Oracle® Fusion Middleware

Use Cases for Securing Web Services Using Oracle Web Services Manager

14*c* (14.1.2.0.0)
G12140-01
December 2024

Oracle Fusion Middleware Use Cases for Securing Web Services Using Oracle Web Services Manager, 14*c* (14.1.2.0.0)

G12140-01

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 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 14*c* (14.1.2.0.0)

This revision contains no new features. Minor updates were made throughout the guide.

1 Introduction to the Use Cases

You can secure web services using various methods in Oracle Web Services Manager (OWSM).

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

You can refer to the use case provided in this chapter to understand how to secure inbound SOAP requests using SAML message protection. To implement SAML message protection, you need to perform a sequence of tasks: such as creating a WebLogic Server user, creating a Java Keystore, and so on.

- Use Case: Securing Inbound SOAP Requests Using SAML-based Authentication
- Securing Inbound SOAP requests using SAML Message Protection
- Implementing SAML Message Protection

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

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to secure inbound SOAP requests using SAML-based authentication.

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

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

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

Symmetric key cryptography relies on a single, shared secret key. The client creates the symmetric key, uses it to sign and encrypt the message, encrypts the symmetric key by using the web service's certificate, and shares it with the web service in the request message. 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?

If the client and web service are in the same domain with access to the same keystore, then 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.
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`. 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, you need to perform a sequence of tasks: creating a WebLogic Server user, creating a Java Keystore, configuring the OWSM Keystore, storing the password for the decryption key in the Credential Store, attaching the policy to your web service and web service client.

- 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 implementing SAML message protection, download and install product components, configure WebLogic domain, start the Administration Server, and get the access to Oracle Enterprise Manager Fusion Middleware Control and Oracle WebLogic Server Remote Console.

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 Remote Console

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

2.3.2 Creating a WebLogic Server User

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 by using the WebLogic Server Remote Console.

It is 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. You can create and manage the Java keystore by 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 OWSM OAuth 2.0 with IDCS OAuth 2.0

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to secure RESTful web services using OWSM OAuth 2.0 with IDCS OAuth 2.0 Server.

This chapter contains the following sections:

3.1 Use Case: Secure RESTful Web Services Using OWSM OAuth 2.0 with IDCS OAuth2 Server

You can develop a RESTful web services and secure them to the resource and client applications on IDCS using OWSM policies.

Use Case
Secure RESTful Web Services Using OWSM OAuth 2.0 with IDCS OAuth2 Server.

Implementation Summary
Develop a RESTful web services and secure them to the resource and client applications on IDCS using OWSM policies

Components

- Oracle WebLogic Server
- Oracle Fusion Middleware
- Oracle Web Services Manager (OWSM)
- IDCS

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

- Developing RESTful Web Services

This use case includes the following steps:

- Configuring IDCS Security provider with WLS
- IDCS OAuth2 Configuration

- Secure JAX-RS REST Service using OWSM OAuth2 security policy
- Secure JAX-RS REST Client using OWSM OAuth2 security policies

3.2 Implementing Web Services for IDCS - Prerequisites

Before implementing Web Services for IDCS, download and install IDCS, configure WebLogic domain, start the Administration Server, and get the access to Oracle Enterprise Manager Fusion Middleware Control and Oracle WebLogic Server Remote Console.

Before you begin, ensure the following:

- Download and install Web Logic Server and create the domain.
- Download and install Oracle Fusion Middleware—includes OWSM.

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

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

  For more information, see "Getting Started with Oracle Identity Cloud Service” in Administering Oracle Identity Cloud Service.
- Configure OPSS SCIM based Identity Store Service.

  For more information, see “Configuring the Identity Store” in Fusion Middleware Securing Applications with Oracle Platform Security Services.

3.3 Configuring IDCS Security provider with WLS

A single security provider named OracleIdentityCloudIntegrator combines identity assertion and authentication. This security provider is associated with an OAuth Client for WLS to authenticate users with IDCS.

Note:

The WebLogic boot user is not present in IDCS by default, so an authentication provider which contains the boot user is required. If the boot user is stored in embedded LDAP, the DefaultAuthenticator will be required for boot.

Follow the steps below to install and configure OracleIdentityCloudIntegrator in WLS:

Install the IDCS Security Provider

The OracleIdentityCloudIntegrator security provider is included with WLS.

Obtain an OAuth Client from IDCS

The IDCS security provider is associated with an OAuth Client to enable WLS to authenticate users with IDCS . This OAuth client is registered with the IDCS instance and allows the security provider to access IDCS. An OAuth Client provides atleast three properties listed below required by the IDCS security provider:

Property Name	Type	Comment
ClientTenant	String	Tenant name where the OAuth Client was provisioned
ClientId	String	OAuth Client Id used to access the IDCS user store
ClientSecret	String	OAuth Client Secret (i.e. Password) to generate access tokens

Note:

The OAuth Client is used within the specific IDCS tenant that it was provisioned. When a requirement to use this client to access other IDCS tenants is raised, the Cross Tenant IDCS AppRole is granted to the client.

