorization.

2.16.2.2 Data Masking

OWSM with OES integration can mask (with asterisks) certain information in the response from the web service, without changing any of the web services code.

Assume you want to ensure that sensitive data is not passed over the wire in response to a web service client request. You use the OES console together with the `oracle/binding_oes_masking_policy` policy to replace any sensitive data leaving the web service such as a social security number or financial information with asterisks, as shown in Figure 2-3.

Figure 2-3 Masking Sensitive Data



Masking sensitive data is based on who asked for it, and on other context attributes present in the request.

Consider the following code flow for the web service response shown in Figure 2-3.

Understanding Data Masking Code Flow

1. The web service client sends a request.
2. On the inbound request, OWSM enforces the request policy and performs the appropriate authentication and authorization for user `Bob Doe`.
3. If the request is permitted, OWSM passes the payload to the service provider. The service provider acts on the payload and prepares a response to be sent back to the caller.
4. During response processing, OWSM invokes the `oracle/binding_oes_masking_policy` policy to determine if there is any sensitive data that needs to be masked.

   OWSM passes the caller's information and any of the user-defined attributes extracted from the response payload.
5. The data masking rules defined in OES take into consideration the client information (through transport attributes), the current subject, resource, action and any response attributes configured on the policy.
6. For each payload attribute, OES responds with About XACML Obligations that specify whether the attribute should be passed as-is, or masked.
7. OWSM honors the About XACML Obligations returned by OES and masks attributes marked as sensitive by OES.

2.16.2.3 About XACML Obligations

OES supports the XACML concept of **Obligations**. As described in "Understanding the Policy Model" in *Oracle Fusion Middleware Administering Oracle Entitlements Server*, when used in a policy, an Obligation may impose an additional requirement for the policy enforcing component.

You configure the Obligation in the OES console. An Obligation is any attribute name/value pair (or any other simple name/value pair) that is returned back to the caller (OWSM). For OWSM OES integration, the Obligation can be an XPath query, HTTP transport header properties, or message context properties.

Another use of Obligations is data masking. In certain applications, such as data security use-cases, a simple yes or no answer may not be sufficient and the OES authorization policy might return an Obligation that specifies what data is to be masked and with what value, as previously shown in Figure 2-3.

2.16.2.4 Overview of OES Fine- and Coarse-Grained Authorization

There are two ways to do authorization with the OWSM OES policies: fine- and coarse-grained authorization. The authorization types are defined in the following topics:

- OES Fine-Grained (Obligations)
- Fine-Grained with SAML
- OES Coarse-Grained Authorization

2.16.2.4.1 OES Fine-Grained (Obligations)

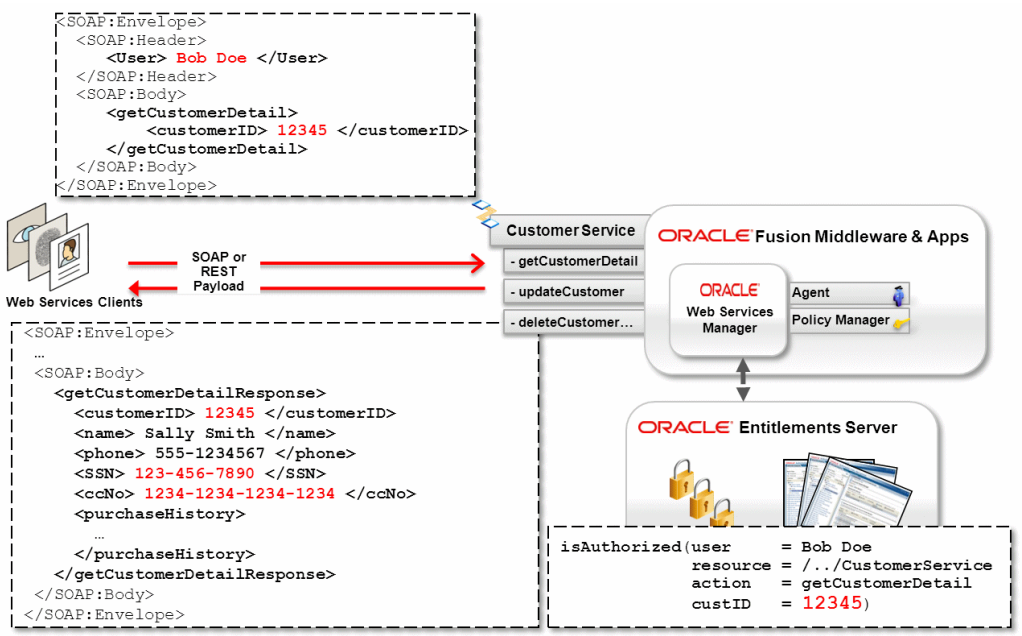
You want to determine access to the resource based on the identity of the consumer, plus specific content from the transport header or the payload specified in Obligations. This is the common use case.

In this use case, you define attributes in the OES access policy that OWSM will then extract from the request and pass back to OES during authorization. That is, the OES access policy is based on a combination of identity attributes or attributes extracted from the request payload.

For example, you might have an OES access check of "Allow access if the SOAP Body contains a particular customer ID and if the authenticated user belongs to group TrustedPartners."

Figure 2-4 shows the fine-grained authorization use case.

Figure 2-4 Fine-Grained Authorization



Consider the following code flow for the web service response shown in Figure 2-4.

1. The web service client sends a SOAP request.
2. The web service is secured with an OWSM authentication policy and an OWSM OES authorization policy.
3. OWSM performs authentication and invokes OES for authorization.
4. OWSM provides the subject, resource, lookup action and all predefined properties to OES.
5. OES calls the lookup action configured for the protected resource and responds with the configured Obligations (if any) to OWSM. Returned obligations can be returned based on actions; for each action you can define different XPaths, and so forth.
6. OWSM evaluates the Obligations and executes the XPath on the SOAP/XML payload or finds the property values from the transport header or message context.
7. OWSM again provides the subject, resource, and action, plus all of the attributes evaluated in Step 6 to OES.
8. OES determines access based on the subject, resource, action and attributes.
9. OES responds with permit or deny.
10. If permit, OWSM passes on the message to the service provider.
11. If deny, OWSM rejects the request with an authentication failure fault.

2.16.2.4.2 Fine-Grained with SAML

You want to determine access to the resource based on the identity of the consumer and on the attributes passed in a SAML token. (See "user.attributes" and "user.roles.include" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for information on passing SAML attributes.

OWSM passes attributes from a SAML assertion. SAML attributes are part of the implicit attributes that are always extracted (if present) and sent automatically. The name of the attribute is the name of the attribute inside the SAML assertion and the value is the list of strings. OES can determine access based on these SAML attributes, as well as the subject, resource and action.

There are two ways to implement this use case. The first approach is to create an OES custom attribute retriever, as described in "Creating Custom Attribute Retrievers" in *Oracle Fusion Middleware Developer's Guide for Oracle Entitlements Server*. The second approach is to have OES respond using Obligations with XPaths that point to the SAML attribute values.

Consider the following code flow for the approach of using an OES custom attribute retriever:

1. The web service client sends a SAML token with an attribute statement in a SOAP request.
2. The web service is secured with `oracle/wss10_saml_token_service_policy` and an OWSM OES authorization policy.
3. OWSM performs authentication and checks the SAML assertion for an `AttributeStatement`. If attributes are present, then OWSM extracts them and passes them as attributes while invoking OES for the access request.
4. OWSM provides the subject, resource, and action, along with any other pre-defined attributes.
5. OES determines access based on the SAML attributes, subject, resource and action. OES uses a custom attribute retriever to get the SAML attributes.
6. OES responds with permit or deny.
7. If permit, OWSM passes on the message to the service provider.
8. If deny, OWSM rejects the request with an access-denied fault.

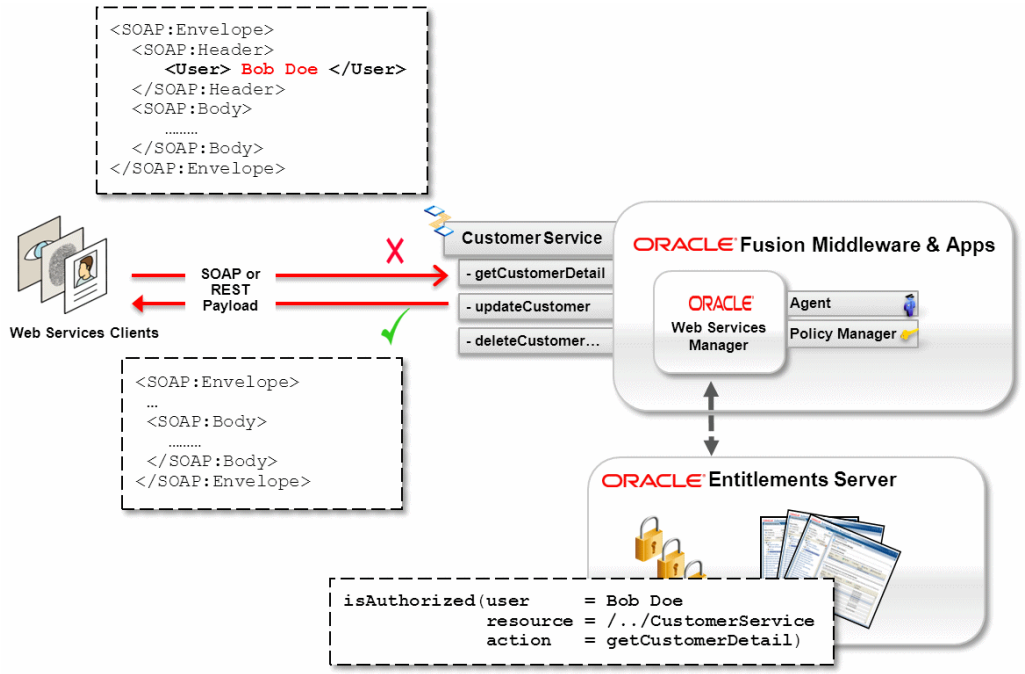
2.16.2.4.3 OES Coarse-Grained Authorization

You want OES to determine access to the resource based on the identity of the consumer and the web service operation being called. The OES access check is based on the identity attributes, which are limited to user name, group, and role. You can also use the implicit attributes, plus OES predefined attributes such as time, date, and so forth.

You must set `use.single.step` to `true` in the OWSM OES policy to use this mode.

Figure 2-5 shows the coarse-grained authorization use case. In this use case, assume that you want to secure the service with an authorization policy that determines whether the consumer is allowed to access the service. You want to determine access to the resource based on the identity (authenticated subject) of the consumer and the web service operation being invoked. For example, in Figure 2-5 user `Bob Doe` might be authorized to get the customer detail but not to delete the customer record.

Figure 2-5 Coarse-Grained Authorization



Consider the following code flow for the web service response shown in Figure 2-5.

