Oracle® Fusion Middleware

Use Cases for Securing Web Services Using Oracle Web Services Manager

12*c* (12.2.1.1)

E71258_01

June 2016

Documentation for developers and administrators that provides use cases that demonstrate how to secure web services using Oracle Web Services Manager (OWSM).

ORACLE

Oracle Fusion Middleware Use Cases for Securing Web Services Using Oracle Web Services Manager, 12c (12.2.1.1)

E71258_01

Copyright © 2013, 2016, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

List of Figures

List of Tables

Preface

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

Audience

The Oracle Web Services Manager (OWSM) security use cases in this guide are 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*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*
- *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*
- *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.

Follow the pointers into this guide to get more information about the features and how to use them.

New and Changed Features for Release 12c (12.2.1.1)

This topic contains the New and Changed Features for Release 12*c* (12.2.1.1).

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

New and Changed Features for Release 12c (12.2.1)

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

New and Changed Features for Release 12*c* (12.1.3)

Oracle Fusion Middleware 12*c* (12.1.3) includes the following new and changed features for this document:

- A new use case has been provided that demonstrates how to secure RESTful web services using basic authentication. For more information, see "Securing RESTful Web Services Using Basic Authentication".
- A new use case has been provided that demonstrates how to propagate identify with RESTful web services. For more information, see "Propagating Security Identity with RESTful Web Services".

Other Significant Changes in this Document for Release 12*c* (12.2.1)

For 12*c* (12.1.3), the use cases contained in *Security and Administrator's Guide for Web Services*, delivered in Oracle Fusion Middleware 12*c* (12.1.2), have been moved to this guide.

1

Introduction to the Use Cases

This chapter provides an overview of the Oracle Web Services Manager (OWSM) security use cases provided in this document.

The following table summarizes the use cases described in this document.

Table 1-1 Summary of Use Cases

Solution	Description
Securing Inbound SOAP Requests Using SAML Message Protection	Secure inbound SOAP requests to: • Enforce message-level protection (that is, message integrity and message confidentiality). • Provide SAML-based authentication for inbound SOAP requests in accordance with the WS-Security 1.1 standard.
Securing RESTful Web Services Using Basic Authentication	Secure a RESTful web service using identity propagation.
Propagating Security Identity with RESTful Web Services	Propagate security identity with RESTful web services.
Configuring Federation with Microsoft ADFS 2.0 STS as the IP-STS and Oracle STS as the RP-STS	Configure web services federation with Microsoft ADFS 2.0 STS as the IP-STS and Oracle STS as the RP-STS.
Configuring Federation with Oracle STS as the IP-STS and Microsoft ADFS 2.0 STS as the RP-STS	Configure web services federation with Oracle STS as the IP-STS and Microsoft ADFS 2.0 STS as the RP-STS.
Configuring SAML HOK Using WS-Trust with OpenSSO STS	Configure SAML holder-of-key (HOK) with message protection using WS-Trust with OpenSSO STS.
Configuring SAML Sender Vouches Using WS-Trust with OpenSSO STS	Configure SAML sender vouches using WS-Trust with OpenSSO STS.
Configuring SAML Bearer Using WS-Trust with OpenSSO STS	Configure SAML bearer using WS-Trust with OpenSSO STS.

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

2

Securing Inbound SOAP Requests Using SAML Message Protection

This chapter describes how to secure inbound SOAP requests using SAML message protection.

2.1 Use Case: Securing Inbound SOAP Requests Using SAML-based Authentication

Implementing the use case requires an understanding of high-level tasks and related concepts.

The following list describes the use case, solution summary, and components involved. It also provides links to the required documentation.

Use Case
Secure inbound SOAP requests to:

- Enforce message-level protection (that is, message integrity and message confidentiality).
- Provide SAML-based authentication for inbound SOAP requests in accordance with the WS-Security 1.1 standard.

Solution
Attach an Oracle Web Services Manager (OWSM) SAML policy that is in accordance with WS-Security 1.1 to the web service and client, and configure the required keys and keystores.

Components

- Oracle Fusion Middleware
- Oracle Web Services Manager (OWSM)
- Web service and client applications to be secured

Required Documentation
To complete this use case, see the following documentation resources:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*

- *Understanding Oracle Web Services Manager*
- `keytool` Javadoc at: `http://download.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce message-level protection using SAML-based authentication for inbound SOAP requests.

  Specifically, you attach the following policies to the client and service, respectively:

  - `wss11_saml_token_with_message_protection_client_policy`
  - `wss11_saml_token_with_message_protection_service_policy`

- Configure the required keys and keystores.

Messages are protected using WS-Security's Basic 128 suite of symmetric key technologies, specifically RSA key mechanisms for message confidentiality, SHA-1 hashing algorithm for message integrity, and AES-128 bit encryption. Therefore, when you use the `keytool` (or other tool) to create the signature and encryption keys needed by this policy, you need to make sure you use the RSA key mechanism, the SHA-1 algorithm, and AES-128 bit encryption to satisfy the policy requirements for the key. For more information about supported algorithm suites, see "Supported Algorithm Suites" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.2 Securing Inbound SOAP requests using SAML Message Protection

The following sections provide additional background about the SAML message protection use case:

- "Message Protection Via Symmetric Keys"
- "What Keys Must Be in the Keystore?"
- "Multi-Domain Use Case (Keystore Hardening)"
- "When to Override the SAML Issuer"

For more information, see "Understanding Keys and Certificates" and "wss11" in *Understanding Oracle Web Services Manager*.

2.2.1 Message Protection Via Symmetric Keys

The SAML security policies in this use case use symmetric key technology.

Symmetric key cryptography relies on a single, shared secret key, as follows:

1. The client creates the symmetric key, uses it to sign and encrypt the message, and shares it with the web service in the request message.

   To protect the symmetric key, the symmetric key sent in the request message is encrypted using the web service's certificate.

2. The web service uses the symmetric key in the request message to verify the signature of the request message and decrypt it, and to then sign and encrypt the response message.

Consider the following process flow.

To create the request, the OWSM agent performs the following steps:

1. Generates the shared symmetric key and uses it to both sign and encrypt the request message.
2. Uses its own private key to "endorse" the signature of the request message.
3. Uses the web service's public key to encrypt the symmetric key.
4. Sends the symmetric key along with the request to the web service. The client sends its public key in the request so that the web service can verify the endorsement.

When the web service receives the request, it performs the following steps:

1. Decrypts the symmetric key using its private key.
2. Decrypts the request message and to verify its signature using the symmetric key.
3. Verifies the endorsement signature using the client's public key in the request message.

To send the response back to the client, the web service performs the following steps:

1. Signs the response message using the same client-generated symmetric key sent along with the request.
2. Encrypts the response message using the same client-generated symmetric key.

When the OWSM agent receives the response message, it performs the following steps:

1. Decrypts the response messages using the symmetric key it generated initially.
2. Verifies the signature of the response messages using the symmetric key it generated initially.

2.2.2 What Keys Must Be in the Keystore?

In this use case, the client and web service are in the same domain with access to the same keystore. As a result, they can share the same private/public key pair.

Specifically, the client can use the private key `orakey` to endorse the signature of the request message and the public key `orakey` to encrypt the symmetric key. The web service in turn uses the public key `orakey` to verify the endorsement, and the private key `orakey` to decrypt the symmetric key.

2.2.3 Multi-Domain Use Case (Keystore Hardening)

If the client and web service are not in the same domain and do not have access to the same keystore, the client and web service must each have a private/public key pair.

Consider the requirements in a multiple-domain use case, described in Table 2-1.

Table 2-1 Multiple-Domain Use Case Requirements

Web Service Client	Web Service
Needs its own private/public key pair in the client keystore.	Needs its own private/public key pair in the service keystore.

Table 2-1 (Cont.) Multiple-Domain Use Case Requirements

Web Service Client	Web Service
Needs the web service public key.	Needs the intermediary and root certificate corresponding to the client's public key in the keystore. These certificates will be used to verify the signature by generating a trusted certificate chain.
Generates symmetric key at run time	Needs the symmetric key, but this is sent in the request message.

For the public key the client uses to encrypt the symmetric key—that is, the public key of the web service—you have two approaches:

- 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 Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. In this case, the web service's public key does not have to be in the client's keystore.
- If the certificates is not published in the WSDL, you can specify a value for `keystore.recipient.alias` on the Configurations page, or override it on a per-client basis using the Security Configuration Details control when you attach the policy. The keystore recipient alias specifies the alias used to look up the public key in the keystore when retrieving a key for encryption of outbound SOAP messages. In this approach, the web service's public key must be in the client's keystore.

2.2.4 When to Override the SAML Issuer

The `saml.issuer.name` property of the client policy identifies the issuer of the SAML token, and defaults to a value of `www.oracle.com`. This use case uses the `www.oracle.com` default.

You can optionally specify a value for `saml.issuer.name` on the Configurations page, or override it on a per-client basis using the Security Configuration Details control when you attach the policy.

If you do use a different SAML authority (issuer) in the policy, that issuer name must be configured in the client and included in the list of possible issuers in the SAML login module. For more information, see "Adding an Additional SAML Assertion Issuer Name" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.3 Implementing SAML Message Protection

To implement SAML message protection, perform the tasks mentioned below in the sequence:

- Implementing SAML Message Protection - Prerequisites
- Creating a WebLogic Server User
- Creating a Java Keystore
- Configuring the OWSM Keystore for Securing Web Services

- Storing the Password for the Decryption Key in the Credential Store
- Attaching the Policy to Your Web Service
- Attaching the Policy to Your Web Service Client

2.3.1 Implementing SAML Message Protection - Prerequisites

Before you begin, ensure that you have performed the following tasks:

1. Download and install the following product components:
   - Oracle Fusion Middleware—includes OWSM

     For more information, see "Preparing for Oracle Fusion Middleware Installation" in *Planning an Installation of Oracle Fusion Middleware*.
   - Oracle JDeveloper

     This is required only for a subset of use cases in this document.

     For more information about locating and downloading Oracle Fusion Middleware products, see the *Oracle Fusion Middleware Download, Installation, and Configuration Readme Files* on OTN.
2. Configure a WebLogic domain.

   For the complete procedure, see "Creating a WebLogic Domain" in *Creating WebLogic Domains Using the Configuration Wizard*.
3. Start the Administration Server in the domain.

   For the complete procedure, see "Starting and Stopping Servers" in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.
4. Ensure that you can access the following administration tools:
   - Oracle Enterprise Manager Fusion Middleware Control:

     `http://localhost:7001/em`
   - Oracle WebLogic Server Administration Console

     `http://localhost:7001/console`

2.3.2 Creating a WebLogic Server User

You need to ensure that the user in the SAML token exists in the WebLogic Server identity store. If it does not, you must create it. Add a user to the identity store using the WebLogic Server Administration Console, as described in "Create users" in *Oracle WebLogic Server Administration Console Online Help*.

The web service run time extracts the SAML token from the WS-Security header and uses the name in the SAML token to validate the user against the WebLogic Server identity store. Specifically, the SAML login module verifies the SAML tokens on behalf of the web service. The SAML login module then extracts the username from the verified token and (indirectly) passes it to Oracle Platform Security Services (OPSS) to complete the authentication. For more information, see "Configuring the SAML and SAML2 Login Modules Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Any configured WebLogic Server authentication provider can then be invoked, including the default Authentication provider.

2.3.3 Creating a Java Keystore

Create a keystore and load the private key and trusted CA certificates.

The following procedure creates and manages the Java keystore using the `keytool` utility. This use case uses the JKS keystore. For the complete procedure, see "Generating Private Keys and Creating the Java Keystore" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note:

You specify an alias when you perform either of the following tasks:

- Add an entity to the keystore using the `-genkey` command to generate a key pair (public and private key).
- Add a certificate or certificate chain to the list of trusted certificates using the `-import` command.