1. Setup the OAuth Client with Identity Domain Administrator IDCS AppRole
   a. Login to the **IDCS Admin console** as Tenant Administrator
   b. Access the **Applications** tab and select **Add**
   c. Select the **Trusted Application** option and follow steps listed below in the setup wizard:
      - Type a client name and description and select **Next**
      - Select **Configure this application as a client now**
      - Select **Client Credentials** option for Allowed Grant Types only
      - Select the box **Grant the client access to Identity Cloud Service Admin APIs**
      - Click on the white drop down box and select **Identity Domain Administrator**
      - Select **Next** keeping other options blank
      - Select **Finish.**
   d. Record the Client Id, Client Secret and Tenant Name for the login
   e. Activate the new application and ensure your OAuth Client is activated by following the steps below:
      - Access **Applications** tab
      - Select the newly created application
      - Click **Activate** in the menu bar.
2. Setup the OAuth Client for access to additional IDCS tenants, if required. Access to additional IDCS tenants can be granted by any of the following options:

   Option 1: Grant using CTAppRoleGranter JSON

```
CTAppRoleGranter.json
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:idcs:CrossTenantAppRoleGranter"
    ]
}
```

3. **Option 2: Grant the Cross Tenant IDCS AppRole via PUT operation**

```
% curl -X PUT -H "Content-type: application/json"
       -H "Authorization: Bearer <access-token>"
 --data @CTAppRoleGranter.json http://tenant1.identity.c9dev0.oc9qadev.com/admin/v1/
CrossTenantAppRoleGranter/a28132cdfdf5367a9f7810a3f47bbdf6
```

Configure the IDCS Security Provider

You can configure the installed IDCS security provider using WLST online, WLST offline, or WLS Admin Console.

To configure using WLST online and WLST offline, see Configuring the Oracle Identity Cloud Integrator Provider: Main Steps and Examples.

To configure using the Admin Console, see Manage Security Providers.

Configure SSL for the IDCS Security Provider

You have to configure SSL in the IDCS security provider and make sure the outbound http connections to the IDCS instance over SSL work properly.

See Configuring TLS/SSL for the Oracle Identity Cloud Integrator Provider.

3.4 IDCS OAuth2 Configuration

OAuth2 Client and OAuth2 Service are registered on IDCS.

See the following sections:

- Registering OAuth2 Service on IDCS
- Configuring OAuth2 Client on IDCS

3.4.1 Registering Oauth2 Service on IDCS

New Resource Applications are configured on IDCS and URL of resources are added to the resource application.

Configuring New Resource Application on IDCS

Follow the steps below to configure new resource application on IDCS:

1. Login in to the **IDCS Admin Console**
2. Select the **Applications** tab
3. Add **New Application**
4. In the application wizard, select **Web Application** and provide details
5. In the wizard select **Skip Client Configuration** and go to the next page
6. On resources page provide scope, primary audience and secondary audiences
7. Finish the wizard
8. Save the application
9. Click **Activate** to activate the application.

> **Note:**
>
> ***"RSApp"*** application is created for testing. You can add more resource URLs as secondary audience(s) to integrate with test resources.

Add URL of Resources to Resource Application on IDCS

Follow the steps below to add URL of Resources to Resource Application on IDCS:

1. Login in to the **IDCS Admin Console**
2. Go to the **Applications** tab
3. Select ***"RSApp"*** application
4. Go to the **Configuration** Tab
5. Expand **Resource** section
6. Type URL against **Secondary Audience**
7. Click on **Add** (against secondary audience) to add secondary audience
8. Click on **Save** , to save changes to application.

3.4.2 Configuring OAuth2 Client on IDCS

OAuth2 Client is configured on IDCS.

You can configure OAuth2 client on IDCS by following the steps below:

1. Login in to the **IDCS Admin Console**
2. Go to the **Applications** tab
3. Add New Application

   > **Note:**
   >
   > Client application is different form resource application.

4. In Application wizard, select **Web Application** and provide details
5. In Client Configuration, register the client and provide the following information:
   - Grant Types
   - Client Type
   - Import Client Certificate
   - Add Allowed Scope

     > **Note:**
     >
     > Scope is selected from resource applications that are already added.

   - Grant the Client Access to Identity Cloud Service Admin APIs. - Select Identity Domain Administrator.

6. Finish the wizard
7. Save the application
8. Click **Activate** to activate the application.

> **Note:**
>
> ***"ClientApp"*** application is created for testing. Note the client ID and secret.

3.5 Secure JAX-RS REST Services using OWSM OAuth2 security policies

Secure REST Service using following service side OWSM WS Policy.

Policy	Description
`oracle/multi_token_over_ssl_rest_service_policy`	Enforces one of the authentication policies - saml, http, spnego, jwt etc, based on the token sent by the client.