1. The web service client sends a SOAP or XML request.
2. The web service is secured with an OWSM authentication policy and an OWSM OES policy.
3. OWSM performs authentication and invokes OES for the access request. OWSM provides the subject, resource and action information to OES.
4. OES determines access based on the subject, resource and action information.
5. OES responds with permit or deny.
6. If permit, OWSM passes on the message to the service provider.
7. If deny, OWSM rejects the request with an access-denied fault.

2.16.3 About OWSM OES Policies

OWSM includes the following OES authorization and masking policies.

See "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for specific configuration information for each of the policies.

- `oracle/binding_oes_authorization_policy` — This policy does user authorization based on the policy defined in OES. Authorization is based on attributes, the current authenticated subject, and the web service action invoked by the client.

  This policy is used for coarse- or fine-grained authorization on any operation on a web service, as determined by the `use.single.step` attribute. (See Overview of OES Fine- and Coarse-Grained Authorization.)

You must use an authentication policy with the OWSM OES authorization policy because the OWSM OES policy requires an authenticated subject.

This policy also uses the guard element (see orawsp:guard) to define resource, action, and constraint match values. These values allow the assertion execution only if the result of the guard is true. If the accessed resource name and action match, only then is the assertion allowed to execute. By default, resource name and action use the wildcard asterisk "*" and everything is allowed.

This policy can be attached to any SOAP-based endpoint.

- `oracle/component_oes_authorization_policy` — This policy does user authorization based on the policy defined in OES.

  This policy is used for coarse- or fine-grained authorization on any operation on a SOA component, as determined by the `use.single.step` attribute. (See Overview of OES Fine- and Coarse-Grained Authorization.)

  You must use an authentication policy with the OWSM OES authorization policy because the OWSM OES policy requires an authenticated subject. Authorization is based on attributes, the current authenticated subject, and the web service action invoked by the client.

  This policy also uses the guard element (see orawsp:guard) to define resource, action, and constraint match values. These values allow the assertion execution only if the result of the guard is true. If the accessed resource name and action match, only then is the assertion allowed to execute. By default, resource name and action use the wildcard asterisk "*" and everything is allowed.

  This policy is used for fine-grained authorization on a SOA component.

- `oracle/binding_oes_masking_policy` — This policy does response masking based on the policy defined in OES. You can use an authentication policy with the OWSM OES masking policy. (If there is no subject, the masking decision does not consider the user when making a decision.) Masking is based on attributes, the current authenticated subject, and the web service action invoked by the client.

  This policy uses the guard element (see orawsp:guard) to define resource, action, and constraint match values. These values allow the assertion execution only if the result of the guard is true. If the accessed resource name and action match, only then is the assertion allowed to execute. By default, resource name and action use the wildcard asterisk "*" and everything is allowed.

  This policy is used for fine-grained masking on any operation of a web service.

2.16.4 Overview of Resource Mapping and Naming

You must map the OES resource name to the OWSM resource name. When making an authorization call from OWSM, the resource name is passed to OES, and this name must exactly match the one defined in the OES policy.

The following topics include:

- Resource Mapping and Naming
- Example of OES Policies

2.16.4.1 Resource Mapping and Naming

Table 2-1 shows how to construct the resource string for the OES policy.

If you follow the naming conventions, you do not have to set the resource name in the OWSM policy, OWSM derives it.

> **Note:**
>
> This is the default mapping. If you need to change this mapping, use configuration overrides, as described in "Configuration Properties and Overrides" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Table 2-1 Determining Resource String

OES Field	Value to Use
Application	Deployed Application Name. For SOA, the composite name is used as the application name.
Resource Type	Fixed, based on subject type. • For SOAP must be `WS_SERVICE`. • For SOA component, must be `COMPONENT`.
Resource Name	• For SOAP and SOA reference, must be of the form `web-service-name/port/web service operation`. • For SOA component, must be of the form `SOA component name/web service operation`.
Action	By default, one of: `request.lookup` (Obligation policy for authorization.) `response.lookup` (Obligation policy for masking.) `mask` (Real masking policy.) `authorize` (Real authorization policy.)

2.16.4.2 Example of OES Policies

Assume that a SOA composite (soa1) has two service bindings (`Serv1` and `Serv2`).

- `Serv1` has `port11`
- `Serv2` has `port21`
- `port11` has `oper11`, `oper12`
- `port21` has `oper21`

In OES, the application, resource type, resource name and actions should be defined as shown in Table 2-2.

Table 2-2 Resource String Example

OES Field	Value to Use
Application	soa1

Table 2-2 (Cont.) Resource String Example

OES Field	Value to Use
Resource Type	`WS_SERVICE`
Resource Name	`Serv1/port11/oper11`, `Serv1/port11/oper12`, `Serv2/port21/oper21`
Action	One of: `request.lookup` (Obligation policy for authorization.) `response.lookup` (Obligation policy for masking.) `mask` (Real masking policy.) `authorize` (Real authorization policy.)

The authorization and masking OES policies based on Table 2-2 are as follows:

- Returning Obligations
  - One policy that returns obligations for any operation:

```
GRANT (action: request.lookup; Resource: WS_SERVICE/Serv1/Port11, WS_SERVICE/
Serv2/Port21; User: any) Obligation: XPath11
```

  - Multiple policies for returning operation-specific Obligations:

```
GRANT (action: request.lookup; Resource: WS_SERVICE/Serv1/Port11/
oper11;User:any) Obligation: XPath11
GRANT (action: request.lookup; Resource: WS_SERVICE/Serv1/Port11/
oper12;User:any) Obligation: XPath12
GRANT (action: request.lookup; Resource: WS_SERVICE/Serv2/Port21/
oper21;User:any) Obligation: XPath21
```

- Real authorization
  - One policy performing same authorization regardless of resource and action:

```
GRANT/DENY (action: authorize; Resource: WS_SERVICE/Serv1/Port11, WS_SERVICE/
Serv2/Port21; User:<actual user>)
```

  - Multiple policies for performing operation-specific authorization:

```
GRANT/DENY (action: authorize; Resource: WS_SERVICE/Serv1/Port11/
oper11;User:<actual user>)
GRANT/DENY (action: authorize; Resource: WS_SERVICE/Serv1/Port11/
oper12;User:<actual user>)
```

- Returning masking Obligations:

```
GRANT (action: response.lookup; Resource: WS_SERVICE/Serv1/Port11/
oper11;User:any) Obligation: XPath11
GRANT (action: response.lookup; Resource: WS_SERVICE/Serv1/Port11/
oper12;User:any) Obligation: XPath12
GRANT (action: response.lookup; Resource: WS_SERVICE/Serv2/Port21/
oper21;User:any) Obligation: XPath21
```

- Real masking:

```
GRANT/DENY (action: mask; Resource: WS_SERVICE/Serv1/Port11/oper11;User:<actual
user>)
```

```
GRANT/DENY (action: mask; Resource: WS_SERVICE/Serv2/Port21/oper21;User:<actual
user>)
```

2.16.5 How Attributes Are Processed

As the OES administrator, you define attributes in the OES policy as Obligations, which OWSM then extracts from the payload and sends back to OES.

Specifically, OES allows you to create an Obligation in the OES console and provide multiple attribute name/value pairs. For example, you can create an Obligation called `Employee` and have multiple attributes such as `{Name=John, Age=21, SSN=123456}`.

The attributes can be obtained from an XPath, an HTTP header, a message context, and constants (name/value). These attributes must follow a specific naming convention, as described in Table 2-3.

Table 2-3 Attribute Types Supported for OES Policies

Attribute Type	Description	Required Format

XPath query	You provide the attribute name and value as an XPath query in the OES console.
	
	OWSM runs this XPath query on the SOAP message and uses the value as the attribute value. The XPath query can result in a single value or multiple values. In case of multiple values, a list of strings is used to pass all values.
	
	If any XPath query fails to evaluate on the SOAP message, it is ignored and a warning message is generated in the logs. OWSM continues to evaluate next XPath query.
		Use `XPath` (case insensitive) as the Obligation name to signify that it is an XPath.
		
		The Obligation should also return all the namespaces being used in the XPath query. All namespaces should be returned with an attribute name of `NAMESPACE` (case insensitive) and the value should be the comma separated namespaces.
		
		For example, if you want to use the SAML issuer name for authorization, use the following Obligation format:
		
		`Name = XPath, values = {saml_issuer=.//saml:Assertion/@Issuer, CC_Name=ns1:sayHello/arg0, NAMESPACE=ns1=http://...,wsse=http://...}`
		
		In the authorization phase, OWSM passes the attribute name `saml_issuer` and the value is the result of the XPath query. The default namespace has to be mapped to a prefix. (The prefix name must be unique within the application.)
		
		For example:
		
		`saml=urn:oasis:names:tc:SAML:1.0:assertion,ns0=http://example.com,myPrefix=http://default_namespace`
		
		Namespace definitions are separated using a comma.

Table 2-3 (Cont.) Attribute Types Supported for OES Policies

Attribute Type	Description	Required Format
HTTP Header	You provide HTTP header names in the OES console. The value is fetched from the current request HTTP header.	To get HTTP Header properties, define an Obligation with the name "HTTPHeader" (case insensitive). It can have multiple HTTP header names. The name of the attribute should be the name to which you want to assign the value; the value should be the actual HTTP header name. For example: `Name = HTTPHeader, values = {AuthHeader=Authorization}` In the authorization phase, OWSM retrieves the HTTP header and assigns it to the name given in the attribute name.

Message Context Properties	You provide message context property names in the OES console.
	The value is fetched from the current message context.
		Define an Obligation with the name "MessageContext" (case insensitive). It can have multiple message context property names.
		The name of the attribute should be the name to which you want to assign a value; the value should be the actual message context property name.
		For example:
		`Name = MessageContext, values = {authMethod=oracle.wsm.internal.authentication.method, endpoint=oracle.j2ee.ws.runtime.endpoint-url}`
		In the authorization phase, OWSM retrieves the message context property and assigns it to the name given in the attribute name.
		For example, the previous example might resolve to:
		`authMethod=USERNAME_TOKEN & endpoint=http://localhost:7001/myService`

Constants	Constants are user-defined attributes that OWSM does not understand and passes "as is."	An Obligation named `Employee` is an example of a constant.