Subsequent `keytool` commands must use this same alias to refer to the entity.

---

1. Create a new key pair and self-signed certificate.

   Use the `genkey` command to create the key pair (public and private key). `genKey` creates a new private key if one does not exist.

   The following command generates in the `default-keystore.jks` keystore an RSA key with RSA-SHA1 as the signature algorithm and alias name `orakey`. You can specify any alias name; you do not need to set the alias name to `orakey`.

   ```
   keytool -genkey -alias orakey -keyalg "RSA" -sigalg "SHA1withRSA" -dname
   "CN=test, C=US" -keystore default-keystore.jks
   ```

   The `keytool` utility prompts for the required key and keystore passwords. You need these passwords later.

2. Generate a certificate request to the certificate authority (CA).

   Use the `-certreq` command to generate the request. The CA will return a certificate or a certificate chain.

   The following command generates a certificate request for the `orakey` alias.

   ```
   keytool -certreq -alias orakey -sigalg "SHA1withRSA" -file certreq_file -
   storetype jks -keystore default-keystore.jks
   ```

3. Replace (import) the self-signed certificate with the trusted CA certificate.

   You must replace the existing self-signed certificate with the certificate returned from the CA. To do this, use the -import command. The following command replaces the trusted CA certificate in the default-keystore.jks keystore. The keytool utility prompts for the needed password.

   ```
   keytool -import -alias orakey -file certreq_file -keystore default-keystore.jks
   ```

2.3.4 Configuring the OWSM Keystore for Securing Web Services

OWSM provides support for KSS, JKS, HSM, and PKCS11 keystores.

After you create the keystores, you need to configure OWSM so that it can access and use the keystore. You can configure the OWSM keystore using the configureWSMKeystore command.

When you configure OWSM to use the JKS keystore, entries are created in the credential store for the credential map oracle.wsm.security, and any keys that you define.

Note that there is one OWSM keystore per domain, and it is shared by all web services and clients running in the domain.

To know how to configure the OWSM keystore, see "Configuring the OWSM Keystore" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.3.5 Storing the Password for the Decryption Key in the Credential Store

Store the password for the decryption key in the credential store. Use `keystore.enc.csf.key` as the key name. For the complete procedure, see "Adding Keys and User Credentials to the Credential Store" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.3.6 Attaching the Policy to Your Web Service

Attach `wss11_saml_token_with_message_protection_service_policy` to your web service and configure the policy assertion for message signing and message encryption.

For the complete procedure, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

By default, the policy signs and encrypts the entire body for the request and response. You have the option to specify individual body elements that you want to sign and encrypt. Additionally, you can specify header elements that you want to sign and encrypt. You can configure the messaging signing and encryption as desired; however, it must match the client policy settings.

For more information about configuring the policy, see "oracle/wss11_saml_token_with_message_protection_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2.3.7 Attaching the Policy to Your Web Service Client

Attach `wss11_saml_token_with_message_protection_client_policy` to your web service client and configure the policy assertion for message signing, message encryption, or both.

For the complete procedure, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

By default, the policy signs and encrypts the entire body for the request and response. You have the option to specify individual body elements that you want to sign and encrypt. Additionally, you can specify header elements that you want to sign and encrypt. You can configure the messaging signing and encryption as desired; however, it must match the service policy settings.

The `saml.issuer.name` property of the client policy identifies the issuer of the SAML token, and defaults to a value of `www.oracle.com`. This use case uses the `www.oracle.com` default. For more information about overriding the `saml.issuser.name` property, see When to Override the SAML Issuer,.

For more information about configuring the policy, see "oracle/wss11_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3

Securing RESTful Web Services Using Basic Authentication

This chapter describes how to secure a RESTful web service using basic authentication.

This chapter contains the following sections:

3.1 Use Case: Secure a RESTful Web Services Using Basic Authentication

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Secure a RESTful web service using basic authentication.

Implementation Summary
Develop a RESTful web service and secure it by attaching an Oracle Web Services Manager (OWSM) basic authentication policy.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- Oracle JDeveloper

Required Documentation
To complete this use case, see the following documentation resources:

- *Developing and Securing RESTful Web Services for Oracle WebLogic Server*
- "Developing and Securing RESTful Web Services" in *Developing Applications with Oracle JDeveloper*

This use case demonstrates the steps required to:

- Create a simple `HelloWorld` RESTful web service using JDeveloper.

- Display the name of the authenticated user in the output message using `javax.ws.rs.core.SecurityContext`.
- Package the RESTful web service with an Application subclass to define the components of a RESTful web service application deployment and provide additional metadata.
- Secure all RESTful web services, by default, by defining an OWSM global policy.
- Deploy the RESTful web service as a WAR file to WebLogic Server using the WebLogic Server Administration Console.
- Verify the `HelloWorld` web service using a browser.

3.2 Implementing the Use Case: RESTful Web Service Using Basic Authentication

To implement the use case: Secure a RESTful web service using basic authentication., complete the following steps:

- Implementing RESTful Web Service Using Basic Authentication- Prerequisites
- Securing All RESTful Resources by Default
- Creating a RESTful Web Service
- Authenticating the User Using SecurityContext
- Packaging With an Application Subclass
- Deploying the RESTful Web Service

3.2.1 Implementing RESTful Web Service Using Basic Authentication- Prerequisites

Before you begin, ensure that you have performed the following tasks:

1. Download and install the following product components:
   - Oracle Fusion Middleware—includes OWSM

     For more information, see "Preparing for Oracle Fusion Middleware Installation" in *Planning an Installation of Oracle Fusion Middleware*.
   - Oracle JDeveloper Studio Edition

     This is required only for a subset of use cases in this document.

     For more information about locating and downloading Oracle Fusion Middleware products, see the *Oracle Fusion Middleware Download, Installation, and Configuration Readme Files* on OTN.
2. Configure a WebLogic domain.

   For the complete procedure, see "Creating a WebLogic Domain" in *Creating WebLogic Domains Using the Configuration Wizard*.
3. Start the Administration Server in the domain.

   For the complete procedure, see "Starting and Stopping Servers" in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

4. Ensure that you can access the following administration tools:

   - Oracle Enterprise Manager Fusion Middleware Control:

     `http://localhost:7001/em`

   - Oracle WebLogic Server Administration Console

     `http://localhost:7001/console`

3.2.2 Securing All RESTful Resources by Default

Before you deploy RESTful resources, first define a global policy to secure all RESTful resources by default.

The following procedure defines an OWSM global policy set and assigns it to all RESTful resources. The `oracle/wss_http_token_service_policy` policy is attached to the policy configure basic authentication for all RESTful resources.

For more information about the web service WLST commands, see "Web Services WLST Custom WLST Commands" in *WLST Command Reference for Infrastructure Components*.

To secure all RESTful resources by default:

Note:

For the complete procedure, see "Attaching Policies Globally Using WLST" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

---

1. Ensure that the WebLogic Server is running.
2. Start WLST and connect to the running instance of WebLogic Server, as described in "Accessing the Web Services Custom WLST Commands" in *Administering Web Services*.

```
wls:/offline> connect('weblogic', 'welcome1', 't3://localhost:7001')
Connecting to t3://localhost:7001 with userid weblogic ...
Successfully connected to Admin Server "AdminServer" that belongs to domain
"base_domain".

Warning: An insecure protocol was used to connect to the
server. To ensure on-the-wire security, the SSL port or
Admin port should be used instead.
```

3. Start a session.

```
wls:/base_domain/serverConfig> beginWSMSession()
Location changed to domainRuntime tree. This is a read-only tree with DomainMBean
as the root.
For more help, use help('domainRuntime')

Session started for modification.
```

4. Define an OWSM global policy set for all RESTful resources.

```
wls:/base_domain/serverConfig> createWSMPolicySet('rest-policy-set','rest-
resource', 'Service("*")')

Description defaulted to "Global policy attachments for RESTful Resource
```

```
resources."
The policy set was created successfully in the session.
```

5. Attach the `oracle/wss_http_token_service_policy` policy to the policy set to require basic authentication for all RESTful resources.

```
wls:/base_domain/serverConfig> attachPolicySetPolicy('oracle/
wss_http_token_service_policy')

Policy reference "oracle/wss_http_token_service_policy" added.
```

6. Commit the session.

```
wls:/base_domain/serverConfig> commitWSMSession()

The policy set rest-policy-set is valid.
Creating policy set rest-policy-set in repository.

Session committed successfully.
```

3.2.3 Creating a RESTful Web Service

Create a simple HelloWorld RESTful web service using JDeveloper by performing the following steps:

Note:

For assistance at anytime when using JDeveloper, press **F1** or click **Help**.

---

1. Start JDeveloper.

   For the complete procedure, see "Next Steps After Installing Oracle JDeveloper Studio" in *Installing Oracle JDeveloper*.

2. Create an application and project using the Create Custom Application wizard.

   Invoke the Create Custom Application wizard by selecting **File > New > Application** and then **General > Applications > Custom Application**.

   - Application Name: **RESTfulApplication**
   - Project Name: **RESTfulService**
   - Project Features: **Java**
   - Default Package: **samples.helloworld**

   For all other values, accept the defaults.

   For the complete procedure, see "Creating Applications and Projects" in *Developing Applications with Oracle JDeveloper*.

3. Create a new Java class using the Create Java Class wizard.

   Invoke the Create Java Class wizard by right-clicking the **RESTfulService** project and selecting **New > Java Class**.

   Define the following characteristics:

   - Name: **HelloWorldResource**

- Constructors from Superclass: **Deselect**
- Implement Abstract Methods: **Deselect**

For all other values, accept the defaults.

For the complete procedure, see "How to Create a New Java Class or Interface" in *Developing Applications with Oracle JDeveloper*.

4. Add the `hello()` method to the Java class, as shown in **bold** below.

```
package samples.helloworld;
public class HelloWorldResource {
    public String hello() {
        return "Hello!";
    }
}
```

5. Create a RESTful service from the Java class using the Create RESTful Service from Java Class wizard.

   Invoke the Create RESTful Service from Java Class wizard by right-clicking **HelloWorldResource.java** and selecting **Create RESTful Service**.

   Define the following characteristics:

   - Root Path: **helloworld**
   - Configure HTTP Methods: **hello**

     - Type: **Get**
     - Produces: **text/plain**

   For all other values, accept the defaults.

   The code is updated as follows:

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("helloworld")
public class HelloWorldResource {

    @GET
    @Produces("text/plain")
    public String hello() {
        return "Hello!";
    }
}
```

For the complete procedure, see "Creating a RESTful Web Service" in *Developing Applications with Oracle JDeveloper*

3.2.4 Authenticating the User Using SecurityContext

The following procedure illustrates how to get the authenticated user using `javax.ws.rs.core.SecurityContext`.

For more information, see "Securing RESTful Web Services Using SecurityContext" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

To get the authenticated user using `SecurityContext`:

1. Access the `SecurityContext` by injecting an instance into a class field, setter method, or method parameter using the `javax.ws.rs.core.Context` annotation.

   Update the `hello()` method, created in the previous step, to print the authenticated user name obtained using the `SecurityContext`, as follows:

```
package samples.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.SecurityContext;
import javax.ws.rs.core.Context;
import java.security.Principal;

@Path("helloworld")
public class HelloWorldResource {

    @GET
    @Produces("text/plain")
    public String hello(@Context SecurityContext sc) {
        String user = "";
        if (sc != null) {
            Principal p = sc.getUserPrincipal();
            if (p != null) {
                user = p.getName();
            }
        }
        return "Hello " + user + "!";
    }
}
```

3.2.5 Packaging With an Application Subclass

The following procedure illustrates how to create a class that extends `javax.ws.rs.core.Application` to define the components of a RESTful web service application deployment and provides additional metadata. For more information, see "Packaging With an Application Subclass" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

To package the RESTful web service with an Application subclass:

1. Create a new Java class using the Create Java Class wizard.

   Invoke the Create Java Class wizard by right-clicking the **samples.helloworld** package and selecting **New > Java Class**. For assistance at anytime, press **F1** or click **Help**.

   Define the following characteristics:

   - Name: **MyApplication**
   - Package: **samples.helloworld**
   - Extends: **javax.ws.rs.core.Application**
   - Constructors from Superclass: **Deselect**

- Implement Abstract Methods: **Deselect**

For all other values, accept the defaults.

For the complete procedure, see "How to Create a New Java Class or Interface" in *Developing Applications with Oracle JDeveloper*.

2. Override the `getClasses()` method to return the list of RESTful web service resources (in this case, `HelloWorldResource`), by adding the code show in **bold** below.