Attach OWSM Policy Globally

```
wls:/service_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.
wls:/service_domain/serverConfig> createWSMPolicySet('oauth-ps', 'rest-resource',
'Domain("*")', 'Policy set for All Rest Resources', 'true')

The policy set was created successfully in the session.

true
wls:/service_domain/serverConfig> attachWSMPolicy('oracle/
multi_token_over_ssl_rest_service_policy')
Policy reference "oracle/multi_token_over_ssl_rest_service_policy" added.
wls:/service_domain/serverConfig> commitWSMSession()
```

Import IDCS Signing Certificate

1. Use the following client ID/secret and scope ***urn:opc:idm:__myscopes__,***

   to get the Access token. The Access Token obtained is used to get IDCS signing certificate.

   > **Note:**
   >
   > Encode (ClientID:ClientSecret) into base64 through https://www.base64encode.org/

Curl to get Access Token

```
$ curl -i -H 'Content-Type:application/x-www-form-urlencoded; charset=utf-8' -H
'Authorization:Basic
MzhlZjQyZmRiOTJlNDY3YjkzNWIxMzhmNmIwMmQyMTE6MDQwN2ViYjMtZWM3NS00Y2FlLTkxMzItODI0M2FiM
2Q4NTNj'
--request POST 'https://owsm.identity.c9dev0.oc9qadev.com:443/oauth2/v1/token' -d
'grant_type=client_credentials&
scope=urn:opc:idm:__myscopes__'
#response
HTTP/1.1 200 OK
Server: Oracle-Traffic-Director/11.1.1.9
Date: Mon, 11 Jul 2016 11:50:27 GMT
X-xss-protection: 1; mode=block
X-content-type-options: nosniff
Cache-control: no-store
Pragma: no-cache
Content-type: application/json;charset=UTF-8
Content-length: 1915
Via: 1.1 net-idcs-config
Proxy-agent: Oracle-Traffic-Director/11.1.1.9

{"access_token":"eyJ4NXQjUzI1NiI6Ijg1a3E1MFVBVmNSRDJOUTR6WVZMVDZXbndUZmVidjBhNGV2YUJG
MjFqbU0iLCJ4NXQiOiJNMm1hRm0zVllsTUJPbjNHZXRWV0dYa3JLcmsiLCJraWQiOiJTSUdOSU5HX0tFWSIsI
mFsZyI6IlJTMjU2In0.eyJzdWIiOiIzOGVmNDJmZGI5MmU0NjdiOTM1YjEzOGY2YjAyZDIxMSIsInVzZXIudG
VuYW50Lm5hbWUiOiJvd3NtIiwic3ViX21hcHBpbmdhdHRyIjoidXNlck5hbWUiLCJpc3MiOiJodHRwczpcL1w
vaWRlbnRpdHkub3JhY2xlY2xvdWQuY29tXC8iLCJ0b2tfdHlwZSI6IkFUIiwiY2xpZW50X2lkIjoiMzhlZjQy
ZmRiOTJlNDY3YjkzNWIxMzhmNmIwMmQyMTEiLCJ1c2VyX2lzQWRtaW4iOnRydWUsImF1ZCI6WyJodHRwczpcL
1wvb3dzbS5pZGVudGl0eS5jOWRldjAub2M5cWFkZXYuY29tOjQ0MyIsInVybjpvcGM6bGJhYXM6bG9naWNhbG
d1aWQ9b3dzbSJdLCJjbGllbnRBcHBSb2xlcyI6WyJBdXRoZW50aWNhdGVkIENsaWVudCJdLCJzY29wZSI6InV
ybjpvcGM6aWRtOnQuc2VjdXJpdHkuY2xpZW50IiwiY2xpZW50X3RlbmFudG5hbWUiOiJvd3NtIiwiZXhwIjox
NDg5MDc1NzAwLCJpYXQiOjE0ODkwNzIxMDAsImNsaWVudF9uYW1lIjoiMTJfMl8xXzIgaWRjcyB0ZXN0IG5vb
i1zc2wgY2xpZW50IiwidGVuYW50Ijoib3dzbSIsImp0aSI6IjY3ZWEzZDk5LWNiNGEtNDlkYS1iNWE5LWYyZj
M4OTA0ODQ2OCJ9.dzKdnUS5hPMduP3jJ-G-v56qmagLNMjKNPsilQuAbxf8uj2z2ZB5I-
RjOocihhahbqlsBZUOOMuzhTZHzFy5AGBKv-mMeraFl87c3Xhjmw3r2phC9T-
YfGgRUSEwxrRsKF0FkIi4TX9Kwi0hdrKiMCMFV1gav5v1dGmklwCfNjQ","token_type":"Bearer","expi
res_in":3600}
```