Table 2-3 (Cont.) Attribute Types Supported for OES Policies

Attribute Type	Description	Required Format
Implicit	OWSM passes implicit attributes in all authorization requests. You do not perform any configuration to pass them. The following implicit attributes are always passed: • `serviceURL` — The URL of the web service. • `serviceNS` — The namespace of the web service. • `clientIP` — The client's IP address. • `processingStage` — Whether this is a request or response. Possible values are request, response, and fault. • `isRequestOverSSL` — Boolean. True if the request is over one- or two-way SSL.) • `authenticationMethod` — The authentication method. Possible values are `SAML_SV`, `KERBEROS`, `SAML_HOK`, `X509_TOKEN_AUTHENTICATION`, `SAML_BEARER`, and `USERNAME_TOKEN` • `requestOrigin` — Where the request came from, internal or external, as determined from the VIRTUAL_HOST_TYPE transport header. • `clientSigningCertDN` — Either the X509 signing cert or the client cert in two-way SSL. • `operationName` — The operation name invoked by the user. • `samlIssuer` — The SAML issuer extracted from the SAML assertion. • `type` — The type of the request to OES. Values can be request.lookup, response.lookup, authorize or mask. This attribute is always sent.	None required, they are always passed. You would typically use these constants in a Condition in the OES console.

2.16.6 About the Guard Element

The OWSM OES authorization policies uses the `orawsp:guard` element. It allows the assertion to execute only if the result of the guard is true. That is, if the accessed resource name and action match, only then OES authorization engine is called.

By default, resource name and action use the wildcard asterisk "*" and everything is allowed. However, if you set a specific resource name, action, and constraint, that requirement must be satisfied before any of the configuration properties and any OES policies are considered.

The resource naming convention for guard differs from the OES standard naming convention. The resource name for the guard must be in the form `<Webservice_NS>/<SERVICE_NAME>`.

2.17 Overview of Personally Identifiable Information

Personally Identifiable Information (PII) refers to Social Security numbers, addresses, bank account numbers, and other similar information that is typically associated with one specific user and must generally be protected.

OWSM provides a solution for protecting PII when outside the control of a security policy so that PII is hidden in logs, in messages, in audits, and so forth.

- Overview of PII Data
- About PII Policy XPath Expressions
- When to Use the PII Policy
- Who Should Have Access to the PII
- About Additional Considerations for Unmarshalling

2.17.1 Overview of PII Data

The OWSM WS-Security policies provide a way to selectively encrypt information through message protection. However, there may be times when this information is outside of the control of a security policy and not encrypted, such as when it is being processed inside a SOA composite.

PII data is described in the following topics:

- About PII Data
- About the PII Security Policy

2.17.1.1 About PII Data

Your business practices may require that information, and particularly PII information, be encrypted even as it flows within your applications. If this is true for your environment, PII information should remain encrypted as the message flows to various components. For example, in the case of SOA, it means that PII must be encrypted at the entry point of a SOA composite and must be decrypted at the reference binding exit point.

The `oracle/pii_security_policy` policy is provided to encrypt data for this purpose.

See "About Additional Considerations for Unmarshalling" for additional information when unmarshalling non-string data.

> **Note:**
>
> Although the OWSM WS-Security policies provide a way to encrypt information, this mechanism cannot be used to encrypt PII data because your applications might expect and depend on the information having a specific XML structure. The OWSM policies add `CipherData` (`http://www.w3.org/TR/xmlenc-core/#sec-CipherData`) elements to the XML structure, which your applications might not expect.

2.17.1.2 About the PII Security Policy

The `oracle/pii_security_policy` contains the following settings and configuration properties that specify exactly which PII data you want to protect. You can set these attributes when you attach a policy, and override them as required.

> **Note:**
>
> Local optimization (see "Using Local Optimization with OWSM Policies (SOA Composites")) is off by default so that the `oracle/pii_security_policy` policy is always enforced and decrypts PII before leaving the SOA composite. Do not turn local optimization on for the `oracle/pii_security_policy` policy without understanding the ramifications.

See "About PII Policy XPath Expressions" for a description of how XPaths are used to specify the PII data.

- encryption-algorithm — The data encryption algorithm, which must be `AES/CBC/PKCS5Padding`.
- algorithm — The key derivation algorithm, which must be PBKDF2.
- Salt — A non-null and non-empty salt for key derivation. The default value is `pii-security`.
- Iteration — The iteration count for key derivation. The default is 1000.
- Keysize — The size of the key for key derivation. The default is 128.
- request.xpaths — A comma-separated list of XPaths for the request. Default value is blank. For example, `//ns2:ShipToLocationId`.
- request.namespaces — A comma-separated list of namespaces for the request, where each namespace has a prefix and URI separated by the equals = sign. Default value is blank.
- response.xpaths —A comma-separated list of XPaths for the response. Default value is blank. For example, `//ns2:ShipToLocationId`.
- response.namespaces — A comma-separated list of namespaces, where each namespace has a prefix and URI separated by the equals = sign. Default value is blank.

- csf.key — The `oracle/pii_security_policy` policy uses the password CSF key attribute you specify to generate a symmetric key. This key is then used to encrypt and decrypt the PII data. Default value is `pii-csf-key`.
- reference.priority — See "Specifying the Priority of a Policy Attachment".

> **Note:**
>
> The pii-security assertion must be the only assertion in a policy. You must not add the pii-security assertion to a policy with any other assertion. If you do so, the policy is invalid.

2.17.2 Example of How PII Data is Protected

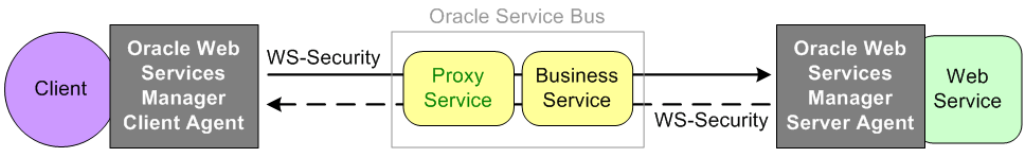
It is important to protect PII with the `oracle/pii_security_policy` policy when outside the control of a security policy.This is explained by the way of example.

Consider the Oracle Service Bus example shown in Figure 2-6. As shown in Figure 2-6, the guiding principle for protecting PII is as follows:

- A PII policy attached at the service side (proxy service) must encrypt PII after receiving request and decrypt PII before sending out a response.
- A PII policy attached at the client side (business service) must decrypt PII before sending out request and encrypt after receiving a response.

Encrypting PII data requires both entry and exit points: PII data is encrypted before entry and decrypted before exit.

Figure 2-6 PII Encryption in Oracle Service Bus



The flow of control in Figure 2-6 is as follows:

1. The OWSM web service client signs and encrypts a client request and sends the request to the proxy service.
2. To virtualize an external web service, you create an OSB proxy service that uses a pipeline to connect to a business service. Therefore, you attach OWSM service side policies to the proxy.

   The OWSM Agent decrypts the message using the message protection policy.
3. Any PII information would now potentially be vulnerable. To prevent that, the PII policy encrypts the PII in the message.
4. Oracle Service Bus assumes control of the message, and the proxy service passes the request to the business service.

5. The business service accesses the message data, perhaps including the now-encrypted PII data, as needed.
6. The OWSM agent again applies the PII policy and decrypts the PII fields.
7. A business service is basically all the client configuration needed to call an external service. Oracle Service Bus supports attaching OWSM client policies to a business service.

   The message protection policy is applied (signed, encrypted) and the message is sent to the web service.
8. The process is reversed for the response back to the web service client.

2.17.3 About PII Policy XPath Expressions

You use XPath expressions to specify the PII data to be protected.

More specifically, you use XPath expressions to specify exactly which elements you want to protect. The XPath must end in an XPath text node.

If the result of the XPath is a text node, the contents of the text node are encrypted. Empty text nodes (that is, containing all whitespace) are not encrypted or decrypted.

If any XPath returns multiple nodes, all of them are encrypted. If nothing is returned, it is ignored.

The specified XPaths will be evaluated with the first element child of `env:Body` set as the root node.

Assume that you use JDeveloper to view the following SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body


xmlns:ns1="http://xmlns.example.com/apps/prc/po/editDocument/purchaseOrderService/
types/">
<ns1:createPurchaseOrder>
 <ns1:createOrderEntry


xmlns:ns2="http://xmlns.example.com/apps/prc/po/editDocument/purchaseOrderService/">
 <ns2:DocumentStyleId>1</ns2:DocumentStyleId>
 <ns2:ProcurementBuId>204</ns2:ProcurementBuId>
<ns2:BuyerId>100010026863783</ns2:BuyerId>
<ns2:RequisitioningBuId>204</ns2:RequisitioningBuId>
<ns2:SupplierId>559</ns2:SupplierId>
<ns2:SupplierSiteId>5058</ns2:SupplierSiteId>
<ns2:SupplierContactId>100000011552368</ns2:SupplierContactId>
<ns2:ApprovalActionCode>BYPASS</ns2:ApprovalActionCode>
<ns2:DocumentDescription>DO NOT TOUCH THIS ORDER (V15)</ns2:DocumentDescription>
<ns2:PurchaseOrderEntryLine>
<ns2:LineTypeId>1</ns2:LineTypeId>
<ns2:ItemId>199</ns2:ItemId>
<ns2:Quantity>10</ns2:Quantity>
<ns2:UnitOfMeasureCode>Ea</ns2:UnitOfMeasureCode>
<ns2:PurchaseOrderEntrySchedule>
<ns2:ShipToLocationId>207</ns2:ShipToLocationId>
<ns2:ShipToOrganizationId>207</ns2:ShipToOrganizationId>
<ns2:NeedByDate>2020-12-31</ns2:NeedByDate>
<ns2:PurchaseOrderEntryDistribution />
```