```
package samples.helloworld;

import javax.ws.rs.core.Application;
import java.util.HashSet;
import java.util.Set;

public class MyApplication extends Application {
    public Set<java.lang.Class<?>> getClasses() {
           Set<java.lang.Class<?>> s = new HashSet<Class<?>>();
           s.add(HelloWorldResource.class);
           return s;
    }
}
```

3. Add the `javax.ws.rs.ApplicationPath` annotation to define the base URI pattern that gets mapped to the servlet. For more information about how this information is used in the base URI of the resource, see "What Happens at Runtime: How the Base URI is Constructed" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

```
package samples.helloworld;

import javax.ws.rs.core.Application;
import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;

@ApplicationPath("resources")
public class MyApplication extends Application {
    public Set<java.lang.Class<?>> getClasses() {
           Set<java.lang.Class<?>> s = new HashSet<Class<?>>();
           s.add(HelloWorldResource.class);
           return s;
    }
}
```

3.2.6 Deploying the RESTful Web Service

Deploy the RESTful web service application as a WAR file to WebLogic Server.

To deploy the RESTful web service:

1. Create a deployment profile for the Web application:

   a. Define the profile type and name using the Create Deployment Profile wizard.

      Invoke the Create Deployment Profile wizard by right-clicking on the RESTful Service application and selecting **Deploy > New Deployment Profile**. For assistance at anytime, press **F1** or click **Help**.

Define the following characteristics.

- Profile Type: **WAR File**

- Deployment Profile Name: **helloworld**

b. Define the context root for the Web application using the Edit WAR Deployment Profile Properties wizard.

The Edit WAR Deployment Profile Properties wizard is invoked automatically when you click **OK** in the Create Deployment Profile wizard. For assistance at anytime, press **F1** or click **Help**.

Define the following characteristics:

- Specify Java EE Web Context Root: **restservice**

2. Deploy the web application with the following characteristics using the Deploy <application> wizard.

Invoke the Deploy <application> wizard by right-clicking the **RESTfulService** application and selecting **Deploy > helloworld**. For assistance at anytime, press **F1** or click **Help**.

Define the following characteristics:

- Deployment Action: **Deploy to WAR**

3. View the WAR file in your configured project directory. For example:

```
c:\JDeveloper\mywork\RESTfulApplication\RESTfulService\deploy\helloworld.war
```

4. Deploy the WAR file on WebLogic Server. For more information, see "Deploy applications and modules" in *Oracle WebLogic Server Administration Console Online Help*.

3.3 Verifying the Use Case: RESTful Web Service

You can access RESTful web service from a browser.

To access the RESTful web service in a browser, enter the following URL in a browser to test the RESTful web service:

```
http://<host>:<port>/restservice/resources/helloworld
```

For example, `http://localhost:7001/restservice/resources/helloworld`.

Enter the WebLogic Server username and password when prompted. For example, **weblogic** and **welcome1**.

The following message is returned in the browser:

```
Hello weblogic!
```

You can test basic and advanced features of your web service using the Web Services Test Client or Test Web Service page in Fusion Middleware Control. For more information, see "Testing Web Services" in *Administering Web Services*.

3.4 Additional Resources for RESTful Web Services Use Case

Refer to the following resources for more information about developing and securing RESTful web services and clients:

- Build RESTful web services with JAX-RS sample, as described in "Java EE 6 Examples" in *Understanding Oracle WebLogic Server*.
- *Developing and Securing RESTful Web Services for Oracle WebLogic Server*
- "Developing and Securing RESTful Web Services" in *Developing Applications with Oracle JDeveloper*

4

Propagating Security Identity with RESTful Web Services

This chapter describes how to propagate security identity with RESTful web services.

This chapter contains the following sections:

4.1 Use Case: Propagate Security Identity with RESTful Web Services

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Propagate security identity with RESTful web services. For example, if a user is trying to access a web portal via the browser and is prompted to enter credentials, then these credentials may be propagated to a back-end service that the web portal needs to access to complete the user request.

Implementation Summary
Develop a RESTful web service and client and secure them using Oracle Web Services Manager (OWSM) SAML policy.

Components

- Oracle Fusion Middleware—includes Oracle Web Services Manager (OWSM)
- Oracle JDeveloper

Required Documentation
To complete this use case, see the following documentation resources:

- *Developing and Securing RESTful Web Services for Oracle WebLogic Server*
- "Developing and Securing RESTful Web Services" in *Developing Applications with Oracle JDeveloper*

This use case demonstrates the steps required to:

- Create a simple `HelloWorld` RESTful web service using JDeveloper.

- Display the name of the authenticated user in the output message using `javax.ws.rs.core.SecurityContext`.
- Deploy the RESTful web service as a WAR file to WebLogic Server.
- Test the `HelloWorld` RESTful web service.
- Build and secure a RESTful client proxy for the RESTful web service using JDeveloper.
- Set up the keystore and certificates required for SAML security.
- Verify the RESTful client proxy.

4.2 Use Case: Propagating Security Identity with RESTful Web Services

This use case comprises the following tasks:

- Propagating Security Identity with RESTful Web Services - Prerequisites
- Create, Secure, and Deploy a RESTful Web Service
  - Creating a RESTful Web Service
  - Authenticating the User Using SecurityContext
  - Modifying the Servlet Name for the Web Project
  - Securing the RESTful Web Service
  - Deploying the RESTful Web Service
- Create, Secure, and Deploy a RESTful Client
  - Creating a RESTful Client
  - Modifying the HTTP Servlet to Call the RESTful Client
  - Securing the Servlet Web Application
  - Creating a weblogic.xml Deployment Descriptor
  - Deploying the RESTful Client
- Creating a Test User
- Set Up the Keystore Service (KSS)

4.2.1 Propagating Security Identity with RESTful Web Services - Prerequisites

Before you begin, ensure that you have performed the following tasks:

1. Download and install the following product components:
   - Oracle Fusion Middleware—includes OWSM

     For more information, see "Preparing for Oracle Fusion Middleware Installation" in *Planning an Installation of Oracle Fusion Middleware*.
   - Oracle JDeveloper

     This is required only for a subset of use cases in this document.

For more information about locating and downloading Oracle Fusion Middleware products, see the *Oracle Fusion Middleware Download, Installation, and Configuration Readme Files* on OTN.

2. Configure a WebLogic domain.

   For the complete procedure, see "Creating a WebLogic Domain" in *Creating WebLogic Domains Using the Configuration Wizard*.

3. Start the Administration Server in the domain.

   For the complete procedure, see "Starting and Stopping Servers" in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

4. Ensure that you can access the following administration tools:

   - Oracle Enterprise Manager Fusion Middleware Control:

     `http://localhost:7001/em`

   - Oracle WebLogic Server Administration Console

     `http://localhost:7001/console`

4.2.2 Create, Secure, and Deploy a RESTful Web Service

Perform the following tasks to create, secure and deploy a RESTful web service:

- Creating a RESTful Web Service
- Authenticating the User Using SecurityContext
- Modifying the Servlet Name for the Web Project
- Securing the RESTful Web Service
- Deploying the RESTful Web Service

4.2.2.1 Creating a RESTful Web Service

To create a simple helloworld RESTful web service using JDeveloper:

Note:

For assistance at anytime when using JDeveloper, press **F1** or click **Help**.

---

1. Start JDeveloper.

   For the complete procedure, see "Next Steps After Installing Oracle JDeveloper Studio" in *Installing Oracle JDeveloper*.

2. Create an application and project using the Java Desktop Application wizard.

   Invoke the Java Desktop Application wizard by selecting **File > New > Application** and then **General > Applications > Java Desktop Application**.

   Define the following characteristics:

   - Application Name: **rest-saml-idprop**

- Application Package Prefix: **examples.wsm.helloworld**
- Project Name: **service**
- Default Package: **examples.wsm.helloworld**

For all other values, use the defaults.

For the complete procedure, see "Creating Applications and Projects" in *Developing Applications with Oracle JDeveloper*.

3. Create a new Java class under the `service` project using the Create Java Class wizard.

   Invoke the Create Java Class wizard by right-clicking the **service** project and selecting **New > Java Class**.

   Define the following characteristics:

   - Name: **HelloWorldIdPropSample**
   - Package: **examples.wsm.helloworld**

   For all other values, use the defaults.

   The `HelloWorldIdPropSample.java` file is created and opened in JDeveloper.

   For the complete procedure, see "How to Create a New Java Class or Interface" in *Developing Applications with Oracle JDeveloper*.

4. Add the `hello()` method to the Java class, as shown in **bold** below.

```
package examples.wsm.helloworld;

public class HelloWorldIdPropSample {
    public HelloWorldIdPropSample() {
        super();
    }

    public String hello() {
        return "Hello";
    }
}
```

5. Create a RESTful service from the Java class using the Create RESTful Service from Java Class wizard.

   Invoke the Create RESTful Service from Java Class wizard by right-clicking **HelloWorldIdPropSample.java** and selecting **Create RESTful Service**.

   Define the following characteristics:

   - Platform: JAX-RS 1.x Style
   - Root Path: helloworld
   - Configure HTTP Methods: hello
     - Method: GET
     - Produces: text/plain
     - Path: user

For the complete procedure, see "Creating a RESTful Web Service" in *Developing Applications with Oracle JDeveloper*.

The code is updated as follows:

```
package examples.wsm.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("helloworld")
public class HelloWorldIdPropSample {
    public HelloWorldIdPropSample() {
        super();
    }

    @GET
    @Produces("text/plain")
    @Path("user")
    public String hello() {
        return "Hello";
    }
}
```

4.2.2.2 Authenticating the User Using SecurityContext

The following procedure illustrates how to get the authenticated user using `javax.ws.rs.core.SecurityContext`.

For more information, see "Securing RESTful Web Services Using SecurityContext" in *Developing and Securing RESTful Web Services for Oracle WebLogic Server*.

To get the authenticated user using `SecurityContext`:

- Access the `SecurityContext` by injecting an instance into a class field, setter method, or method parameter using the `javax.ws.rs.core.Context` annotation.

  Update the `hello()` method, created in the previous step, to print the authenticated user name obtained using the `SecurityContext`, as follows:

```
package examples.wsm.helloworld;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.SecurityContext;
import javax.ws.rs.core.Context;
import java.security.Principal;

@Path("helloworld")
public class HelloWorldIdPropSample {
    public HelloWorldIdPropSample() {
        super();
    }

    @GET
    @Produces("text/plain")
    @Path("user")
    public String hello(@Context SecurityContext sc) {
        String user = "No user";
        if (sc != null) {
```

```
                Principal p = sc.getUserPrincipal();
                if (p != null) {
                    user = p.getName();
                }
            }
            return "Hello " + user;
        }
    }
```

4.2.2.3 Modifying the Servlet Name for the Web Project

When you created the RESTful web service using the Create RESTful Service from Java Class wizard, as described in "Creating a RESTful Web Service", JDeveloper automatically changed the project to a web project and added the `web.xml` file. By default, the servlet name for the web project is `jersey`.

Edit the `web.xml` file located in the `Web Content/WEB-INF` folder to specify a more user-friendly name, such as **helloworld**. For example:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">
  <servlet>
    <servlet-name>helloworld</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>helloworld</servlet-name>
    <url-pattern>/resources/*</url-pattern>
  </servlet-mapping>
</web-app>
```

4.2.2.4 Securing the RESTful Web Service

To secure RESTful web services, you can attach one of the OWSM predefined security policies described in "Which OWSM Policies Are Supported for RESTful Web Services?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Secure the RESTful web service by attaching the following policy using the Policy wizard: `oracle/multi_token_rest_service_policy`

Invoke the Policy wizard by right-clicking on the `web.xml` file and selecting **Secure RESTful Application**.