2. Use Access Token obtained above to get the signing certificates.

Curl to get Signing Certificates

```
$ curl -X GET -H 'Content-Type:application/scim+json' -H 'Authorization:Bearer
eyJ4NXQjUzI1NiI6Ijg1a3E1MFVBVmNSRDJOUTR6WVZMVDZXbndUZmVidjBhNGV2YUJGMjFqbU0iLCJ4NXQiO
iJNMm1hRm0zVllsTUJPbjNHZXRWV0dYa3JLcmsiLCJraWQiOiJTSUdOSU5HX0tFWSIsImFsZyI6IlJTMjU2In
0.eyJzdWIiOiIzOGVmNDJmZGI5MmU0NjdiOTM1YjEzOGY2YjAyZDIxMSIsInVzZXIudGVuYW50Lm5hbWUiOiJ
vd3NtIiwic3ViX21hcHBpbmdhdHRyIjoidXNlck5hbWUiLCJpc3MiOiJodHRwczpcL1wvaWRlbnRpdHkub3Jh
Y2xlY2xvdWQuY29tXC8iLCJ0b2tfdHlwZSI6IkFUIiwiY2xpZW50X2lkIjoiMzhlZjQyZmRiOTJlNDY3YjkzN
WIxMzhmNmIwMmQyMTEiLCJ1c2VyX2lzQWRtaW4iOnRydWUsImF1ZCI6WyJodHRwczpcL1wvb3dzbS5pZGVudG
l0eS5jOWRldjAub2M5cWFkZXYuY29tOjQ0MyIsInVybjpvcGM6bGJhYXM6bG9naWNhbGd1aWQ9b3dzbSJdLCJ
jbGllbnRBcHBSb2xlcyI6WyJBdXRoZW50aWNhdGVkIENsaWVudCJdLCJzY29wZSI6InVybjpvcGM6aWRtOnQu
c2VjdXJpdHkuY2xpZW50IiwiY2xpZW50X3RlbmFudG5hbWUiOiJvd3NtIiwiZXhwIjoxNDg5MDc1NzAwLCJpY
XQiOjE0ODkwNzIxMDAsImNsaWVudF9uYW1lIjoiMTJfMl8xXzIgaWRjcyB0ZXN0IG5vbi1zc2wgY2xpZW50Ii
widGVuYW50Ijoib3dzbSIsImp0aSI6IjY3ZWEzZDk5LWNiNGEtNDlkYS1iNWE5LWYyZjM4OTA0ODQ2OCJ9.dz
KdnUS5hPMduP3jJ-G-v56qmagLNMjKNPsilQuAbxf8uj2z2ZB5I-
RjOocihhahbqlsBZUOOMuzhTZHzFy5AGBKv-mMeraFl87c3Xhjmw3r2phC9T-
YfGgRUSEwxrRsKF0FkIi4TX9Kwi0hdrKiMCMFV1gav5v1dGmklwCfNjQ' https://
owsm.identity.c9dev0.oc9qadev.com:443/admin/v1/SigningCert/jwk

#GET response
{"keys":
[{"kty":"RSA","e":"AQAB","x5t":"M2maFm3VYlMBOn3GetVWGXkrKrk","kid":"SIGNING_KEY","x5c
":
["MIICUDCCAbmgAwIBAgIELfGcXDANBgkqhkiG9w0BAQUFADBXMRMwEQYKCZImiZPyLGQBGRYDY29tMRYwFAY
KCZImiZPyLGQBGRYGb3JhY2xlMRUwEwYKCZImiZPyLGQBGRYFY2xvdWQxETAPBgNVBAMTCENsb3VkOUNBMB4X
DTE1MTEyMDA5MzI0OFoXDTI1MTExNzA5MzI0OFowXzETMBEGCgmSJomT8ixkARkWA2NvbTEWMBQGCgmSJomT8
ixkARkWBm9yYWNsZTEVMBMGCgmSJomT8ixkARkWBWNsb3VkMRkwFwYDVQQDDBBvcmNsTVQxMjMyMzJfaWRtMI
GfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCLVvyue+qFraxwM5LxaNLt2QH3wHn/
n0+yk2jmP7mpYkz1xrKuEk2e2SCggzK8MT9jJ5VUaNlF0MwhIZ8/naxA5LPCzGEVfZ/
41GPtGNADFyspqGHkdsNv+M2eCBme7MDp9L3noBtt2peqGqxSu0DHyt1wgNr6p6EXqTT4AbLdyQIDAQABoyEw
HzAdBgNVHQ4EFgQU2rtogHKC0/
ws2dS3Zq7s9wwMofkwDQYJKoZIhvcNAQEFBQADgYEAK1jtcbRpYFAl2Bp9X02MaA/
igq3WXykizH7uQvrWgNQluf7ADbxaB7J96jaIN2GLQFxl6cbPwOvBIu7xd9a26eK6F5gq4iJKm7GeOgV5PZ4r
5umvSZgA0aLOAbhZ/
gwy40RauF0X+4I7JqamnV0DizM2YEDsFWKfTSvCy90ZizM=","MIICXjCCAcegAwIBAgIEYHXCUDANBgkqhki
G9w0BAQUFADBXMRMwEQYKCZImiZPyLGQBGRYDY29tMRYwFAYKCZImiZPyLGQBGRYGb3JhY2xlMRUwEwYKCZIm
iZPyLGQBGRYFY2xvdWQxETAPBgNVBAMTCENsb3VkOUNBMCAXDTE1MTExOTEyMDA0MloYDzIxMTUxMDI2MTEwM
DQyWjBXMRMwEQYKCZImiZPyLGQBGRYDY29tMRYwFAYKCZImiZPyLGQBGRYGb3JhY2xlMRUwEwYKCZImiZPyLG
QBGRYFY2xvdWQxETAPBgNVBAMTCENsb3VkOUNBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCXl3J0Xln
1SIanXnUCsvqovrTKdjbLYMR1orCejmX+zvigvVsz/l/
lMIkEzFM6fgeFFlBG6RjVu3b+44sinbIGBq56cCDZejk+MK5Sg/
K9vu8kCzQbLw0I1XlpoR06hX4Kk33f8ItzAtorX7fiONDuBp0i9/1Q3E0sSWTGooCoswIDAQABozUwMzASBgN
VHRMBAf8ECDAGAQH/
AgEAMB0GA1UdDgQWBBT8Km+50GihFXPqEmu6IbzsSBjH1zANBgkqhkiG9w0BAQUFAAOBgQA8nRvd8/
whkVX1MlXw/1C0/3fkWH5t7K4hoDz2ZRDvonQoAOrRDHJyDhX95T+QhQHRauspJhRzDn9eQmL6pL+42VC4i98
zIMuzoAeCNItFjFAAMm6nomZVPdMvOs3dsnwIEBDOJ3FAh4Pg8H9lxdKpmCtFyxewkm/
4UhCSaeFtow=="],"alg":"RS256","n":"i1b8rnvqha2scDOS8WjS7dkB98B5_59PspNo5j-5qWJM9cayrh
JNntkgoIMyvDE_YyeVVGjZRdDMISGfP52sQOSzwsxhFX2f-
NRj7RjQAxcrKahh5HbDb_jNnggZnuzA6fS956AbbdqXqhqsUrtAx8rdcIDa-qehF6k0-AGy3ck"}]}
```