```
</ns2:PurchaseOrderEntrySchedule>
</ns2:PurchaseOrderEntryLine>
</ns1:createOrderEntry>
</ns1:createPurchaseOrder>
</soap:Body>
</soap:Envelope>
```

Some XPath examples are as follows:

- `//ns2:ShipToLocationId`

  Use the `//` notation (descendant-or-self axis) to indicate that you want to search the whole body for `ShipToLocationIds`. These XPaths require a complete document search and are slower.

- `/ns1:createPurchaseOrder/ns1:createOrderEntry/ns2:BuyerId`

  This XPath clearly specifies that you want to look at the `createPurchaseOrder` child of the body, and then the `createOrderEntry` child of the `createPurchaseOrder`, and finally the `BuyerId` child of `createOrderEntry`.

The PII policy requires XPath lists for both the request and response messages.

2.17.4 When to Use the PII Policy

You can use the `oracle/pii_security_policy` policy **only** in single-SOA composite use cases, and uses cases where PII is at the JCA binding.

> **Note:**
>
> Global Policy Attachment is not recommended with the `oracle/pii_security_policy` policy.

Different usecases are explained in the following topics:

- Single SOA Composite Use Case
- Oracle Service Bus Proxy Service to Business Service Use Case
- PII at the JCA Binding Use Case

> **Note:**
>
> Fusion Middleware Control and JDeveloper control produce runtime validation errors if you attach the `pii_security_policy` policy to a non-supported subject type.
>
> With WLST, validation errors are generated if you attach the policy to a non-supported subject type:
>
> ```
> The web service configuration is invalid in this context because of the
> following error: WSM-01832 : PII Policy oracle/pii_security_policy is not
> supported on the Resource, as the PII policy
> is not supported on SubjectType : WS_SERVICE
> ```

2.17.4.1 Single SOA Composite Use Case

You can attach the `pii_security_policy` policy only to a SOA composite and only to protect PII within that composite. No other SOA use case is supported.

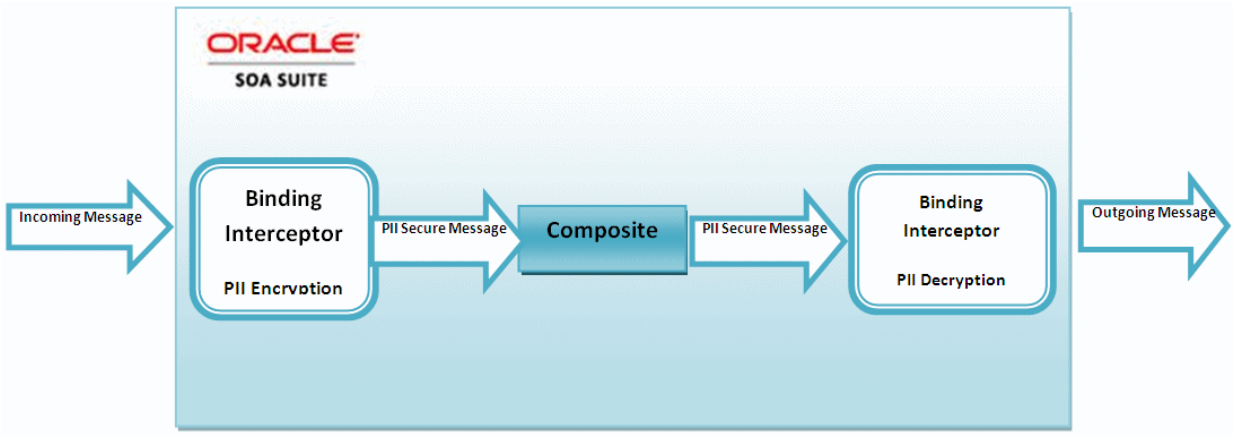
When you attach the `pii_security_policy` policy to a SOA composite, the PII is encrypted when a message enters the SOA composite and decrypted when exiting from the composite at a SOA reference binding component. Specifically:

- At the service side (service binding), the `pii_security_policy` policy encrypts PII after receiving a request and decrypts PII before sending out a response.
- At the client side (reference binding), the `pii_security_policy` policy decrypts PII before sending out a request and encrypts PII after receiving a response.

> **Note:**
>
> PII data requires both entry and exit points: PII data is encrypted before entry and decrypted before exit.
>
> When you attach the `pii_security_policy` policy at the client side, you must also attach it at the service side, and vice versa. The encryption/decryption mechanism requires both pieces to be in place.

As shown in Figure 2-7, the PII remains encrypted as the message flows to various components of the composite such as a BPEL, Oracle Mediator, and so forth.

Figure 2-7 Single SOA Composite Use Case



Understanding important considerations when both PII and Authorization Policy are attached

If the `pii_security_policy` policy and an authorization policy are both attached to a SOA composite, the authorization policy is executed before the PII policy. Otherwise, the `pii_security_policy` policy might encrypt the field to be used for authorization.

However, if the authorization policy is instead attached at the SOA component level, and authorization requires a field that is encrypted by `pii_security_policy`, authorization fails. This is not a supported use case.

2.17.4.2 Oracle Service Bus Proxy Service to Business Service Use Case

See "Hiding Personally Identifiable Information in Messages" in Developing Services with Oracle Service Bus for information on how to attach `oracle/pii_security_policy` to Oracle Service Bus.

2.17.4.3 PII at the JCA Binding Use Case

You can attach the PII policy to JCA adapters for both SOA and Oracle Service Bus.

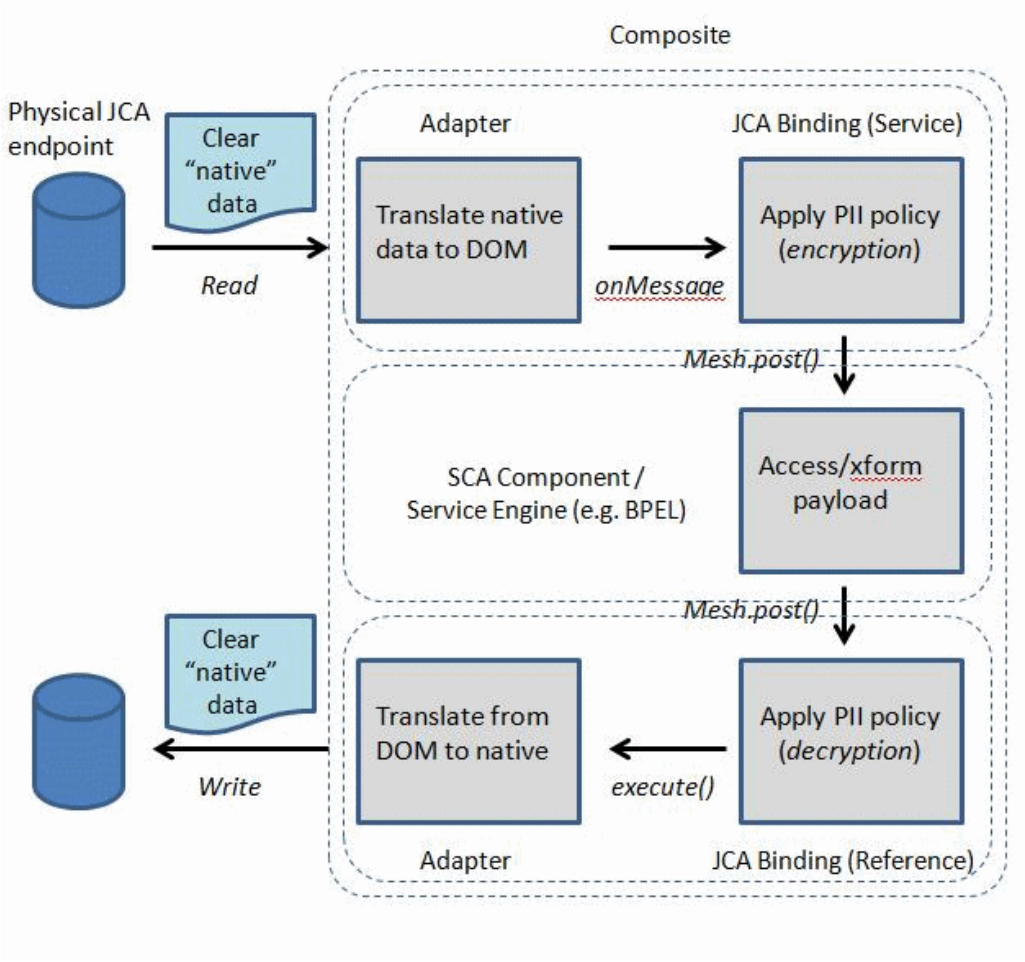
For OSB, see "Hiding Personally Identifiable Information in Messages" in *Developing Services with Oracle Service Bus* for information on how to attach `oracle/pii_security_policy` to Oracle Service Bus.

The remainder of this section describes the SOA use case.

As described in "JCA Adapters", in *Developing SOA Applications with Oracle SOA Suite*, JCA adapters enable you to integrate SOA services and references with technologies such as databases and file systems. JCA adapters integrate with the JCA binding component of the Oracle Fusion Middleware platform, thereby integrating with other service engines and binding components.

Consider the JCA adapter PII use case shown in Figure 2-8.

Figure 2-8 JCA Adapter PII Use Case



The composite includes an inbound service binding component (an inbound adapter), a service component such as a BPEL process, and an outbound reference binding component (an outbound adapter).

> **Note:**
>
> PII data requires both entry and exit points: PII data is encrypted before entry and decrypted before exit.
>
> When you attach the `pii_security_policy` policy at the reference binding side, you must also attach it at the service binding side, and vice versa. The encryption/decryption mechanism requires both pieces to be in place.

In this use case, PII is protected as follows:

- Service binding components provide the outside world with an entry point to the SOA composite application.

At the service side (JCA binding), the `pii_security_policy` policy encrypts PII after receiving a request and decrypts PII before sending out a response.

- The PII remains encrypted as the message flows to various components of the composite such as a BPEL, Oracle Mediator, and so forth.
- Reference binding components enable messages to be sent from the SOA composite application to external services in the outside world.

  At the client side (JCA binding (reference)), the `pii_security_policy` policy decrypts PII before sending out a request and encrypts PII after receiving a response.

See Oracle SOA Composite Integration with Adapters for additional information on how JCA adapters integrate with SOA environments.

2.17.5 Who Should Have Access to the PII

As the administrator, you always have access to the keys used to encrypt the PII and you can reconfigure PII encryption. Therefore, the PII policy does not protect data from an administrator or any one else with these administrator privileges, and it is not intended to do so.

However, as the administrator, you need to make sure that the encryption key and PII data are not visible in the following scenarios:

- In any kind of logs, including the OWSM message logs, the server diagnostic logs, SOA message logs, and so forth. Logs are often copied and made available to non-administrative users.
- In any screens that can be viewed by non-administrative users.

  Such a user should not be able to see any PII information. This user can view logs, but the logs will have encrypted PII data, and this user will not be able to view the key information required to decrypt them.
- To roles such as "Operator", "Monitor," and so forth. These roles should not be able to access either the PII encryption keys or the PII data, and should not be able to access any log files that contain decrypted PII data.

2.17.6 About Additional Considerations for Unmarshalling

Unmarshalling converts an XML document to create a tree of Java program elements, or objects, that represents the content and organization of the document that can be accessed by your Java code. In the content tree, complex types are mapped to value classes. Attribute declarations or elements with simple types are mapped to properties or fields within the value class and you can access the values for them using `get` and `set` methods.

Unmarshalling is managed by the JAXB binding framework.

After the PII data is encrypted, the original text in the message is replaced by an encrypted string. However, if there are non-string data types (integer, date, and so forth) in the encrypted string, any subsequent unmarshalling may break.

Before you implement the `pii_security_policy` policy, be aware of the implications for unmarshalling: if unmarshalling might be involved, then only the string data type works and others may break.

Unmarshalling can happen in SOA at the following bindings:

- EJB Adapters
- Legacy SDO EJB adapter
- ADF Adapter
- Direct bindings
- Rules engine

Unmarshalling can happen at the following SOA components:

- BPEL Entity Variable
- Spring service engine

Unmarshalling can happen in Oracle Service Bus at the following points:

- SOA direct bindings
- EJB transport

2.18 Understanding OAuth 2.0 for REST and SOAP Services and Clients

Oracle Web Services Manager allows web service clients to interact with the Mobile and Social OAuth 2.0 server implementation for both SOAP and REST web services, for "2-legged" authorization.

For more information, see "Using OAuth2 with Oracle Web Services Manager" in *Oracle® Fusion Middleware Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.19 Understanding REST APIs for Managing Credentials and Keystores

The credential and keystore management REST API provides endpoints for creating and configuring credential stores, keystores, and trust stores for your domain or Web services.

For more information, see Introduction to REST API in *REST API for Managing Credentials and Keystores with Oracle Web Services Manager*.

3

Understanding the OWSM Policy Framework

OWSM policy framework manages and secures web services consistently across your organization.
OWSM policies are described in the following sections:

- Overview of OWSM Policy Framework
- Understanding Web Service Policies
- Overview of Building Web Service Policies Using Policy Assertions
- Understanding Policy Subjects
- Overview of Attaching Policies to Policy Subjects
- Understanding How Policies are Executed
- About OWSM Predefined Policies and Assertion Templates
- About Overriding the Security Policy Configuration
- About Recommended Naming Conventions for Documents Created in WSM Repository

3.1 Overview of OWSM Policy Framework

Oracle Web Services Manager (OWSM) provides a policy framework to manage and secure web services consistently across your organization. It provides capabilities to build, enforce, run and monitor web service policies, such as security, reliable messaging, MTOM, and addressing policies. OWSM can be used by both developers, at design time, and system administrators in production environments.

OWSM policy framework includes the following topics:

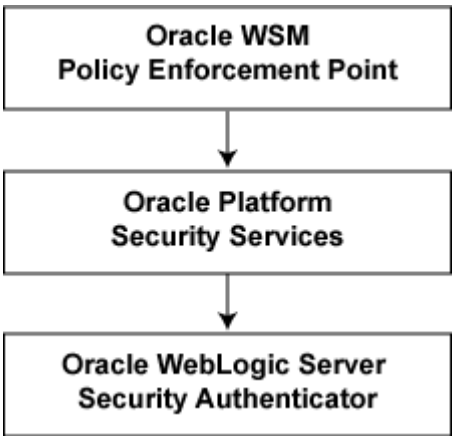
- About OWSM Policy Framework Components
- Understanding OWSM Agent and Policy Manager Interaction
- About OWSM Agent and Policy Manager Characteristics
- Understanding the OWSM Agent and Policy Manager Request Flow
- About OWSM Configuration Artifacts

3.1.1 About OWSM Policy Framework Components

The OWSM policy framework is built using the WS-Policy standard.

The OWSM Policy Enforcement Point (PEP) leverages Oracle Platform Security Service (OPSS) and the Oracle WebLogic Server authenticator for authentication and permission-based authorization, as shown in Figure 3-1.

Figure 3-1 OWSM Policy Framework Leverages OPSS and Oracle WebLogic Server Security



The OWSM Policy Manager, Agent, Repository, and Enterprise Manager form the Policy Framework.

- **Policy Manager** reads and writes policies including predefined and custom policies from the OWSM Repository. In Oracle SOA installations it is typically deployed on the Oracle SOA Service Infrastructure managed servers. You can deploy the Policy Manager on separate Managed Servers.