The security policy configuration is saved to the `wsm-assembly.xml` deployment descriptor file, shown below, in the `Web Content/WEB-INF` folder. If the `wsm-assembly.xml` file does not exist, it will be created.

```
<orawsp:wsm-assembly xmlns:orawsp="http://schemas.oracle.com/ws/2006/01/policy">
  <sca11:policySet xmlns:sca11="http://docs.oasis-open.org/ns/opencsa/sca/200912"
name="policySet"
                appliesTo="REST-RESOURCE()" attachTo="SERVICE('helloworld')"
orawsp:highId="1"
                xml:id="REST-RESOURCE__SERVICE__helloworld__">
    <wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"
          DigestAlgorithm="http://www.w3.org/ns/ws-policy/Sha1Exc"
```

```
          URI="oracle/multi_token_rest_service_policy" orawsp:status="enabled"
orawsp:id="1"/>
  </sca11:policySet>
</orawsp:wsm-assembly>
```

For the complete procedure, see "Attaching Policies to RESTful Services" in *Developing Applications with Oracle JDeveloper*.

4.2.2.5 Deploying the RESTful Web Service

Deploy the RESTful web service application as a WAR file to WebLogic Server.

To deploy the RESTful web service:

1. Create a deployment profile for the Web application:

   a. Define the profile type and name using the Create Deployment Profile wizard:

      Invoke the Create Deployment Profile wizard by right-clicking on the **service** project and selecting **Deploy > New Deployment Profile**.

      Define the following characteristics:

      - Profile Type: **WAR File**

      - Deployment Profile Name: **helloworld**

   b. Define the context root for the Web application using the Edit WAR Deployment Profile Properties wizard.

      The Edit WAR Deployment Profile Properties wizard is invoked automatically when you click **OK** in the Create Deployment Profile wizard.

      Define the following characteristics:

      - Specify Java EE Web Context Root: **rest-saml-idprop**

2. Deploy the web application using the Deploy <application> wizard.

   Invoke the Deploy <application> wizard by right-clicking the **service** application and selecting **Deploy > helloworld**.

   Define the following characteristics:

   - Deployment Action: **Deploy to WAR**

3. View the WAR file in your configured project directory. For example:

   ```
   c:\JDeveloper\mywork\rest-saml-idprop\service\deploy\helloworld.war
   ```

4. Ensure that you have started WebLogic Server to which you want to deploy the RESTful web service.

   Invoke Fusion Middleware Control and deploy the WAR file.

   ```
   http://localhost:7001/em
   ```

5. Deploy the WAR file using the Deploy Java EE Application Assistant.

   Access the Deploy Java EE Application Assistant, by selecting **WebLogic Domain > *domainname* > AdminServer** in the navigation pane, selecting **WebLogic Server > Deployments** in the content pane, and clicking **Deploy**.

For more information, see "Deploying Java EE Applications" in *Administering Oracle Fusion Middleware*.

4.2.2.6 Testing the RESTful Web Service Using Fusion Middleware Control

Test the RESTful web service application using Fusion Middleware Control.

To test the RESTful web service:

1. Invoke Fusion Middleware Control.

   ```
   http://localhost:7001/em
   ```

2. View the summary page for the RESTful web service application.

   a. In the navigation pane, expand the Application Deployments folder to expose the applications in the domain, expand the application deployment, and select the **helloworld (AdminServer)** application name.

   b. In the content pane, select **Application Deployment**, then **Web Services**.

   c. In the Web Service Details section of the page, click the **RESTful Services** tab and click the application name **helloworld** to navigate to the RESTful Service Application page.

   For the complete procedure, see "Viewing the Details for a RESTful Service Application" in *Administering Web Services*.

3. Click **Test RESTful Service**.

   The RESTful web service application WADL file is parsed automatically. By default, the `GET(hello)` method is selected (since this is the only method available in the application).

4. Configure the test client:

   a. On the Request tab, select **OWSM Security Policies**.

   b. Select **oracle/wss_http_token_client_policy** in the Client Policies list.

   c. Enter **weblogic** and **welcome1** in the Configuration Properties Username and Password field.

5. Click **Test Web Service**.

The following information is returned on the Response tab:

```
Hello weblogic
```

For more information, see "Testing Web Services" in *Administering Web Services*.

4.2.3 Create, Secure, and Deploy a RESTful Client

Perform the following tasks to create and secure a RESTful client:

- Creating a RESTful Client
- Modifying the HTTP Servlet to Call the RESTful Client
- Securing the Servlet Web Application
- Creating a weblogic.xml Deployment Descriptor

- Deploying the RESTful Client

4.2.3.1 Creating a RESTful Client

To create a simple RESTful client using JDeveloper:

1. Create a new web project using the Create Web Project wizard.

   Invoke the Java Desktop Application wizard by selecting **File > New > Project** and then **Web Project**.

   Define the following characteristics:

   - Location

     - Project Name: **rest-client**

   - Web Application

     - Servlet 3.0/JSP 2.2 (Java EE 6)

   - Web Project Profile

     - Java EE Web Application Name: **rest-saml-idprop-client**

     - Java EE Context Root: **rest-saml-idprop-client**

   For all other values, use the defaults.

   For the complete procedure, see "Creating Applications and Projects" in *Developing Applications with Oracle JDeveloper*.

2. Create an HTTP servlet that will serve as the RESTful client using the Create HTTP Servlet wizard.

   Invoke the Create HTTP Servlet wizard by right-clicking the **rest-client** project and selecting **New > From Gallery** and then **Web Tier > Servlets > HTTP Servlet**.

   Define the following characteristics:

   - Servlet Information

     - Class: **HelloWorldServlet**

   - Servlet Mapping

     - URL Pattern: **/hellorestclient**

   For all other values, use the defaults.

   The `HelloWorldServlet.java` file is created within the project directory and opened automatically in JDeveloper.

   For the complete procedure, see "How to Generate an HTTP Servlet" in *Developing Applications with Oracle JDeveloper*.

3. Create a RESTful client proxy using the Create RESTful Client and Proxy wizard.

   Invoke the Create RESTful Client and Proxy wizard by right-clicking the **rest-client** project and selecting **New > From Gallery** and then **Business Tier > Web Services > RESTful Client and Proxy**.

   Define the following characteristics:

   - Select Deployment Platform

- Jersey 1.x Style

- Select WADL

  - URL: **http://localhost:7001/rest-saml-idprop/resources/application.wadl**

- Customize Proxy Names

  - Class Name: **HelloWorldRestClient**

- Client Policy Configuration

  - Security Policy: **oracle/http_saml20_token_bearer_client_policy**

For all other values, use the defaults.

For the complete procedure, see "How to Create RESTful Web Service Clients" in *Developing Applications with Oracle JDeveloper*.

4.2.3.2 Modifying the HTTP Servlet to Call the RESTful Client

Modify the `HelloWorldServlet` HTTP servlet to call the RESTful client, as shown in bold below:

```
package examples.wsm.helloworld;
...
import com.sun.jersey.api.client.Client;
import examples.wsm.helloworld.HelloWorldRestClient.Helloworld;
...
   public void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
       response.setContentType(CONTENT_TYPE);
       Client client = HelloWorldRestClient.createClient();
       HelloWorldRestClient.Helloworld
             hello = HelloWorldRestClient.helloworld(client);
       String output = hello.user().getAsTextPlain(String.class);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloWorldServlet</title></head>");
        out.println("<body>");
        out.println("<p>The servlet has received a GET. This is the reply.</p>");
        out.println("<p>Output from RESTful service:" + output + "</p>");
        out.println("</body></html>");
       out.close();
   }
}
```

4.2.3.3 Securing the Servlet Web Application

Secure the servlet web application by editing the `web.xml` file for the rest-client project, located in the `Web Content/WEB-INF` folder, as follows:

1. Under Servlets, add an entry for the HelloWorldServlet as follows:

   - Name: **HelloWorldServlet**

   - Type: **Servlet Class**

   - Servlet Class/JSP File: **examples.wsm.helloworld.HelloWorldServlet**

2. Under Security, configure the following values:

   - Login Authentication

- **Http Basic Authentication**

- Security Roles

  - **webuser**

- Security Constraints: Web Resource Collection

  - Web Resource Name: **Success**

  - Applies to: **All HTTP Methods**

  - URL Patterns: **/hellorestclient**

- Security Constraints: Authorization

  - Authorize: **Enabled**

  - Security Role: **webuser**

The `web.xml` is updated as follows:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">
  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>examples.wsm.helloworld.HelloWorldServlet</servlet-class>
  </servlet>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Success</web-resource-name>
      <url-pattern>/hellorestclient</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>webuser</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
  </login-config>
  <security-role>
    <role-name>webuser</role-name>
  </security-role>
</web-app>
```

4.2.3.4 Creating a weblogic.xml Deployment Descriptor

To create a `weblogic.xml` deployment descriptor:

1. Create a `weblogic.xml` deployment descriptor using the Create WebLogic Deployment Descriptor wizard.

   Invoke the Create WebLogic Deployment Descriptor wizard by right-clicking the **rest-client** project and selecting **New > From Gallery**, and then **General > Deployment Descriptors > WebLogic Deployment Descriptor**.

   Define the following characteristics:

   - Select Descriptor

- Descriptor: **weblogic.xml**

- Select Version

  - Version: **12.2.1**

The `weblogic.xml` file is created in the `WebContent/WEB-INF` folder and opened automatically in JDeveloper.

2. Under Security, configure the following values:

   - Run-As Role Assignments

     - Role Name: **webuser**

     - Principals: **weblogic**

The `weblogic.xml` file is created, as follows:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-
app http://xmlns.oracle.com/weblogic/weblogic-web-app/1.5/weblogic-web-app.xsd"
                  xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app">
  <security-role-assignment>
    <role-name>webuser</role-name>
    <principal-name>weblogic</principal-name>
  </security-role-assignment>
</weblogic-web-app>
```

4.2.3.5 Deploying the RESTful Client

Deploy the RESTful client application as a WAR file to WebLogic Server.

To deploy the client:

1. Create a deployment profile for the Web application:

   a. Define the profile type and name using the Create Deployment Profile wizard.

      Invoke the Create Deployment Profile wizard by right-clicking on the **rest-client** project and selecting **Deploy > New Deployment Profile**.

      Define the following characteristics:

      - Profile Type: **WAR File**

      - Deployment Profile Name: **helloworld-restclient**

   b. Define the context root for the Web application using the Edit WAR Deployment Profile Properties wizard.

      The Edit WAR Deployment Profile Properties wizard is invoked automatically when you click **OK** in the Create Deployment Profile wizard.