3. Create certificate file for each certificate part of the above response by placing the same between *“-----BEGIN CERTIFICATE-----”* and *“-----END CERTIFICATE-----”*
4. Save first certificate as idcs.cert and second as idcs-ca.cert
5. Import certificate to KSS keystore of OWSM.

   Importing Certificate to KSS

```
wls:/jrfServer_domain/serverConfig/> svc = getOpssService(name='KeyStoreService')
wls:/jrfServer_domain/serverConfig/> svc.importKeyStoreCertificate(appStripe='owsm',
name='keystore', password='', alias='idcs', keypassword='',
type='TrustedCertificate', filepath='/scratch/ankianja/idcs.cert')
Already in Domain Runtime Tree
Certificate imported.
wls:/jrfServer_domain/serverConfig/> svc.importKeyStoreCertificate(appStripe='owsm',
name='keystore', password='', alias='idcs-ca.cert', keypassword='',
type='TrustedCertificate', filepath='/scratch/ankianja/idcs-ca.cert')
Already in Domain Runtime Tree
Certificate imported.
```

Configuring Trusted Issuers in Service Domain Trust

Trusted issuers are configured in service domain and trust entries are added in agent trust documents for trusted issuer by following the steps below:

1. Print the cert

```
keytool -printcert -file idcs.crt
Entry type: trustedCertEntry
Owner: CN=orclMT123232_idm, DC=cloud, DC=oracle, DC=com
Issuer: CN=Cloud9CA, DC=cloud, DC=oracle, DC=com
Serial number: 2df19c5c
Valid from: Fri Nov 2001:32:48PST 2015until: Mon Nov 1701:32:48PST 2025
Certificate fingerprints:
         MD5:  08:82:9E:3B:E1:2B:D3:0B:A1:9A:EC:32:1A:03:EC:05
         SHA1: 33:69:9A:16:6D:D5:62:53:01:3A:7D:C6:7A:D5:56:19:79:2B:2A:B9
         Signature algorithm name: SHA1withRSA
         Version: 3
Extensions:
#1: ObjectId: 2.5.29.14Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: DA BB 68807282D3 FC   2C D9 D4 B7 66AE EC F7  ..h.r...,...f...
0010: 0C 0C A1 F9                                        ....
]
]
```

Note:

- Use the owner of certificate obtained above in to configure the trusted issuers in OWSM. For example the owner here is - *'CN=orclMT123232_idm, DC=cloud, DC=oracle, DC=com'*
- Trusted Issuer will be *"https://identity.oraclecloud.com"* for IDCS.
- In scenarios where trust document with same name exists in domain, an error message shows while executing createWSMTokenIssuerTrustDocument('trust-doc',None), ignore it and proceed with next command.