- **Agent** is responsible for policy enforcement, execution and gathering of runtime statistics. The OWSM Agent is available on all Oracle Fusion Middleware Managed servers. It is configured on the same server as the application it protects.

  The OWSM Agent is made up of a set of jar files, which are a part of underlying web service stack. It does not have any session state. The Agent maintains an in-memory policy cache, which is populated at the Agent startup time. It does not use any JTA or JMS.

  The OWSM Agent is made up of the following two pieces:

  – **Policy Access Point (PAP)** communicates with Policy Manager. The Agent communicates with the Policy Manager through EJB invocations.

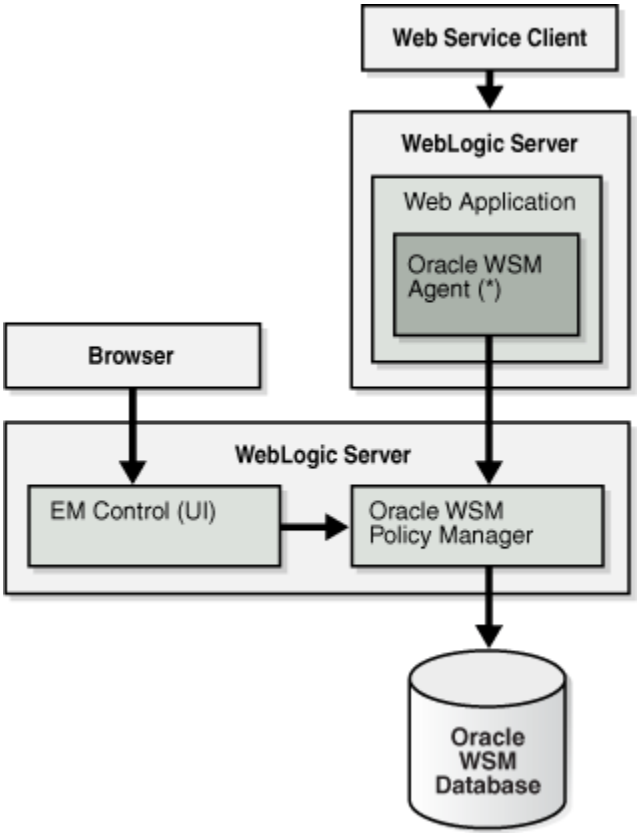
  – **Policy Interceptor** is generated when a web service is deployed and activated, or when a policy is attached to a web service using Enterprise Manager. If new web services are protected using OWSM, an additional instance of the interceptor is generated for each new web service. Interceptor is responsible for policy enforcement.
- **OWSM Repository** Policies are stored in the OWSM Repository. It is typically backed by an Oracle database. For high availability purposes, Oracle recommends using an Oracle RAC database as the back end for OWSM Repository.
- **Enterprise Manager** is used to configure OWSM. It also displays different web services metrics gathered by OWSM.

3.1.2 Understanding OWSM Agent and Policy Manager Interaction

The OWSM Agent expects the OWSM Policy Manager to be deployed on at least one node of the domain.

A high-level view of the interaction between the OWSM Agent and the OWSM Policy Manager is shown in Figure 3-2.

Figure 3-2 OWSM Agent and Policy Manager Interaction



The Policy Manager is a stateless application which does not perform any caching. There is no special application level startup sequence performed when the Managed Server where the Policy Manager is deployed starts up. The Policy Manager communicates with the OWSM Repository to retrieve policies. The OWSM Repository can be stored in a database to provide MDS high availability.

The OWSM agent has an auto-discovery feature to locate and connect to an OWSM Policy Manager. See "Configuring OWSM Policy Access Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for additional information.

When the Agent connects to the Policy Manager, it downloads and caches the latest revision of policies. Once the Agent is up and running, it periodically attempts a cache refresh at a configurable interval. The default time is every 10 minutes.

For high availability scenarios, if an OWSM application is targeted to multiple nodes, it should be targeted to a cluster rather than to individual Managed Servers.

If a Managed Server has web services deployed that are protected by OWSM, and the OWSM Agent is not able to communicate with any of the Policy Managers at startup time, web service invocation fails.

3.1.3 About OWSM Agent and Policy Manager Characteristics

The OWSM Agent is a set of JAR files available on every Oracle Fusion Middleware Managed server in a web services stack.

The Policy Manager is contained in the `wsm-pm.ear` file. None of the services provided by OWSM are singletons, therefore, it can run in full active-active mode. OWSM services can be validated by `http://host:port/wsm-pm/validator`. This validator displays OWSM policies, assertion templates, and the contribution details such as build label and creation timestamp.

The OWSM Agent and Oracle Enterprise Manager interact with the Policy Manager using the EJB interfaces. The EJBs used in OWSM are stateless and can be deployed in a clustered environment. Therefore, there is no requirement to enable state replication in the cluster.

The OWSM Agent and Policy Manager need not be co-located. However, the Agent expects the Policy Manager to be deployed on at least one node of the domain. The OWSM Agent has capabilities to auto-discover Policy Managers deployed in the domain.

External Dependencies

The OWSM Policy Manager depends on the following components:

- OWSM Repository for storing the policies
- OWSM Agent depends only on OWSM Policy Manager.

Both components must be available for OWSM to start and run properly.

3.1.4 Understanding the OWSM Agent and Policy Manager Request Flow

When a protected web service is accessed by a client application, the OWSM Agent queries the policy cache and enforces the applicable policies. Based on the policies, the request is authenticated, encrypted, decrypted, authorized or logged. It does not connect to the Policy Manager for any of these operations.

Runtime availability of the Policy Manager does not affect the functioning of the OWSM Agent, unless there is a configuration change, such as new web services, which are protected by OWSM, being deployed, or new policies attached to existing web services. If there is such a configuration change, then the OWSM Agent must connect to the Policy Manager to get the applicable policies. If it cannot connect after initial startup, it continues to operate based on the cached policies.

3.1.5 About OWSM Configuration Artifacts

The OWSM domain configuration settings are available from Oracle Enterprise Manager Fusion Middleware Control and are specific to each OWSM Agent installation.

As described in "Managing OWSM Domain Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, among other settings you can specify:

- Policy Manager URL (if configured)
- Cache Refresh Interval
- Clock skew, to allow for differences in system clock of the client and servers

Other configuration options at the container level, such as data sources for OWSM Repository location, and application targeting, are maintained as part of Oracle WebLogic Server Domain configuration, and are synchronized across a cluster of Oracle WebLogic Servers by Oracle WebLogic Server core infrastructure.

3.2 Understanding Web Service Policies

A web service provider may define conditions (or policies) under which a service is to be provided. The WS-Policy framework enables you to specify policy information that can be processed by web service applications, such as OWSM.

A policy is expressed as one or more policy assertions representing a web service's capabilities or requirements. For example, a policy assertion may stipulate that a request to a web service be encrypted. Likewise, a policy assertion can define the maximum message size that a web service can accept.

WS-Policy expressions are associated with various web services components using the WS-PolicyAttachment specification. WS-Policy information can be embedded in a WSDL file, thus making it easy to expose web service policies through a UDDI registry.

Policies can be attached directly to endpoints or globally to a range of endpoints of the same type, regardless of the deployment state using policy sets.

Oracle Fusion Middleware 12*c* supports the categories of policies defined in Table 3-1. The policies are part of the OWSM enterprise policy framework which allows policies to be centrally created and managed.

Table 3-1 Policy Categories

Policy Category	Description	Applies to SOAP, REST, or Both
Addressing	WS-Addressing policies that verify that SOAP messages include WS-Addressing headers in conformance with the WS-Addressing specification. Transport-level data is included in the XML message rather than relying on the network-level transport to convey this information. For more information on the WS-Addressing, see "Understanding Web Services Addressing".	SOAP

Table 3-1 (Cont.) Policy Categories

Policy Category	Description	Applies to SOAP, REST, or Both
Atomic Transactions	WebLogic web services enable interoperability with other external transaction processing systems, such as WebSphere, Microsoft .NET, and so on, through the support of the following specifications: • WS-AtomicTransaction Version (WS-AT) 1.0, 1.1, and 1.2: `http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html` • WS-Coordination Version 1.0, 1.1, and 1.2: `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html` For more information about atomic transactions, see "Using Web Services Atomic Transactions" in *Developing Oracle Infrastructure Web Services*.	SOAP
Configuration	Configuration policies that enable you to configure web service features, such as Fast Infoset, schema validation, persistence, and so on.	SOAP
Management	Management policies that log request, response, and fault messages to a message log. Management policies may include custom policies.	SOAP
Message Transmission Optimization Mechanism (MTOM) Attachments	Binary content, such as an image in JPEG format, can be passed between the client and the web service. In order to be passed, the binary content is typically inserted into an XML document as an `xsd:base64Binary` string. Transmitting the binary content in this format greatly increase the size of the message sent over the wire and is expensive in terms of the required processing space and time. Using MTOM, binary content can be sent as a MIME attachment, which reduces the transmission size on the wire. The binary content is semantically part of the XML document. Attaching an MTOM policy ensures that the message is converted to a MIME attachment before it is sent to the web service or client.	SOAP
Reliable Messaging	Reliable messaging policies that implement the WS-ReliableMessaging standard describes a wire-level protocol that allows guaranteed delivery of SOAP messages, and can maintain the order of sequence in which a set of messages are delivered. The technology can be used to ensure that messages are delivered in the correct order. If a message is delivered out of order, the receiving system can be configured to guarantee that the messages will be processed in the correct order. The system can also be configured to deliver messages at least once, not more than once, or exactly once. If a message is lost, the sending system re-transmits the message until the receiving system acknowledges it receipt. For more information on WS-ReliableMessaging, see "Understanding Web Services ReliableMessaging".	SOAP

Table 3-1 (Cont.) Policy Categories

Policy Category	Description	Applies to SOAP, REST, or Both
Security	Security policies that implement the WS-Security 1.0 and 1.1 standards. They enforce message protection (message integrity and message confidentiality), and authentication and authorization of web service requesters and providers. The following token profiles are supported: username token, X.509 certificate, Kerberos ticket, and Security Assertion Markup Language (SAML) assertion. For more information about web service security tokens, see "Understanding Security Policies" and "Overview of Security Tokens".	Both A subset of security policies are supported for RESTful web services, as described in "Which OWSM Policies Are Supported for RESTful Web Services?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
SOAP Over JMS Transport	Using SOAP over JMS transport, web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits: • Reliability • Scalability • Quality of service For more information about using SOAP over JMS transport, see "Using SOAP Over JMS Transport" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.	SOAP

3.3 Overview of Building Web Service Policies Using Policy Assertions

A web service policy is comprised of one or more policy **assertions**. A policy assertion is the smallest unit of a policy that performs a specific action for the request and response operations. Assertions, like policies, belong to one of the following categories: Atomic Transactions, Configuration, Management, MTOM Attachments, Reliable Messaging, Security, SOAP Over JMS Transport, and WS-Addressing.