      Define the following characteristics:

      - General: Specify Java EE Web Context Root: **rest-saml-idprop-client**

2. Deploy the web application using the Deploy <application> wizard:

   Invoke the Deploy <application> wizard by right-clicking the **rest-client** application and selecting **Deploy > helloworld-restclient**.

   Define the following characteristics:

- Deployment Action: **Deploy to WAR**

3. View the WAR file in your configured project directory. For example:

   ```
   c:\JDeveloper\mywork\rest-saml-idprop\rest-client\deploy\helloworld-
   restclient.war
   ```

4. Invoke Fusion Middleware Control and deploy the WAR file.

   ```
   http://localhost:7001/em
   ```

   For more information, see "Deploying Java EE Applications" in *Administering Oracle Fusion Middleware*.

4.2.3.6 Testing Access to the RESTful Client

Until the keystore service (KSS) is configured, as described in the next step, "Set Up the Keystore Service (KSS)", access to the RESTful web service client will fail.

To access the RESTful web service client in a browser, enter the following URL in a browser to test the RESTful web service:

```
http://<host>:<port>/rest-saml-idprop-client/hellorestclient
```

Enter the WebLogic Server username and password when prompted. For example, **weblogic** and **welcome1**.

Note that the following error is returned: `Error 500--Internal Server Error.`

4.2.4 Set Up the Keystore Service (KSS)

OWSM uses public key cryptography to sign the SAML bearer token and requires you to set up a keystore.

Keys and the keystore provide the basis for configuring message protection.

4.2.4.1 Why Use KSS?

KSS is a service provided by Oracle Platform Security Services (OPSS).

KSS offers the following benefits over JKS:

- Integrated tooling

  - Use Fusion Middleware Control or WLST to perform CRUD operations on KSS keys and certificates.

  - Internal CA for generating CA-signed keys and certificates.

- Improved lifecycle management

  - Ability for multiple domains to share the same keystore is simplified with centralized storage (for example, database storage).

  - Ability to segregate keystores (for example, OWSM can have its own keystore via the concept of a "stripe".

  - Simplified management as passwords are not required for accessing private keys in the keystore.

4.2.4.2 Setting Up the Keystore Services

To set up the KSS keystore:

1. Invoke Fusion Middleware Control.

   `http://localhost:7001/em`

2. Create a keystore from the Keystore page.

   To navigate to the Keystore page, select **WebLogic Domain > Security > Keystore**.

   a. Click **Create Stripe** and define the following characteristics:

      - Stripe Name: **owsm**

   b. Select owsm in the list, and click **Create Keystore** and define the following characteristics:

      - Keystore Name: **keystore**

      - Protection: **Policy**

      - Grant Permission: **Disabled**

   For the complete procedure, see "Using the OPSS Keystore Service for Message Protection" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Generate a key-pair using the Generate Keypair dialog.

   To navigate to the Generate Keypair dialog, select **owsm** > **keystore** on the Keystore page, click **Manage**, and click **Generate Keypair**.

   Define the following characteristics:

   - Alias: **orakey**
   - Common Name: **orakey**
   - Organization Unit: **us**
   - Country: **United States**
   - RSA Key Size: **1024**

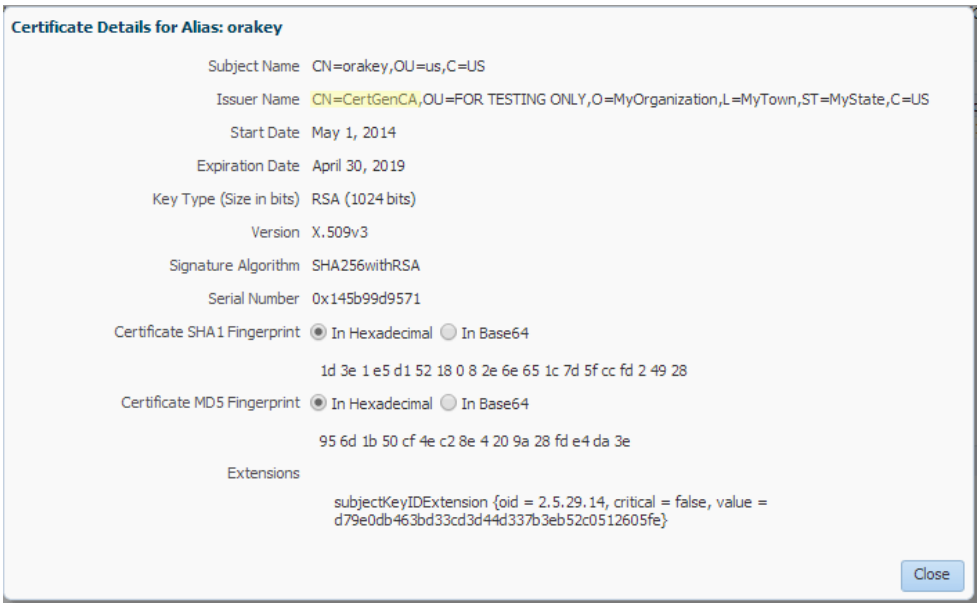
   For the complete procedure, see "Using the OPSS Keystore Service for Message Protection" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

4. Import the `democa` CA certificate into the `owsm` stripe.

   By default, the keypair is signed by the `democa` CA that ships with KSS.

   To view the certificate in use, select **owsm** > **keystore** on the Keystore page, click **Manage**, and click the **orakey** alias to display the Certificate Details for Alias: orakey dialog.

Figure 4-1 Certificate Details for Alias: orakey Dialog



Validation of the certificate on the service side will fail until you import the CA into the `owsm` keystore, as the OWSM Agent is unable to validate the certificate path for the signing certificate.

Export the `democa` CA certificate from the `castore` keystore in the `system` stripe and import it into the `orakey` keystore in the `owsm` stripe.

a. To export the `democa` CA certificate, select **system** > **castore** on the Keystore page, click **Manage**, select the **democa** alias, and click **Export**.

In the Certificate dialog, click Export Certificate to save it to a local file (or copy and paste the contents into a file of your choice).

For the complete procedure, see "Exporting a Certificate or Trusted Certificate with Fusion Middleware Control" in *Securing Applications with Oracle Platform Security Services*.

b. To import the `democa` CA certificate, select **owsm** > **keystore** on the Keystore page, click **Manage**, and click **Import**.

Define the following characteristics:

- Certificate Type: **Trusted Certificate**

- Alias: **democa**

- Select a file that contains the Certificate or Certificate Chain: **Enabled**

Click **Choose File,** navigate to the exported certificate file, click **Open**, and click **OK** to import the certificate.

For the complete procedure, see "Importing a Certificate or Trusted Certificate with Fusion Middleware Control" in *Securing Applications with Oracle Platform Security Services*.

4.2.5 Creating a Test User

To create a test user:

1. Invoke the WebLogic Server Administration Console. For example:

   ```
   http://localhost:7001/console
   ```

   For the complete procedure, see "Starting the Administration Console" in *Oracle WebLogic Server Administration Console Online Help*.

2. Create a user named **testuser**, with the password **welcome1**.

   For the complete procedure, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*.

3. In JDeveloper, edit the `weblogic.xml` file for the `rest-client` project, located in the `Web Content/WEB-INF` folder, to map `testuser` to the `webuser` role.

   Under Security > Security Role Assignments, select **webuser** and add **testuser** as a valid principal.

4.3 Verifying the Use Case: Propagating Security Identity with RESTful Web Services

To access the RESTful web service client in a browser, enter the following URL in a browser to test the RESTful web service:

```
http://<host>:<port>/rest-saml-idprop-client/hellorestclient
```

Enter the WebLogic Server username and password when prompted. For example, **weblogic** and **welcome1** or **testuser** and **welcome1**.

The following message is returned in the browser:

```
The servlet has received a GET. This is the reply.
Output from RESTful service: Hello testuser
```

You can test basic and advanced features of your web service using the Web Services Test Client or Test Web Service page in Fusion Middleware Control. For more information, see "Testing Web Services" in *Administering Web Services*.

5

Configuring Federation with Microsoft ADFS 2.0 STS as the IP-STS and Oracle STS as the RP-STS

This chapter describes how to configure web services federation with Microsoft ADFS 2.0 STS as the Identity Provided STS (IP-STS) and Oracle STS as the Replying Party (RP-STS).

This chapter contains the following sections:

5.1 Use Case: Microsoft ADFS 2.0 STS as IP-STS and Oracle STS as RP-STS

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Configure web services federation with Microsoft ADFS 2.0 STS as the IP-STS and Oracle STS as the RP-STS.

Solution
Attach Oracle Web Services Manager (OWSM) WS-Trust policies to the web service and client, and configure Oracle STS and Microsoft ADFS 2.0 STS to establish trust across security domains.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- Oracle STS
- Microsoft ADFS 2.0 STS
- Web service and client applications to be secured

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce message-level protection using SAML bearer authentication.

  Specifically, you attach the following policies to the client and service, respectively:

  – `oracle/wss_sts_issued_saml_bearer_token_over_ssl_client_policy` and policies based on `oracle/sts_trust_config_client_template`

  – `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`

- Configure web services federation using Microsoft ADFS 2.0 STS as the IP-STS and Oracle STS is used as the RP-STS.

Transport security with SSL is used to protect the service, the RP-STS, and IP-STS.

5.2 Use Case: Implementing Web Services federation with Microsoft ADFS2.0 STS

To implement the use case, complete the following tasks:

- Configuring the Web Service
- Configuring Oracle STS as the RP-STS
- Configuring Microsoft ADFS 2.0 STS as the IP-STS
- Configuring the Web Service Client

Note:

In the following sections, high-level configuration steps for Oracle STS and Microsoft ADFS 2.0 STS are provided. For detailed information about how to perform these configuration steps, refer to the documentation for the particular STS:

- For Oracle STS: `http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oraclests-166231.html`
- For Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

5.2.1 Configuring the Web Service

To configure the web service:

1. Attach the `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy` policy to the web service. For the complete procedure, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
2. Import the signing certificate for the Oracle STS `/wssbearer` endpoint into the OWSM keystore.
3. Define the Oracle STS endpoint as a trusted issuer and a trusted DN. For the complete procedure, see "Defining Trusted Issuers and Trusted Distinguished

Names List for SAML Signing Certificates" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

5.2.2 Configuring Oracle STS as the RP-STS

To configure Oracle STS as the RP-STS, perform the following steps.

For the complete procedure, see the Oracle STS documentation at `http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oraclests-166231.html`.

1. Configure WebLogic Server to enable one-way SSL on port `14101`.
2. Configure the Oracle STS `/wssbearer` endpoint as follows:
   - Attach the policy with the URI `sts/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`.
   - Create an `OWSM LRG SAML Validation` validation template to validate the incoming SAML token and apply it to the endpoint.
3. Add the service as a replying party partner in Oracle STS.
4. Add the Microsoft ADFS 2.0 STS instance acting as the IP-STS as a trusted identity provider:
   a. Configure an issuing authority partner profile for the Microsoft ADFS 2.0 STS instance.
   b. Add the Microsoft ADFS 2.0 STS instance as an issuing authority partner, giving as the partner name the issuer of the SAML assertion for the instance.
   c. Import the signing certificate for the Microsoft ADFS 2.0 STS instance into the OWSM keystore.

5.2.3 Configuring Microsoft ADFS 2.0 STS as the IP-STS

To configure Microsoft ADFS 2.0 STS as the IP-STS, perform the following steps.

For the complete procedure, see the Microsoft ADFS 2.0 STS documentation at `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`.)

1. Confirm that the `/usernamemixed` endpoint is enabled.
2. Add the Oracle STS instance acting as the IP-STS as a relying party using the ADFS 2.0 management console.
3. Configure ADFS 2.0 STS to issue SAML bearer tokens for the RP-STS.