2. Configuring trust document in OWSM using WLST

```
wls:/jrfServer_domain/serverConfig/> beginWSMSession()
Session started for modification.
true
wls:/jrfServer_domain/serverConfig/> createWSMTokenIssuerTrustDocument('trust-
doc',None)
New Token Issuer Trust document named "trust-doc" created.
To use the new document in the domain configuration,you must run the
setWSMConfiguration command where category = "TokenIssuerTrust", property name =
"name" and value = "trust-doc".
true
wls:/jrfServer_domain/serverConfig/> setWSMConfiguration(None, 'TokenIssuerTrust',
'name', None, ['trust-doc'])
A new property "name" within category "TokenIssuerTrust" has been added.
The values "[trust-doc]" have been added to property "name" within category
"TokenIssuerTrust".
Configuration properties associated with the context "/WLS/jrfServer_domain" has
been created.
true
wls:/jrfServer_domain/serverConfig/> selectWSMTokenIssuerTrustDocument('trust-doc')
Token Issuer Trust document named "trust-doc" selected in the session.
true
wls:/jrfServer_domain/serverConfig/> setWSMTokenIssuerTrust('dns.jwt','https://
identity.oraclecloud.com/',['CN=orclMT123232_idm, DC=cloud, DC=oracle, DC=com'])
New issuer - "https://identity.oraclecloud.com/" added to the document.
Issuer set with the given trusted keys.
The issuer and trusted DN values have been updated successfully.
true
wls:/jrfServer_domain/serverConfig/>
setWSMTokenIssuerTrustAttributeFilter('CN=orclMT123232_idm, DC=cloud, DC=oracle,
DC=com', 'user.tenant.name',['owsm'])
New TokenAttributeRule added for DN: CN=orclMT123232_idm, DC=cloud, DC=oracle,
DC=com.
true
wls:/jrfServer_domain/serverConfig/> commitWSMSession()
The tokenissuertrust trust-doc is valid.
```

```
Updating tokenissuertrust trust-doc in repository.

Session committed successfully.
true
```

3.6 Secure JAX-RS REST Client using OWSM OAuth 2.0 Security Policies

Secure REST Service using following Client side OWSM Policy.

Setup Keystore at Client Domain

keystore.sig.csf.key is used to sign the jwt token issued to OAuth server during request of access token. The default value is *orakey*.

Creating KSS in client domain

```
wls:/o ffline> connect('weblogic','gumby1234','10.229.140.110:11926')
Connecting to t3://10.229.140.110:11926 with userid weblogic ...
wls:/jrfServer_domain/serverConfig/> svc = getOpssService(name='KeyStoreService')
wls:/jrfServer_domain/serverConfig/> svc.createKeyStore(appStripe='owsm',
name='keystore', password='', permission=true)
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')
Keystore created
```

KeyPair is generated using KSS generateKeyPair Keystore Operations directly in KSS store.

Generating KeyPair using generateKeyPair

```
wls:/jrfServer_domain/serverConfig/> svc = getOpssService(name='KeyStoreService')
wls:/jrfServer_domain/serverConfig/> svc.generateKeyPair(appStripe='owsm',
name='keystore', password='', dn='CN=weblogic,OU=MT Orakey Test Encryption Purposes
Only,O=Oracle,C=US', keysize='2048', alias='orakey12212ssl', keypassword='')
Already in Domain Runtime Tree
Key pair generated
```

Import Sign Certificate from Client Domain to OAuth Client in OAuth Server

1. Export the signing certificate from KSS keystore. Use the alias with which the keypair is generated.

   Export the Certificate

   ```
   wls:/jrfServer_domain/serverConfig/> svc = getOpssService(name='KeyStoreService')
   wls:/jrfServer_domain/serverConfig/> svc.exportKeyStoreCertificate(appStripe='owsm',
   name='keystore', password='', alias='orakey12212ssl', keypassword='',
   type='TrustedCertificate',filepath='/scratch/ankianja/orakey12212nonssl.pem')
   Already in Domain Runtime Tree
   Certificate exported.
   ```

2. Go to OAuth Configuration and modify the client. Click on *"Trusted"* and upload the certificate file created above by clicking *"Import"* button.

Create OAuth Client csf key at Client Domain Credential Store

oauth2.client.csf.key is used for authentication with OAuth Server while requesting access token. The default value is *"basic.client.credentials"*.