For more information, refer to the following topics:

- About Building Web Service Policies Using Policy Assertions
- About Defining Multiple Policy Alternatives (OR Groups)

3.3.1 About Building Web Service Policies Using Policy Assertions

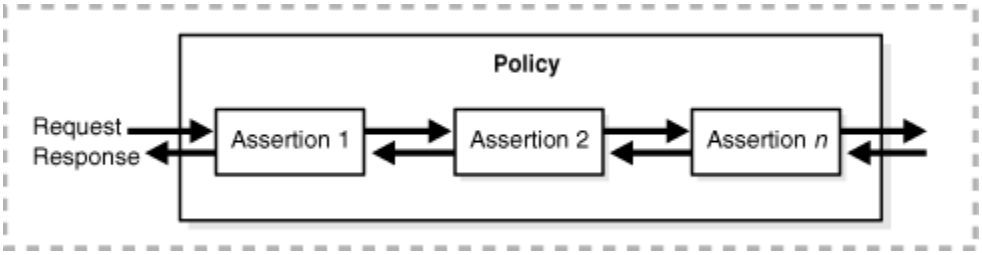
Policy assertions are chained together in a pipeline. The assertions in a policy are executed on the request message and the response message, and the same set of assertions are executed on both types of messages.

The assertions are executed in the order in which they appear in the pipeline.

> **Note:**
>
> See About Defining Multiple Policy Alternatives (OR Groups) to define multiple alternatives for policy enforcement with an **OR group**.

Figure 3-3 illustrates a typical execution flow. For the request message, Assertion 1 is executed first, followed by Assertion 2, and Assertion *n*. Although the same assertions may be executed on the response message (if a response is returned at all), the actions performed on the response message differ from the request message, and the assertions are executed on the response message in reverse order. For the response message in Figure 3-3, Assertion *n* is executed first, followed by Assertion 2, then Assertion 1.

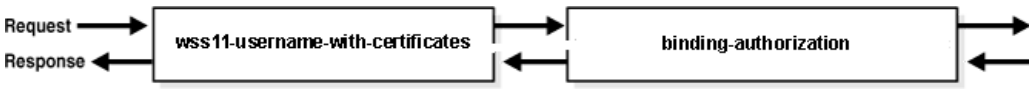
Figure 3-3 Policy Containing Assertions



For example, in Figure 3-4, the policy contains two assertions:

1. wss11-username-with-certificates—Built using the `wss11_username_token_with_message_protection_service_template`, authenticates the user based on credentials in the WS-Security UsernameToken SOAP header.
2. binding-authorization—Built using the `binding_authorization_template`, provides simple role-based authorization for the request based on the authenticated subject at the SOAP binding level.

Figure 3-4 Example Policy With Two Assertions



When the request message is sent to the web service, the assertions are executed in the order shown. When the response message is returned to the client, the same assertions are executed, but this time in reverse order. The behavior of the assertion for the request message differs from the behavior for the response message. And, in some instances, it is possible that nothing happens on the response. For example, in the example above, the authorization assertion is only executed as part of the request.

3.3.2 About Defining Multiple Policy Alternatives (OR Groups)

To define multiple alternatives for policy enforcement, you can define a set of assertions, called an **OR group**, within a service policy.

At run time, based on the assertions defined in the OR group on the service side, a client has the flexibility to choose which *one* of the assertions to enforce.

For example, if a service-side policy defines an OR group that consists of the following assertions:

- wss11-saml-with-certificates
- wss11-username-with-certificates

At run-time, the client can choose to enforce either the wss11-saml-with certificates assertion OR wss11-username-with-certificates assertion.

There is no limit to the number of assertions that can be included in an OR group. Each assertion must be valid for the policy and should support the policy requirements. For example, you should not include a log assertion in an OR group that otherwise contains security assertions and that is designed to enforce security. In this case, the log assertion would pass in the event the security assertions failed, resulting in no security.

When defining the OR group, carefully consider the order in which the assertions are added and the settings that are configured. For example, consider the following scenario:

- On the client side, you have attached the wss11_username_token_with_message_protection_client_policy policy with `Include Timestamp` enabled.
- On the service side, you have attached a custom OR group policy with two `wss11_username_token_with_message_protection_service_template` assertions defined, the first with `Include Timestamp` disabled and the second with `Include Timestamp` enabled.

In this scenario, the first assertion will get executed and the response will be sent with no timestamp. As a result, processing on the client side will fail because it is expecting a timestamp. This type of situation can occur whenever a client policy assertion expects a greater number of security requirements than the executed service policy assertion.

The following predefined service policies contain OR groups:

- `oracle/wss_saml_or_username_token_over_ssl_service_policy`—For more information, see "oracle/wss_saml_or_username_token_over_ssl_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- `oracle/wss_saml_or_username_token_service_policy`—For more information, see "oracle/wss_saml_or_username_token_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- `oracle/wss11_saml_or_username_token_with_message_protection_service_policy`—For more information, see "oracle/wss11_saml_or_username_token_with_message_protection_service_policy" in

Securing Web Services and Managing Policies with Oracle Web Services Manager.

- `oracle/multi_token_rest_service_policy`—For more information, see "oracle/multi_token_rest_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- `oracle/multi_token_over_ssl_rest_service_policy`—For more information, see "oracle/multi_token_over_ssl_rest_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.4 Understanding Policy Subjects

A *policy subject* is the target resource to which policies are attached. There are different policies for different types of resources (for example, a web service or client).

As defined in the Web Services Policy 1.5 Framework specification, at `http://www.w3.org/TR/ws-policy/`, a policy subject is an entity (for example, an endpoint, message, resource, or operation) with which a policy can be associated.

Table 3-2 lists the policy subjects to which you can attach OWSM policies. In addition, the table lists equivalent name that is used to identify the policy subject type using WLST, and the valid resource scope for each policy subject type. Resource scopes are applicable when you are creating policy sets, as described in "Overview of Global Policy Attachments Using Policy Sets". For details about how to specify resource scopes in WLST, see "Defining the Resource Scope" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Table 3-2 Policy Subjects and Resource Scopes

Policy Subject	WLST Name	Valid Resource Scope (Policy Sets)
ADF RESTful Web Service Connection	rest-connection Reserved for future use.	Reserved for future use. • Domain • Application • Application Module or Connection • Resource Path
ADF SOAP Web Service Connection	ws-connection	• Domain Name • Application Name • Application Module Name or Connection Name • Reference or Web Service Client Name • Port Name
ESS SOAP JOB Callback	job-callback	• Domain Name • Application Name • ESS Job Name
ESS SOAP JOB Invoker	job-invoke	• Domain Name • Application Name • ESS Job Name
OSB JCA Business Service	business-jca-service	• Domain Name • Application Name • Resource Path

Table 3-2 (Cont.) Policy Subjects and Resource Scopes

Policy Subject	WLST Name	Valid Resource Scope (Policy Sets)
OSB JCA Proxy Service	proxy-jca-service	• Domain Name • Application Name • Resource Path
OSB RESTful Business Service	biz-rest-service	• Domain Name • Application Name • Resource Path
OSB RESTful Proxy Service	proxy-rest-reference	• Domain Name • Application Name • Resource Path
OSB SOAP Business Service	biz-service	• Domain Name • Application Name • Resource Path
OSB SOAP Proxy Service	proxy-service	• Domain Name • Application Name • Resource Path
RESTful Client	rest-client	• Domain Name • Application Name • Application Module Name or Connection Name • Resource Path
RESTful Resource	rest-resource	• Domain Name • Application Name • Application Module Name or Connection Name • RESTful Application, Service, or Web Service Endpoint Name • Resource Path
SOA Component		
SOA JCA Reference	sca-jca-reference	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • RESTful Application, Service, or Web Service Endpoint Name
SOA JCA Service	sca-jca-service	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • Reference or Web Service Client Name
SOA RESTful Reference	sca-rest-reference	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • Reference or Web Service Client Name

Table 3-2 (Cont.) Policy Subjects and Resource Scopes

Policy Subject	WLST Name	Valid Resource Scope (Policy Sets)
SOA RESTful Service	sca-rest-service	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • RESTful Application, Service, or Web Service Endpoint Name
SOA SOAP Reference	sca-reference	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • Reference or Web Service Client Name • Port Name • Callback Interface Name
SOA SOAP Service	sca-service	• Domain Name • Application Name • SOA Partition Name • SOA Composite Name • RESTful Application, Service, or Web Service Endpoint Name • Port Name
SOAP Asynchronous Callback Client	ws-callback	• Application Name • Application Module Name or Connection Name • Callback Interface Name
SOAP Web Service	ws-service	• Application Name • Application Module Name or Connection Name • RESTful Application, Service, or Web Service Endpoint Name • Port Name
SOAP Web Service Client	ws-client	• Application Name • Application Module Name or Connection Name • Reference or Web Service Client Name • Port Name • Java EE Web Service Client EJB Name

3.5 Overview of Attaching Policies to Policy Subjects

Different types of policies and process of attaching different policies to an application is explained in the following sections.

- About Attaching Policies to Policy Subjects
- About Direct Policy Attachment

- Overview of Global Policy Attachments Using Policy Sets

3.5.1 About Attaching Policies to Policy Subjects

OWSM places a limit on the number of policies that may be attached to a subject based on the categories of the assertions that they contain. To support the attachment of policies both directly and externally (globally), OWSM determines the effective set of policies for a subject by taking into account the category of assertions within each policy, the priority of policy attachments, run-time constraints, and the status (enabled/disabled) of any policy attachments.

There are two points in the life cycle of an application in which you can attach policies: at design time and post deployment.

- At design time, you can attach OWSM policies to applications programmatically. You typically do this using your favorite IDE, such as Oracle JDeveloper. Oracle JDeveloper automates ADF and SOA client policy attachment. For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*
- Post-deployment, you can attach OWSM policies to Oracle Infrastructure web Services, RESTful web services, and Java EE web services using Oracle Enterprise Manager Fusion Middleware Control or WLST. This provides the most power and flexibility because it moves web service security to the control of the security administrator. Polices can be attached directly to an endpoint, or globally to a range of endpoints using policy sets.

For more information about effective policy calculation for an endpoint, see "How the Effective Set of Policies is Calculated" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Regardless of whether you attach a policy at design time or post-deployment, the client-side policy must be the equivalent of the one associated with the web service. If the two policy files are different, and there is a conflict in the assertions contained in the files, then the invocation of the web service operation returns an error.