5.2.4 Configuring the Web Service Client

To configure the web service client:

1. Attach the policy `oracle/wss_sts_issued_saml_bearer_token_over_ssl_client_policy` and configure it to refer to the web service. For the complete procedure, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Additionally, set `sts.in.order` to the URI of the Oracle STS endpoint followed by the ADFS 2.0 STS endpoint. For example:

```
http://m2.example.com:14100/sts/wssbearer;
http://http://m1.example.com/adfs/services/trust/13/usernamemixed
```

2. Create a policy from `oracle/sts_trust_config_client_template`, modify it as follows, and attach it to the client:

   - Set Port URI to the ADFS 2.0 STS endpoint. For example:

     ```
     http://m1.example.com/adfs/services/trust/13/usernamemixed
     ```

   - Set Client Policy URI `oracle/wss_sts_issued_saml_bearer_token_over_ssl_client_policy`.

   For the complete procedure, see "Creating and Editing Web Service Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Create a policy from `oracle/sts_trust_config_client_template`, modify it as follows, and attach it to the client:

   - Set Port URI to the Oracle STS endpoint. For example:

     ```
     http://m2.example.com:14100/sts/wssbearer
     ```

   - Set WSDL Uri to the Oracle STS endpoint. For example:

     ```
     http://m2.example.com:14100/sts/wss11user?wsdl
     ```

   For the complete procedure, see "Creating and Editing Web Service Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

5.3 Additional Resources on Oracle Web Services Manager

See the following resources for more information about the technologies and tools used to implement the solutions in this chapter:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding Oracle Web Services Manager*
- Oracle STS documentation at `http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oraclests-166231.html`
- Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

6

Configuring Federation with Oracle STS as the IP-STS and Microsoft ADFS 2.0 STS as the RP-STS

This chapter describes how to configure web services federation with Oracle STS as the Identity Provided STS (IP-STS) and Microsoft ADFS 2.0 STS as the Replying Party (RP-STS).

6.1 Use Case: Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Configure web services federation with Oracle STS as the IP-STS and Microsoft ADFS 2.0 STS as the RP-STS.

Solution
Attach Oracle Web Services Manager (OWSM) WS-Trust policies to the web service and client, and configure Oracle STS and Microsoft ADFS 2.0 STS to establish trust across security domains.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- Oracle STS
- Microsoft ADFS 2.0 STS
- Web service and client applications to be secured

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce message-level protection using SAML holder-of-key (HOK) authentication.

  Specifically, you attach the following policies to the client and service, respectively:

  - `oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy` and policies based on `oracle/sts_trust_config_client_template`
  - `oracle/wss11_sts_issued_saml_hok_with_message_protection_service_policy`

- Configure web services federation using Oracle STS as the IP-STS and Microsoft ADFS 2.0 STS is used as the RP-STS.

6.2 Use Case: Implementing Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS

This use case consists of the following tasks:

- Configuring the Web Service
- Configuring Microsoft ADFS 2.0 STS as the RP-STS
- Configuring Oracle STS as the IP-STS
- Configuring the Web Service Client

6.2.1 Configuring the Web Service

To configure the web service:

1. Attach `oracle/wss11_sts_issued_saml_hok_with_message_protection_service_policy` to the web service. For the complete procedure, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
2. Import the signing certificate for the ADFS 2.0 STS `/issuedtokensymmetricbasic256` endpoint into the OWSM keystore.
3. Define the ADFS 2.0 STS endpoint as a trusted issuer and a trusted DN. For the complete procedure, see "Defining Trusted Issuers and Trusted Distinguished Names List for SAML Signing Certificates" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

6.2.2 Configuring Microsoft ADFS 2.0 STS as the RP-STS

To configure Microsoft ADFS 2.0 STS as the RP-STS, perform the following steps.

For the complete procedure, see the Oracle STS documentation at `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`.

1. Confirm that the `/issuedtokensymmetricbasic256` endpoint is enabled.
2. Add the service as a relying party using the ADFS 2.0 management console.
3. Add the Oracle STS instance acting as the IP-STS as a trusted claim provider using the ADFS 2.0 management console.

6.2.3 Configuring Oracle STS as the IP-STS

To configure Oracle STS as the IP-STS, perform the following steps.

For the complete procedure, see the Oracle STS documentation at `http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oraclests-166231.html`.

1. Configure the Oracle STS `/wss11user` endpoint as follows:
   - Attach the policy with the URI `sts/wss11_username_token_with message_protection_service_policy`.
   - Create an `OWSM LRG UN Validation` validation template to validate the incoming token and apply it to the endpoint.
2. In Oracle STS, add the Microsoft ADFS 2.0 STS instance acting as the RP-STS as a relying partner party.
3. Enable the Audience Restriction Condition in Oracle STS.

   This step is necessary because ADFS 2.0 requires the SAML assertion for a claim provider to have AudienceRestrictionUri set, and assertions issued by Oracle STS do not have this set by default.
4. Configure a separate issuance template that issues 256 byte proof keys for Oracle STS to use.

6.2.4 Configuring the Web Service Client

To configure the web service client:

1. Create a policy from `oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy`, modify it as follows, and attach it to the client:
   - Set Algorithm Suite to Basic256 instead of Basic128.
   - Set Derived Keys to enabled.
   - Set `sts.in.order` to the URI of the ADFS 2.0 STS endpoint followed by the Oracle STS endpoint. For example:

     ```
     http://m1.example.com/adfs/services/trust/13/issuedtokensymmetricbasic256;
     http://m2.example.com:14100/sts/wss11user
     ```
2. Create a policy from `oracle/sts_trust_config_client_template`, modify it as follows, and attach it to the client:
   - Set Port URI to the ADFS 2.0 STS endpoint. For example:

     ```
     http://m1.example.com/adfs/services/trust/13/issuedtokensymmetricbasic256
     ```
   - Set Client Policy URI to the policy you created in Step 1.

     ```
     oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy_adfs
     ```
3. Create a policy from `oracle/sts_trust_config_client_template`, modify it as follows, and attach it to the client:

- Set Port URI to the Oracle STS endpoint; for example:

  `http://m2.example.com:14100/sts/wss11user`

- Set WSDL URI to the Oracle STS endpoint.

6.3 Additional Resources on Oracle Web Services Manager

See the following resources for more information about the technologies and tools used to implement the solutions in this chapter:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding Oracle Web Services Manager*
- Oracle STS documentation at `http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oraclests-166231.html`
- Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

7

Configuring SAML HOK Using WS-Trust with OpenSSO STS

This chapter describes how to configure SAML holder-of-key (HOK) with message protection using WS-Trust with OpenSSO STS.

This chapter contains the following sections:

7.1 Use Case: Configuring SAML HOK Using WS-Trust with OpenSSO STS

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Configure SAML holder-of-key (HOK) with message protection using WS-Trust with OpenSSO STS.

Solution
Attach Oracle Web Services Manager (OWSM) SAML HOK with message protection using WS-Trust policies to the web service and client, and configure OpenSSO STS.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- OpenSSO STS
- Web service and client applications to be secured

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce SAML HOK with message-level protection using WS-Trust with OpenSSO STS.

  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, these credentials can be obtained from a trusted Security Token Service (STS), which acts as trust broker. That is, the Web service client and the Web service do not explicitly trust each other; instead, they implicitly trust each other because they both trust the STS. For more information, see "Overview of Web Services WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Specifically, you attach the following policies to the client and service, respectively:

- `oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy`
- `oracle/wss11_sts_issued_saml_hok_with_message_protection_service_policy` and `oracle/sts_trust_config_service_policy`

- Configure OpenSSO STS.

This use case consists of a Java EE web service and SOA Composite client.

7.2 Configuring SAML HOK Using WS-Trust with OpenSSO STS

This use case describes the following tasks to implement SAML HOK message protection using WS-Trust with OpenSSO STS:

- Configuring OpenSSO STS
- Configuring SAML Holder-of-Key With Message Protection Using WS-Trust with OpenSSO STS

7.2.1 Configuring OpenSSO STS

To configure OpenSSO STS:

1. Log in to the OpenSSO STS instance.
2. Navigate to **Configuration > Global > Security Token Service**.
3. Under Security: Security Mechanism: Security Token Accepted by STS Services, enable all options.
4. Under the Credential for User Token section, add a new credential for the token with the username and password set as required.

   For this example, set the username and password both to **test**.
5. Under the On Behalf of Token section, select **ldapService** from the **Authentication Chain for On Behalf of Token** drop-down list.
6. Under the Signing section, enable the following options:

   - **Is Request Signature Verified**

   - **Is Response Signed Enabled** (select **Body** and **Timestamp**)
7. Under the Encryption section, enable the following options:

   - **Is Request Decrypted** (select **Body** and **Header**)

- **Is Response Encrypted**

8. Select **AES** from the **Encryption Algorithm** drop-down list, and select **128** from the **Encryption Strength** drop-down list.

9. To support the WS-Security 1.1 Kerberos token with message protection requestor token, under the Kerberos Configuration section and configure the following values:

   - Kerberos Domain Server

     Fully qualified hostname of the domain server.

   - Kerberos Domain

     Domain name.

   - Kerberos Service Principal

     Service principal name in the following format: *<host>/<machine name>@<REALM NAME>*

   - Kerberos Key Tab File

     Location of the key tab file created for the STS.

   - Is Verify Kerberos Signature

     Enable only when JDK6 is used.

10. To support SSL, perform the following steps:

    a. In the Token Issuance Attributes section, edit the SSL Endpoint based on your OpenSSO instance.

    b. Under Signing, enable the **Disable signature validation when transport is secured with SSL** option.

    c. Under Encryption, enable the **Disable decryption when transport is secured with SSL** option.

11. To support SSL on the server hosting the OpenSSO STS:

    - On the WebLogic Server hosting the OpenSSO STS, to configure SSL, perform the steps described in "Configuring Keystores for SSL" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

    - On GlassFish server hosting the Open SSO STS, perform the following steps:

      a. Generate a new key pair for the application server by issuing the following command:

      ```
      keytool -genkey -keyalg <algorithm for generating the
      key pair> -keystore keystore.jks -validity <days> -
      alias <alias_name>
      ```

      For example:

      ```
      keytool -genkey -keyalg RSA -keystore
      <glassfish_install_dir>/domains/<sts_deploy_domain>/
      config/keystore.jks -validity 365 -alias owsm
      ```

When prompted for first and last name, enter the hostname of the machine for which the certificate is to be generated. Enter the appropriate details for the other prompts.

b. Generate a Certificate Signing Request (CSR) by issuing the following command:

```
keytool -certreq -alias owsm -file owsm.csr -keystore
keystore.jks -storepass changeit
```

The request that is generated and written to the `owsm.csr` file needs to be submitted to a Certificate Authority in order to get a valid certificate. For example, the Certificate Management Server maintained by the OpenSSO QA team at `https://mahogany.red.iplanet.com`.

c. Access the Certificate Management Server at `https://mahogany.red.iplanet.com`, click **SSL Server** in the left pane, and paste the contents of the `.csr` file, starting from `BEGIN CERTIFICATE REQUEST` and ending at `END CERTIFICATE REQUEST`, into the **PKCS # 10 Request** field.