Note:

Username and Password used for creation of password credential is the client id and secret obtained in *"Configuring OAuth Client on IDCS".*

Create OAuth Client Credential

```
wls:/jrfServer_domain/serverConfig/>
createCred(map="oracle.wsm.security",key="idcs.oauth2.client.credentials",user="38ef42fdb
92e467b935b138f6b02d211",password="0407ebb3-ec75-4cae-9132-8243ab3d853c",desc="OAuth
Client user for MT")
Credential created successfully.
```

Securing REST Client using OAuth Client OWSM WS Policies

The OAuth Client OWSM WS Policies for securing REST Client are as follows:

Policy	Description
`oracle/http_oauth2_token_over_ssl_idcs_client_policy`	This policy includes OAuth2 access token in the HTTP header. The access token is obtained from IDCS OAuth Server. This policy can be attached to any HTTP-based, SOAP or REST client, invoking service over ssl.
`oracle/oauth2_config_client_policy`	This policy provides OAuth2 Server information on the client side. This policy is enforced only when an OAuth2 token client policy is also attached. Otherwise, it is ignored. This policy is usually attached globally.

Attach oauth2_config_client_policy as GPA

Connect to WLST and run following commands to create GPA for oauth2 config policy.

Note:

Provide the value of OAuth2 server token endpoint as the "*token.uri*". In example below the *'token.uri'* is set to - `https://owsm.identity.c9dev0.oc9qadev.com:443/oauth2/v1/token`

```
wls:/jrfServer_domain/serverConfig/> beginWSMSession()
Session started for modification.
true
wls:/jrfServer_domain/serverConfig/> createWSMPolicySet('oauth-ps', 'rest-client',
'Domain("*")', 'Policy set forAll Rest Clients', 'true')
The policy set was created successfully in the session.
true
wls:/jrfServer_domain/serverConfig/> attachWSMPolicy('oracle/
oauth2_config_client_policy')
Policy reference "oracle/oauth2_config_client_policy" added.
true
wls:/jrfServer_domain/serverConfig/> setWSMPolicyOverride('oracle/
oauth2_config_client_policy','token.uri','https://owsm.identity.c9dev0.oc9qadev.com:443/
oauth2/v1/token')
The configuration override property "token.uri" having value "https://
owsm.identity.c9dev0.oc9qadev.com:443/oauth2/v1/token" has been added to the reference
```

```
to policy with URI "oracle/oauth2_config_client_policy".
wls:/jrfServer_domain/serverConfig/> setWSMPolicyOverride('oracle/
oauth2_config_client_policy', 'oauth2.client.csf.key', 'idcs.oauth2.client.credentials')
wls:/jrfServer_domain/serverConfig/> commitWSMSession()
INFO: Attachment of an oauth2 config policy without any oauth2 client policy is invalid.
Ensure you attach a valid oauth2 policy either via Direct Policy Attachment or via
Global Policy Attachment along with an oauth2 config policy.
The policy set oauth-ps is valid.
Creating policy set oauth-ps in repository.
Session committed successfully.
true
```

Give WSM Identity Permission to Client App

```
grantPermission(appStripe=None,codeBaseURL='file:${common.components.home}/modules/
oracle.wsm.common/wsm-agent-
core.jar',principalClass=None,principalName=None,permClass='oracle.wsm.security.WSIdentit
yPermission',permTarget='resource=idcsclientapp', permActions='assert')
```

Note:

Create user with clientId/clientsecret in webogic security realm in case of Client Only Flow.

Update your servlet client code with *http_oauth2_token_over_ssl_idcs_client_policy*

```
public void testJaxRsHttpOAuth2ClientCredsJwtMT(
Map<, String> config) {
String BASE_URI = "https://den01zxb.us.oracle.com:7002/idcsserviceapp/test/helloworld";
PropertyFeature scope = new PropertyFeature(
SecurityConstants.ConfigOverride.CO_SCOPE, "http://owsm/idcs_test");
PropertyFeature signCsfKey =
                  new PropertyFeature(SecurityConstants.ConfigOverride.CO_SIG_CSF_KEY,
"orakey12212ssl");

PolicyReferenceFeature[] clientPRF = new PolicyReferenceFeature[] {
                  new PolicyReferenceFeature("oracle/
http_oauth2_token_over_ssl_idcs_client_policy", scope, signCsfKey) };
    ClientConfig cc = new ClientConfig();
        cc.property(AbstractPolicyFeature.ABSTRACT_POLICY_FEATURE,new
PolicySetFeature(clientPRF));
    Client client = ClientBuilder.newClient(cc);
    WebTarget webTarget = client.target(BASE_URI);
    String res = webTarget.request("text/plain").header("X-RESOURCE-IDENTITY-SERVICE-
GUID","owsm").get(String.class);
    PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>IDCSClientServlet</title></head>");
        out.println("<body>");
        out.println("<p>The servlet has received a GET. This is the reply.</p>");
        out.println("<p>" + res + "</p>");
        out.println("</body></html>");
         out.close();
 }
```

4 Securing Services for Multiple Tenants

OWSM supports policy enforcement for multi tenant systems. OWSM can work with multiple tenant specific data-sources. OWSM enforces tenant specific security with GPA.