For more information about attaching policies, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.5.2 About Direct Policy Attachment

After the application is deployed, you can attach OWSM policies directly to policy subjects, such as SOAP web service and client endpoints and RESTful resources and clients.

For a complete list of policy subjects to which you can attach policies, see "Understanding Policy Subjects". For details about how to attach policies directly, see "Attaching Policies Directly Using Fusion Middleware Control" and "Attaching Policies to Web Services and Clients Using WLST".

3.5.3 Overview of Global Policy Attachments Using Policy Sets

A policy set, which can contain multiple policy references, is an abstract representation that provides a means to attach policies globally to a range of endpoints of the same type, regardless of the deployment state.

You can create and manage policy sets using both Fusion Middleware Control and the WebLogic Scripting Tool (WLST).

- Understanding Global Policy Attachments Using Policy Sets
- About Subject Types and Scope of Resources
- Understanding Typical Uses for Global Policy Attachments

3.5.3.1 Understanding Global Policy Attachments Using Policy Sets

Global policy attachments (using policy sets) are supported for SOAP and RESTful-based Oracle Infrastructure and Java EE web services and clients. However, non-security policies are ignored when the effective policy set for Java EE endpoints is calculated. Global policy attachments are not supported for standalone Java EE clients.

Attaching policies globally using policy sets allows an administrator to ensure that all subjects are secured in situations where the developer, assembler, or deployer did not explicitly specify the policies to be attached. For example, if the developer did not specify policies in annotations or include policy references in deployment descriptors, then the deployer must attach them or chance a potential security risk. By attaching policies globally to a set of subjects by type, the administrator can ensure that all subjects are secured by default independent of, and even prior to, deployment. The administrator can, for example, define a policy set that attaches a security policy to all web service endpoints in a domain. In this case, any new services added to the domain automatically inherit the security configuration defined in the policy set. For more information, see "Determining the Secure Status of an Endpoint" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Policies attached globally using policy sets also provide the following:

- The ability to specify configuration overrides on a referenced policy that apply to all endpoints to which the policy set is scoped. For information about configuring overrides, see "Overriding Configuration Properties for Globally Attached Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- The ability to specify a run-time constraint that determines the context in which the policy set is relevant. For example, you can specify that a service use message protection when communicating with external clients only since the message may be transmitted over insecure public networks. However, when communicating with internal clients on a trusted network, message protection may not be required. For more information, see "Specifying Run-time Constraints in Policy Sets" *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can disable a globally attached policy for a specific endpoint or range of endpoints using predefined policies that do not enforce any behavior that are included with your Fusion Middleware installation. When you attach one of these policies to a specific endpoint, or at a lower scope, you disable the behavior of the policy that was attached globally at the higher scope. For more information, see "Disabling a Globally Attached Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Policy set definitions are stored as separate XML documents in the OWSM Repository under the `/policysets/global` directory.

3.5.3.2 About Subject Types and Scope of Resources

Table 3-2 lists the policy subjects to which you can attach OWSM policies and the valid resource scopes. For more information, see "Defining the Type and Scope of Resources for Globally Attached Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

> **Note:**
>
> When creating policy sets, the SOAP Web Service and SOAP Web Service Client subject types refer both to Oracle Infrastructure web services and clients and to Java EE web services and clients.

3.5.3.3 Understanding Typical Uses for Global Policy Attachments

Typical scenarios in which attaching policies globally can be useful include:

- All subjects of a given type need to be protected with the same set of policies, each using their default configuration. For example, all services in a domain need to be protected with authentication (using SAML or Username token) and WSS11 message protection. You can create a policy set to attach the appropriate policy to all services in the domain.
- A subset of subjects need to be protected with the same set of policies, but these policies are different from the domain-wide default. For example, all services need to be protected with authentication (using SAML or Username token), but the General Ledger application also needs stronger WSS11 message protection. You create one policy set that attaches an authentication policy to all services, and a second policy set that attaches the stronger message protection policy to the General Ledger application.
- A single subject needs to be protected by a policy in a category that is not already covered by the current set of global policy attachments and both policies need to be applied. For example, a highly-sensitive financials-based service endpoint requires permission for a client to access it in addition to the authentication and message protection required. In this case, directly attach the authorization policy to the financials-based service endpoint. The direct attachment is combined with the policies attached globally and both policies will be enforced.
- An application has been deployed with design-time policy attachments and needs to convert to using global policy attachments. The `migrateAttachments` WLST command can be used to migrate the attachments. For more information, see "Migrating Direct Policy Attachments to Global Policy Attachments" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.6 Understanding How Policies are Executed

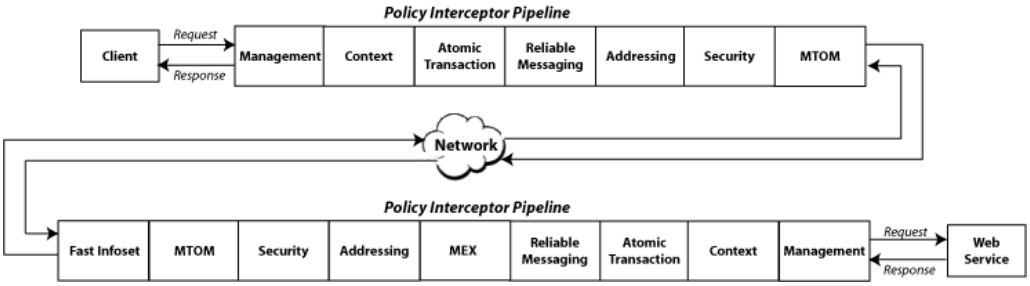
When a request is made from a service consumer (also known as a client) to a service provider (also known as a web service), the request is intercepted by one or more policy interceptors. These interceptors execute policies that are attached to the client and to the web service. There are several types of interceptors that together form a policy interceptor chain. Each interceptor executes policies of the same type. The

security interceptor intercepts and executes security policies, the MTOM interceptor intercepts and executes MTOM policies, and so on.

Policies attached to a client or web service are executed in a specific order via the Policy Interceptor Pipeline, as shown in Figure 3-5.

> **Note:**
>
> A subset of OWSM policies are supported for RESTful web services, as described in Which OWSM Policies Are Supported for RESTful Web Services? in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. REST uses only the security policy interceptor type shown in Figure 3-5.

Figure 3-5 Policy Interceptors Acting on Messages Between a Client and Web Service (SOAP)



As shown in the previous figure, when a client or a web service *initiates* a message over SOAP, whether it be a request message in the case of a client, or a response message in the case of a web service, the policies are intercepted in the following order: Management, Context (for SOAP request and response message handling), Atomic Transaction, Reliable Messaging, Addressing, Security, and MTOM. When a client or a web service *receives* a message over SOAP, that is, a request message in the case of the web service or a response message in the case of a client, the policies are executed in the reverse order and include additional interceptors: Fast Infoset, MTOM, Security, Addressing, MEX, Reliable Messaging, Atomic Transactions, Context, and Management.

A message may have one or more policies attached. Not every message will contain each type of policy. A message may contain a security policy and an MTOM policy. In this instance, the security interceptor executes the security policy, and the MTOM interceptor executes the MTOM policy. In this example, the other interceptors are not involved in processing the message.

The following describes how the policy interceptors act on messages between the client and the web service over SOAP. (Refer to Figure 3-5.)

1. The client sends a request message to a web service.
2. The policy interceptors intercept and execute the policies attached to the client. After the client policies are successfully executed, the request message is sent to the web service.

3. The request message is intercepted by policy interceptors which then execute any service policies that are attached to the web service.
4. After the service policies are successfully executed, the request message is passed to the web service. The web service executes the request message and returns a response message.
5. The response message is intercepted by the policy interceptors which execute the service policies attached to the web service. After the service policies are successfully executed, the response message is sent to the client.
6. The response message is intercepted by the policy interceptors which execute any client policies attached to the client.
7. After the client policies are successfully executed, the response message is passed to the client.

3.7 About OWSM Predefined Policies and Assertion Templates

There is a set of predefined policies and assertion templates that are automatically available when you install Oracle Fusion Middleware. The predefined policies are based on common best practice policy patterns used in customer deployments.

> **Note:**
>
> The installed predefined policies and assertion templates are read only.

You can immediately begin attaching these predefined policies to your web services or clients. You can configure the predefined policies or create a new policy by making a copy of one of the predefined policies.

Predefined policies are constructed using assertions based on predefined assertion templates. You can create new assertion templates, as required.

For more information about the predefined policies and assertion templates, see:

- "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- "Predefined Assertion Templates" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

> **Note:**
>
> WS-SecurityPolicy defines *scenarios* that describe examples of how to set up WS-SecurityPolicy policies for several security token types described in the WS-Security specification (supporting both WS-Security 1.0 and 1.1). The OWSM predefined policies support a subset of the WS-SecurityPolicy scenarios that represents the most common customer use cases.

3.8 About Overriding the Security Policy Configuration

Multiple web services or clients may use the same policy. Each may have different policy configuration requirements such as username and password.