Fill out the other fields, as appropriate, and submit the request. Once the request is approved, the certificate can be retrieved from the retrieval tab on the same page.

d. Copy the certificate content (PKCS # 7 format) starting from `BEGIN CERTIFICATE` to `END CERTIFICATE` into a file with `.cert` extension and import the server certificate into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/keystore.jks` file by using the following keytool command:

```
keytool -import -v -alias owsm -file owsm.cert -
keystore keystore.jks -storepass changeit
```

Enter **YES** when prompted if you trust the certificate.

e. Access the Certificate Authority's SSL Certificate. Go to `https://mahogany.red.iplanet.com` and navigate to **SSL Server -> Retrieval tab -> List Certificates -> Find**. Click on the first **Details** button on the page and copy the Base 64 encoded certificate into another `.cert` file. For example: `mahogany.cert`

f. Import this certificate with alias as `rootca` into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/cacerts.jks` file, using the following command:

```
keytool -import -v -alias rootca -file mahogany.cert -
keystore cacerts.jks -storepass changeit
```

g. The previous step may need to be repeated for client side `truststore.jks` file. Delete any existing `rootca` aliases from that file and import the new one as shown above (changing the location of the keystore file).

h. To configure GlassFish with the new certificate, access the Administration Console at `http://hostname:admin-port/`, navigate to **Configuration -> HTTP Service -> http-listener2 (default SSL enabled port) -> SSL**, and change the certificate nickname from `s1as` (self-signed cert) to `owsm`.

i. Restart Glassfish.

7.2.2 Configuring SAML Holder-of-Key With Message Protection Using WS-Trust with OpenSSO STS

To configure SAML holder-of-key with message protection using WS-Trust with OpenSSO STS:

1. Configure the STS service policy. For the complete procedure, see "Configuring a Policy for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Make a copy of `oracle/sts_trust_config_service_policy` and edit the policy configuration, as described below, based on the requestor token type.

   To support WS-Security 1.0 username token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss10un"`
   - `orasp:wsdl-uri="http://<host>:<port>/openssosts/sts/wss10un?wsdl"` (Optional)

   To support WS-Security 1.0 username token over SSL with message protection requestor token:

   - `orasp:port-uri="https://<host>:<sslport>/openssosts/sts/tlswss10un"`
   - `orasp:wsdl-uri="https://<host>:<sslport>/openssosts/sts/tlswss10un?wsdl"` (Optional)

   To support WS-Security 1.0 X509 token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss10x509"`
   - `orasp:wsdl-uri="http://<host>:<port>/openssosts/sts/wss10x509?wsdl"` (Optional)

   To support WS-Security 1.1 Kerberos token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss11kerberos"`
   - `orasp:wsdl-uri="http://<host>:<port>` (Optional)

2. Configure the Web service. For the complete procedure, see "Configuring a Web Service for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Attach the policy created in step 1, followed by the `oracle/wss11_sts_issued_saml_hok_with_message_protection_service_policy` to the Java EE web service. For the complete procedure, see "Attaching Policies Directly to a Single Subject Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note:

By default, the `oracle/wss11_sts_issued_saml_hok_with_message_protection_service_policy policy` is configured with token type of SAML 1.1. If you wish to configure the token type to be SAML 2.0, you will need to make a copy of the policy and edit it, as described in "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. (This value should match the client policy.)

3. Configure the Web service client policy. For the complete procedure, see "Configuring a Web Service Client for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Attach the `oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy` policy to the SOA composite client and override the client configuration properties, described in "oracle/ws11_sts_issued_saml_hok_with_message_protection_client_template" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, as required for your requestor token.

   The `sts.auth.user.csf.key` should be set to the user credentials available in the default OpenSSO STS configuration. Namely, username `test`, with password set to `test`. Though, it is not required to be set for the X509 requestor token.

   For more information about overriding client configuration properties when attaching a policy, see "Attaching Policies Directly to Web Service Clients Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note:

By default, the `oracle/wss11_sts_issued_saml_hok_with_message_protection_client_policy policy` is configured with token type of SAML 1.1. If you wish to configure the token type to be SAML 2.0, you will need to make a copy of the policy and edit it, as described in "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

7.3 Additional Resources on Oracle Web Services Manager

See the following resources for more information about the technologies and tools used to implement the solutions in this chapter:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding Oracle Web Services Manager*
- `keytool` Javadoc at: `http://download.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`

8

Configuring SAML Sender Vouches Using WS-Trust with OpenSSO STS

This chapter describes how to configure SAML sender vouches using WS-Trust with OpenSSO STS.

This chapter contains the following sections:

8.1 Use Case: SAML Sender Vouches Using WS-Trust with OpenSSO STS

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Configure SAML sender vouches using WS-Trust with OpenSSO STS.

Solution
Attach Oracle Web Services Manager (OWSM) SAML sender vouches with message protection using WS-Trust policies to the web service client, an OWSM SAML sender vouches with message protection policy to the web service, and configure OpenSSO STS.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- OpenSSO STS
- Web service and client applications to be secured

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce SAML sender vouches with message-level protection using WS-Trust with OpenSSO STS.

  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, these credentials can be obtained from a trusted Security Token Service (STS), which acts as trust broker. That is, the Web service client and the Web service do not explicitly trust each other; instead, they implicitly trust each other because they both trust the STS. For more information, see "Overview of Web Services WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Specifically, you attach the following policies to the client and service, respectively:

- `oracle/wss11_sts_issued_saml_with_message_protection_client_policy` and `oracle/sts_trust_config_client_policy`
- `oracle/wss11_saml_token_with_message_protection_service_policy`

- Configure OpenSSO STS.

This use case consists of a Java EE web service and SOA Composite client.

8.2 Use Case: Implementing SAML Sender Vouches Using WS-Trust with OpenSSO STS

This use case describes the following tasks to implement SAML sender vouches message protection using WS-Trust with OpenSSO STS:

- Configuring OpenSSO STS
- Configuring SAML Sender Vouches With Message Protection Using WS-Trust with OpenSSO STS

8.2.1 Configuring OpenSSO STS

To configure OpenSSO STS:

1. Log in to the OpenSSO STS instance.
2. Navigate to **Configuration > Global > Security Token Service**.
3. Under Security: Security Mechanism: Security Token Accepted by STS Services, enable all options.
4. Under the Credential for User Token section, add a new credential for the token with the username and password set as required.

   For this example, set the username and password both to **test**.
5. Under the On Behalf of Token section, select **ldapService** from the **Authentication Chain for On Behalf of Token** drop-down list.
6. Under the Signing section, enable the following options:

   - **Is Request Signature Verified**

   - **Is Response Signed Enabled** (select **Body** and **Timestamp**)
7. Under the Encryption section, enable the following options:

   - **Is Request Decrypted** (select **Body** and **Header**)

- **Is Response Encrypted**

8. Select **AES** from the **Encryption Algorithm** drop-down list, and select **128** from the **Encryption Strength** drop-down list.

9. To support the WS-Security 1.1 Kerberos token with message protection requestor token, under the Kerberos Configuration section and configure the following values:

   - Kerberos Domain Server

     Fully qualified hostname of the domain server.

   - Kerberos Domain

     Domain name.

   - Kerberos Service Principal

     Service principal name in the following format: `<host>/<machine name>@<REALM NAME>`

   - Kerberos Key Tab File

     Location of the key tab file created for the STS.

   - Is Verify Kerberos Signature

     Enable only when JDK6 is used.

10. To support SSL, perform the following steps:

    a. In the Token Issuance Attributes section, edit the SSL Endpoint based on your OpenSSO instance.

    b. Under Signing, enable the **Disable signature validation when transport is secured with SSL** option.

    c. Under Encryption, enable the **Disable decryption when transport is secured with SSL** option.

11. To support SSL on the server hosting the OpenSSO STS:

    On the WebLogic Server hosting the OpenSSO STS, to configure SSL, perform the steps described in "Configuring Keystores for SSL" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

    On GlassFish server hosting the Open SSO STS, perform the following steps:

    a. Generate a new key pair for the application server by issuing the following command:

       ```
       keytool -genkey -keyalg <algorithm for generating the key pair> -keystore keystore.jks -validity <days> -alias <alias_name>
       ```

       For example:

       ```
       keytool -genkey -keyalg RSA -keystore <glassfish_install_dir>/domains/<sts_deploy_domain>/config/keystore.jks -validity 365 -alias owsm
       ```

When prompted for first and last name, enter the hostname of the machine for which the certificate is to be generated. Enter the appropriate details for the other prompts.

b. Generate a Certificate Signing Request (CSR) by issuing the following command:

```
keytool -certreq -alias owsm -file owsm.csr -keystore
keystore.jks -storepass changeit
```

The request that is generated and written to the `owsm.csr` file needs to be submitted to a Certificate Authority in order to get a valid certificate. For example, the Certificate Management Server maintained by the OpenSSO QA team at `https://mahogany.red.iplanet.com`.

c. Access the Certificate Management Server at `https://mahogany.red.iplanet.com`, click **SSL Server** in the left pane, and paste the contents of the `.csr` file, starting from `BEGIN CERTIFICATE REQUEST` and ending at `END CERTIFICATE REQUEST`, into the **PKCS # 10 Request** field.

Fill out the other fields, as appropriate, and submit the request. Once the request is approved, the certificate can be retrieved from the retrieval tab on the same page.

d. Copy the certificate content (PKCS # 7 format) starting from `BEGIN CERTIFICATE` to `END CERTIFICATE` into a file with `.cert` extension and import the server certificate into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/keystore.jks` file by using the following keytool command:

```
keytool -import -v -alias owsm -file owsm.cert -keystore
keystore.jks -storepass changeit
```

Enter **YES** when prompted if you trust the certificate.

e. Access the Certificate Authority's SSL Certificate. Go to `https://mahogany.red.iplanet.com` and navigate to **SSL Server -> Retrieval tab -> List Certificates -> Find**. Click on the first **Details** button on the page and copy the Base 64 encoded certificate into another `.cert` file. For example: `mahogany.cert`

f. Import this certificate with alias as `rootca` into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/cacerts.jks` file, using the following command:

```
keytool -import -v -alias rootca -file mahogany.cert -
keystore cacerts.jks -storepass changeit
```

g. The previous step may need to be repeated for client side `truststore.jks` file. Delete any existing `rootca` aliases from that file and import the new one as shown above (changing the location of the keystore file).

h. To configure GlassFish with the new certificate, access the Administration Console at `http://hostname:admin-port/`, navigate to **Configuration -> HTTP Service -> http-listener2 (default SSL enabled port) -> SSL**, and change the certificate nickname from `s1as` (self-signed cert) to `owsm`.

i. Restart Glassfish.

8.2.2 Configuring SAML Sender Vouches With Message Protection Using WS-Trust with OpenSSO STS

To configure SAML sender vouches with message protection using WS-Trust with OpenSSO STS:

1. Configure the client-side STS policy. For the complete procedure, see "Configuring a Policy for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Note:

   Automatic Policy Configuration cannot be used for SAML sender vouches confirmation because the trust is between the Web service and the client. For more information, see "Configuring SAML Sender Vouches with WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Make a copy of `oracle/sts_trust_config_client_policy` and edit the policy configuration based on the requestor token type.