OWSM policy configuration override enables you to update the configuration on a per service or client basis without creating new policies for each. In this way, you can create policies that define default configuration values and customize those values based on your run-time requirements.

For example, you might specify the username and password when configuring a client policy, as the information may vary from client to client.

For more information about overriding security policy configuration, see "Overriding Policy Configuration Properties" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can define whether a configuration property can be overridden when creating custom assertions, as described in "Creating Custom Assertions" in *Developing Extensible Applications for Oracle Web Services Manager*.

3.9 About Recommended Naming Conventions for Documents Created in WSM Repository

Oracle recommends that you encode as much information as possible into the name of the policy, policy set, or assertion template so that you can tell, at a glance, what the document does.

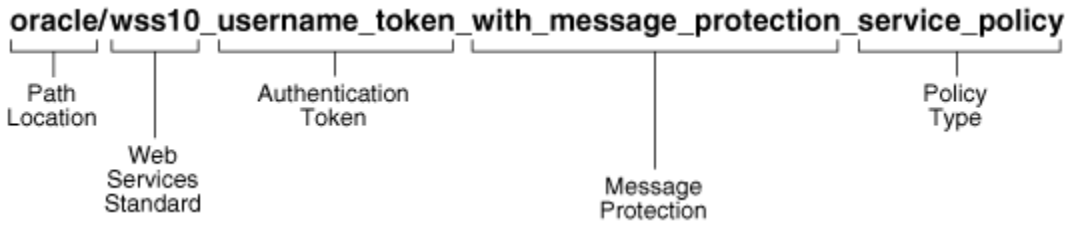
The valid characters for directory, policy, and assertion template names are:

- Uppercase and lowercase letters
- Numerals
- Currency symbol ($)
- Underscore (_)
- Hyphen (-)
- Spaces

> **Note:**
>
> The first character in the name cannot be a hyphen or space.

For example, one of the predefined security policies that is delivered with Oracle Fusion Middleware 12*c* is named `oracle/wss10_username_token_with_message_protection_service_policy`. Figure 3-6 identifies the different parts of this predefined policy name.

Figure 3-6 Identifying the Different Parts of a Policy Name



The following convention is used to name the predefined policies. The parts of the policy name are separated with an underscore character (_).

- Path Location – All policies are identified by the directory in which the policy is located. All predefined OWSM policies are in the `oracle` directory. Oracle recommends that you keep any policies that you create in a directory that is separate from the `oracle` directory in which the predefined policies are located.
- Web services Standard – If the policy uses a WS-Security standard, it is identified with wss10 (WS-Security 1.0) or wss11 (WS-Security 1.1). Or it could just be set to indicate that it is independent of WS-Security 1.0 or 1.1.
- Authentication token – If the policy authenticates users, then the type of token is specified. The predefined options include:
  - http_token – HTTP token
  - kerberos_token – Kerberos token
  - saml_token – SAML token
  - username_token – Username and password token
  - x509_token – X.509 certificate token
  - jwt_token – JWTT token
  - oauth2_token – Oauth token

  You can also define custom authentication tokens.
- Transport security – If the policy requires that the message be sent over a secure transport layer, then the token name is followed by *over_ssl,* for example, `wss_http_token_over_ssl_client_template`.
- Message protection – If the policy also provides message confidentiality and message integrity, then this is indicated using the phrase *with_message_protection* as in Figure 3-6.
- Policy Type – Indicates the type of policy or assertion template— *client* or *service*. Use the term *policy* to indicate that it is a policy, or *template* to indicate that it is an assertion template. For example, there are predefined policy and template assertions that are distinguished, as follows:

  `wss10_message_protection_service_policy`

  `wss10_message_protection_service_template`

Whatever conventions you adopt, Oracle recommends you take some time to consider how to name your policies. This will make it easier for you to keep track of your policies as your enterprise grows and you create new policies.

It is recommended that you keep any policies you create in a directory that is separate from the oracle directory where the predefined policies are located. You can organize your policies at the root level, in a directory other than oracle, or in subdirectories. For example, all of the following are valid:

- `wss10_message_protection_service_policy`
- `oracle/hq/wss10_message_protection_service_policy`
- `hq/wss10_message_protection_service_policy`

> **Note:**
>
> Use of the prefix "`oracle_`" in the policy name (for example, `oracle_wss_http_token_service_policy`) is not recommended as a best practice.

A

Web Service Security Standards

This appendix summarizes the security standards for Oracle Infrastructure Web Services.

For a complete list of standards supported for Oracle Infrastructure Web services, see "Supported Standards" in *Developing Oracle Infrastructure Web Services*.

Security Standards is explained in detail in the following topic:

- Security Standards

A.1 Security Standards

Security standards are implemented in non-XML frameworks at the transport level, and in XML frameworks at the application level.

Table A-1 lists the standards that are key to providing secure and manageable SOA environments at both the transport and application levels.

For a complete list and descriptions of standards for WebLogic Web services, see "Features and Standards Supported by WebLogic Web Services" in *Understanding WebLogic Web Services for Oracle WebLogic Server*.

Table A-1 Web Services Standards and Specification URLs

Standard	Description and Specification URL
Web Services Interoperability Organization—Basic Security Profile	Oracle considers interoperability of Web services platforms to be more important than providing support for all possible edge cases of the Web services specifications. Oracle complies with the following specification from the Web Services Interoperability Organization and considers it to be the baseline for Web services interoperability. For more information, see: • Basic Security Profile 1.0 Specification: `http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html`
Transport Layer Security —SSL	Secure Sockets Layer (SSL), also known as Transport Layer Security (TLS), is the most widely used transport-layer data-communication protocol. For more information, see: • "Understanding Transport-level and Application-level Security" • SSL 3.0: `http://tools.ietf.org/html/rfc6101`
XML Encryption (Confidentiality)	The XML encryption specification describes a process for encrypting data and representing the result in XML. For more information, see: • "About Message Encryption" • XML Encryption Syntax and Processing Specification: http://www.w3.org/TR/xmlenc-core/

Table A-1 (Cont.) Web Services Standards and Specification URLs

Standard	Description and Specification URL
XML Signature (Integrity, Authenticity)	The XML Signature specification describes signature processing rules and syntax. XML Signature binds the sender's identity (or "signing entity") to an XML document. The document is signed using the sender's private key; the signature is verified using the sender's public key. For more information, see: • "About Message Signing (XML Signature)" • XML Signature WG Specification: https://www.w3.org/Signature/
WS-Security	Web Services Security (WS-Security) specifies SOAP security extensions that provide confidentiality using XML Encryption and data integrity using XML Signature. WS-Security also includes profiles that specify how to insert different types of binary and XML security tokens in WS-Security headers for authentication and authorization purposes. For more information, see: • "Understanding Security Policies" • OASIS Web Services Security (WSS) TC Specification: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
Username Token	The username token carries basic authentication information. The `username-token` element propagates username and password information to authenticate the message. For more information, see: • "About the Username Token" • Web Services Security UsernameToken Profile 1.0 Specification: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
X.509 Certificate	An X.509 digital certificate is a signed data structure designed to send a public key to a receiving party. A certificate includes standard fields such as certificate ID, issuer's Distinguished Name (DN), validity period, owner's DN, owner's public key, and so on. For more information, see: • "About the X.509 Certificate" • Web Services Security X.509 Certificate Token Profile Specification: https://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
Kerberos Token	Kerberos token is a cross-platform authentication and single sign-on system. The Kerberos protocol provides mutual authentication between two entities relying on a shared secret (symmetric keys). For more information, see: • "About the Kerberos Token" • Web Services SecurityKerberos Token Profile 1.1 Specification: http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf
SAML Token	The Security Assertion Markup Language (SAML) is an open framework for sharing security information over the Internet through XML documents. For more information, see: • "About the SAML Token" • Web Services Security SAML Token Profile Specification: http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

Table A-1 (Cont.) Web Services Standards and Specification URLs

Standard	Description and Specification URL
WS-Policy	A Web service provider may define conditions (or policies) under which a service is to be provided. The WS-Policy framework enables one to specify policy information that can be processed by web services applications, such as Oracle WSM. For more information, see: • "Understanding Web Service Policies" • Web Services Policy 1.2 - Framework (WS-Policy) Specification: http://www.w3.org/Submission/WS-Policy/
WS-SecurityPolicy	WS-SecurityPolicy defines a set of security policy assertions used in the context of the WS-Policy framework. WS-SecurityPolicy assertions describe how messages are secured on a communication path. For more information, see: • "Understanding Security Policies" • WS-SecurityPolicy 1.2 Specification: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html
Web Services Addressing (WS-Addressing)	SOAP does not provide a standard way to specify where a message is going or how responses or faults are returned. WS-Addressing provides an XML framework for identifying web services endpoints and for securing end-to-end endpoint identification in messages. For more information, see: • "Understanding Web Services Addressing" • Web Services Addressing 1.0 - Core Specification: http://www.w3.org/TR/ws-addr-core/
WS-Trust	Defines extensions to WS-Security that provide a framework for requesting and issuing security tokens, and to broker trust relationships. WS-Trust extensions provide methods for issuing, renewing, and validating security tokens. For more information, see: • "Understanding Web Services Trust" • WS-Trust 1.3 Specification: http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html
WS-ReliableMessaging	WS-ReliableMessaging (WS-RM) defines a framework for identifying and managing the reliable delivery of messages between Web services endpoints. For more information, see: • "Using Web Services Reliable Messaging" in Oracle Fusion Middleware Developer's Guide for Oracle Infrastructure Web Services • Web Services Reliable Messaging Specification: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01-e1.html
WS-SecureConversation	The Web Services Secure Conversation Language (WS-SecureConversation) is built on top of the WS-Security and WS-Policy models to provide secure communication between services. This specification defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation For more information, see: • WS-SecureConversation 1.4 Specification: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html