   To support WS-Security 1.0 username token with message protection requestor token:

   - `orasp:policy-reference-uri="oracle/wss10_username_token_with_message_protection_client_policy"`
   - `orasp:port-endpoint="http://<host>:<port>/openfm/SecurityTokenService/#wsdl.endpoint(SecurityTokenService/ISecurityTokenService_Port_UN_WSS10_SOAP12)"`
   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss10un"`
   - `orasp:sts-keystore-recipient-alias="test"`

   To support WS-Security 1.0 username token over SSL with message protection requestor token:

   - `orasp:policy-reference-uri="oracle/wss_username_token_over_ssl_client_policy"`
   - `orasp:port-endpoint="http://localhost:8080/openfm/SecurityTokenService/#wsdl.endpoint(SecurityTokenService/ISecurityTokenService_Port_TLS_UN_WSS10_SOAP12)"`
   - `orasp:port-uri="https://<host>:<sslport>/openssosts/sts/tlswss10un"`
   - `orasp:sts-keystore-recipient-alias="test"`

   To support WS-Security 1.0 X509 token with message protection requestor token:

   - `orasp:policy-reference-uri="oracle/wss10_x509_token_with_message_protection_client_policy"`

- `orasp:port-endpoint="http://localhost:8080/openfm/SecurityTokenService/#wsdl.endpoint(SecurityTokenService/ISecurityTokenService_Port_X509_WSS10_SOAP12)"`
- `orasp:port-uri="http://<host>:<port>`/openssosts/sts/wss10x509"
- `orasp:sts-keystore-recipient-alias="test"`

2. Attach the `oracle/wss11_saml_token_with_message_protection_service_policy` policy to the Java EE web service (there is no corresponding issued token policy for SAML sender vouches scenarios) and override the `keystore.enc.csf.key` to specify the service encryption key alias and password. For the complete procedure, see "Attaching Policies Directly to a Single Subject Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Note:

   By default, the `oracle/wss11_saml_hok_with_message_protection_service_policy` policy is configured with token type of SAML 1.1. If you wish to configure the token type to be SAML 2.0, you will need to make a copy of the policy and edit it, as described in "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Attach the policy created in step 1 followed by the `oracle/wss11_sts_issued_saml_with_message_protection_client_policy` policy to the SOA composite client and override the client configuration properties described in "wss11_sts_issued_saml_with_message_protection_client_template" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, as required for your requestor token.

   For the complete procedure, see "Attaching Policies Directly to a Single Subject Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   The "On Behalf Of" use case relies on the `sts.auth.on.behalf.of.csf.key` and `on.behalf.of` properties, as described in "wss11_sts_issued_saml_with_message_protection_client_template" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. For more information, see "On Behalf Of Use Cases" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   The `on.behalf.of` property should be set to `true`. The `sts.auth.on.behalf.of.csf.key` should be set to the user credentials available in the default OpenSSO STS configuration that support the "on behalf of" use case. Namely, `demo`, with password set to `changeit`.

   Note:

   For more information about overriding client configuration properties when attaching a policy, see "Attaching Policies Directly to Web Service Clients Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

4. To grant permission to the client application to request a token from OpenSSO STS "on behalf of" a user, grant the `WSIdentityPermission` to `wsm-agent-core.jar`. For the complete procedure, see "Set the WSIdentityPermission Permission" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

8.3 Additional Resources on Oracle Web Services Manager

See the following resources for more information about the technologies and tools used to implement the solutions in this chapter:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding Oracle Web Services Manager*
- `keytool` Javadoc at: `http://download.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`

9

Configuring SAML Bearer Using WS-Trust with OpenSSO STS

This chapter describes how configure SAML bearer using WS-Trust with OpenSSO STS.

This chapter contains the following sections:

9.1 Use Case: Configuring SAML Bearer Using WS-Trust with OpenSSO STS

The use case summary helps you quickly determine whether information in this chapter meets your needs.

The following list summarizes the use case goals, solution, and components. Links to required documentation are also provided.

Use Case
Configure SAML bearer using WS-Trust with OpenSSO STS.

Solution
Attach Oracle Web Services Manager (OWSM) SAML bearer with message protection using WS-Trust policies to the web service and client, and configure OpenSSO STS.

Components

- Oracle WebLogic Server
- Oracle Web Services Manager (OWSM)
- OpenSSO STS
- Web service and client applications to be secured

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce SAML bearer with message-level protection using WS-Trust with OpenSSO STS.

  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, these credentials can be obtained from a trusted Security Token Service (STS), which acts as trust broker. That is, the Web service client and the Web service do not explicitly trust each other; instead, they implicitly trust each other because they both trust the STS. For more information, see "Overview of Web Services WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Specifically, you attach the following policies to the client and service, respectively:

- `oracle/ws11_sts_issued_saml_bearer_token_over_ssl_client_policy`
- `oracle/wss11_sts_issued_saml_bearer_token_over_ssl_service_policy` and `oracle/sts_trust_config_service_policy`

- Configure OpenSSO STS.

This use case consists of a Java EE web service and SOA Composite client.

9.2 Use Case: Implementing SAML Bearer Using WS-Trust with OpenSSO STS

This use case describes the following tasks to implement SAML bearer message protection using WS-Trust with OpenSSO STS:

- Configuring OpenSSO STS
- Configuring SAML Bearer With Message Protection Using WS-Trust with OpenSSO STS

9.2.1 Configuring OpenSSO STS

To configure OpenSSO STS:

1. Log in to the OpenSSO STS instance.
2. Navigate to **Configuration > Global > Security Token Service**.
3. Under Security: Security Mechanism: Security Token Accepted by STS Services, enable all options.
4. Under the Credential for User Token section, add a new credential for the token with the username and password set as required.

   For this example, set the username and password both to **test**.
5. Under the On Behalf of Token section, select **ldapService** from the **Authentication Chain for On Behalf of Token** drop-down list.
6. Under the Signing section, enable the following options:

   - **Is Request Signature Verified**

   - **Is Response Signed Enabled** (select **Body** and **Timestamp**)
7. Under the Encryption section, enable the following options:

   - **Is Request Decrypted** (select **Body** and **Header**)

- **Is Response Encrypted**

8. Select **AES** from the **Encryption Algorithm** drop-down list, and select **128** from the **Encryption Strength** drop-down list.

9. To support the WS-Security 1.1 Kerberos token with message protection requestor token, under the Kerberos Configuration section and configure the following values:

   - Kerberos Domain Server

     Fully qualified hostname of the domain server.

   - Kerberos Domain

     Domain name.

   - Kerberos Service Principal

     Service principal name in the following format: <host>/<machine name>@<REALM NAME>

   - Kerberos Key Tab File

     Location of the key tab file created for the STS.

   - Is Verify Kerberos Signature

     Enable only when JDK6 is used.

10. To support SSL, perform the following steps:

    a. In the Token Issuance Attributes section, edit the SSL Endpoint based on your OpenSSO instance.

    b. Under Signing, enable the **Disable signature validation when transport is secured with SSL** option.

    c. Under Encryption, enable the **Disable decryption when transport is secured with SSL** option.

11. To support SSL on the server hosting the OpenSSO STS:

    On the WebLogic Server hosting the OpenSSO STS, to configure SSL, perform the steps described in "Configuring Keystores for SSL" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

    On the GlassFish server hosting the Open SSO STS, perform the following steps:

    a. Generate a new key pair for the application server by issuing the following command:

       ```
       keytool -genkey -keyalg <algorithm for generating the key
       pair> -keystore keystore.jks -validity <days> -alias
       <alias_name>
       ```

       For example:

       ```
       keytool -genkey -keyalg RSA -keystore
       <glassfish_install_dir>/domains/<sts_deploy_domain>/
       config/keystore.jks -validity 365 -alias owsm
       ```

When prompted for first and last name, enter the hostname of the machine for which the certificate is to be generated. Enter the appropriate details for the other prompts.

b. Generate a Certificate Signing Request (CSR) by issuing the following command:

```
keytool -certreq -alias owsm -file owsm.csr -keystore
keystore.jks -storepass changeit
```

The request that is generated and written to the `owsm.csr` file needs to be submitted to a Certificate Authority in order to get a valid certificate. For example, the Certificate Management Server maintained by the OpenSSO QA team at `https://mahogany.red.iplanet.com`.

c. Access the Certificate Management Server at `https://mahogany.red.iplanet.com`, click **SSL Server** in the left pane, and paste the contents of the `.csr` file, starting from `BEGIN CERTIFICATE REQUEST` and ending at `END CERTIFICATE REQUEST`, into the **PKCS # 10 Request** field.

Fill out the other fields, as appropriate, and submit the request. Once the request is approved, the certificate can be retrieved from the retrieval tab on the same page.

d. Copy the certificate content (PKCS # 7 format) starting from `BEGIN CERTIFICATE` to `END CERTIFICATE` into a file with `.cert` extension and import the server certificate into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/keystore.jks` file by using the following keytool command:

```
keytool -import -v -alias owsm -file owsm.cert -keystore
keystore.jks -storepass changeit
```

Enter **YES** when prompted if you trust the certificate.

e. Access the Certificate Authority's SSL Certificate. Go to `https://mahogany.red.iplanet.com` and navigate to **SSL Server -> Retrieval tab -> List Certificates -> Find**. Click on the first **Details** button on the page and copy the Base 64 encoded certificate into another `.cert` file. For example: `mahogany.cert`

f. Import this certificate with alias as `rootca` into the `<glassfish_install_dir>/domains/<sts_deploy_domain>/config/cacerts.jks` file, using the following command:

```
keytool -import -v -alias rootca -file mahogany.cert -
keystore cacerts.jks -storepass changeit
```

g. The previous step may need to be repeated for client side `truststore.jks` file. Delete any existing `rootca` aliases from that file and import the new one as shown above (changing the location of the keystore file).

h. To configure GlassFish with the new certificate, access the Administration Console at `http://hostname:admin-port/`, navigate to **Configuration -> HTTP Service -> http-listener2 (default SSL enabled port) -> SSL**, and change the certificate nickname from `s1as` (self-signed cert) to `owsm`.

i. Restart Glassfish.

9.2.2 Configuring SAML Bearer With Message Protection Using WS-Trust with OpenSSO STS

To configure SAML bearer with message protection using WS-Trust with OpenSSO STS:

1. Configure the STS service policy. For the complete procedure, see "Setting Up Automatic Policy Configuration for STS" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Make a copy of `oracle/sts_trust_config_service_policy` and edit the policy configuration, as described below, based on the requestor token type.

   To support WS-Security 1.0 username token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss10un"`
   - `orasp:wsdl-uri="http://<host>:<port>/openssosts/sts/wss10un?wsdl"` (Optional)

   To support WS-Security 1.0 username token over SSL with message protection requestor token:

   - `orasp:port-uri="https://<host>:<sslport>/openssosts/sts/tlswss10un"`
   - `orasp:wsdl-uri="https://<host>:<sslport>/openssosts/sts/tlswss10un?wsdl"` (Optional)

   To support WS-Security 1.0 X509 token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss10x509"`
   - `orasp:wsdl-uri="http://<host>:<port>/openssosts/sts/wss10x509?wsdl"` (Optional)

   To support WS-Security 1.1 Kerberos token with message protection requestor token:

   - `orasp:port-uri="http://<host>:<port>/openssosts/sts/wss11kerberos"`
   - `orasp:wsdl-uri="http://<host>:<port>` (Optional)

2. Configure the Web service. For the complete procedure, see "Configuring a Web Service for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Attach the policy created in step 1 followed by the `oracle/wss11_sts_issued_saml_bearer_token_over_ssl_service_policy`. For the complete procedure, see "Attaching Policies Directly to a Single Subject Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Configure the Web service client. For the complete procedure, see "Configuring a Web Service Client for Automatic Policy Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   Attach the `oracle/ws11_sts_issued_saml_bearer_token_over_ssl_client_policy` policy to the SOA composite client and override the client configuration properties described in "oracle/ws11_sts_issued_saml_bearer_token_over_ssl_client_template" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, as required for your requestor token. For the complete procedure, see "Attaching Policies Directly to a Single Subject Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   The `sts.auth.user.csf.key` should be set to the user credentials available in the default OpenSSO STS configuration. Namely, username `test`, with password set to `test`. Though, it is not required to be set for the X509 requestor token.

   For more information about overriding client configuration properties when attaching a policy, see "Attaching Policies Directly to Web Service Clients Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

9.3 Additional Resources on Oracle Web Services Manager

See the following resources for more information about the technologies and tools used to implement the solutions in this chapter:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Understanding Oracle Web Services Manager*
- `keytool` Javadoc at: `http://download.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`