This chapter contains the following sections:

4.1 Use Case: Secure RESTful Web Services for Multiple Tenants

You can develop a RESTful web services and secure them for multiple tenants through tenant provisioning and enforcing tenant specific GPA.

Use Case
Secure RESTful Web Services for Multiple Tenants.

Implementation Summary
Develop a RESTful web service and secure it for two tenants (tenant1 and tenant2) through tenant provisioning and enforcing tenant specific GPA.

Components

- Oracle WebLogic Server
- Oracle Fusion Middleware
- Oracle Web Services Manager (OWSM)

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

- Developing RESTful Web Services

This use case includes the following steps:

- Create Weblogic domain and install templates.
- Provision tenants: Create database schema and data sources for tenants.
- Create GPA for tenants.
- Enforce policies for specific tenants at run time.

4.2 Implementing Web Services for Multiple Tenants - Prerequisites

Before implementing Web Services for multiple tenants, download and install product components, configure WebLogic domain, start the Remote Server, and get the access to Oracle Enterprise Manager Fusion Middleware Control and Oracle WebLogic Server Remote Console.

Before you begin, ensure the following:

- Download and install Oracle Fusion Middleware—includes OWSM.

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

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

4.3 Creating Weblogic Domain and Installing Templates

Create Weblogic domain using the following OWSM templates: `oracle.wsmpm_cloud_template.jar` or `oracle.wsmpm_cloud_file_template.jar`.

- **oracle.wsmpm_cloud_template.jar**: This template require the MDS db schema during domain creation.
- **oracle.wsmpm_cloud_file_template.jar**: This is a file based template, and it doesn't require the MDS db schema during domain creation. It creates OWSM seed documents inside domain home. This is the preferred template.

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

To know the procedure to create the above domain templates, see Creating and Using a Domain Template (Offline) in *Understanding the WebLogic Scripting Tool*

4.4 Provisioning Tenants

Install MDS schema in db/pdb's for a particular tenant. Then, create a tenant specific data source into the weblogic domain for the MDS schema created.

This chapter contains the following sections:

- Creating Database Schemas using RCU
- Creating Data Source for Tenants
- Setting up Data Sources for Tenants

4.4.1 Creating Database Schemas using RCU

Create database schemas using RCU. RCU is available only on Linux and Windows platforms. Use Linux RCU to install for UNIX supported databases and Windows RCU for supported Windows databases.

Note:

See Using the -silent Command in the *Creating Schemas with the Repository Creation Utility* to run RCU with `—silent` mode from the command line to have minimal interactions.

Use the following procedure to create database schemas for tenant1 and tenant2.

To create database schema for a tenant, for example tenant1:

1. Navigate to the directory into which RCU is installed. For example:

   UNIX: `$rcuHome/bin/rcu`

   Windows: `<rcuHome>\BIN`
2. Start the application.

   UNIX: `./rcu`

   Windows: `rcu.bat`

   The **Repository Creation Utility** welcome screen appears.
3. Click **Next**. The **Create Repository** page appears.
4. Make sure **Create Repository** is selected, then click **Next**. The **Database Connection Details** page appears. For simplicity, many users use their sys name here. Enter database details in the fields provided (Database Type, Host Name, Port, Service Name, Username, Password, Role).

   Database Type
   Select the database type as Oracle Database.

   Host Name
   Specify the name of the server where your database is running in the following format: `host.example.com`.

   Port
   Specify the port number for your database.

   Service Name
   Specify the service name for the database. For example, `rdbms.host.example.com`.

   Username
   Enter the user name for your database. The default user name is SYS.

   Password
   Enter the password for your database user.

   Role
   Select the database user's role from the drop-down list: Normal or SYSDBA.

5. Click **Next**. The Installer checks the prerequisites. When the prerequisite checks are complete, click **OK**. Click **Next**. The **Components** screen appears.
6. Click the Create a new prefix option and choose a schema prefix (for example: `tenant1`).
7. Select the accompanying components **Metadata Services** and **Oracle Platform Security Services**, in addition to the components selected by default. Click **OK**.
8. Click **Next**. The **Schema Passwords** page appears.
9. Select **Use same passwords** for all schemas.
10. Enter your password in the field provided and confirm it.
11. Click **Next**. The **Map Tablespaces** page appears. Click **Manage Tablespaces**, if you want to modify existing tablespaces.
12. For this installation, click **Next**. A **Repository Creation** notification appears. Click **OK**. Tablespaces are created, and the progress will be displayed in a pop-up notification. When the operation is completed, click **OK**. The **Summary** page appears.
13. Click **Create**. The schema is created. A **Completion Summary** screen appears.
14. Click **Close**.

It creates database schema for tenant1. Repeat the above procedure to create database schema for tenant2.

4.4.2 Creating Data Source for Tenants

Use the Oracle WebLogic Server Remote Console to set up a JDBC data source in the WebLogic Server instance for your applications.

Use the following procedure to create data sources for two tenants: tenant1 and tenant2

To configure data source for a particular tenant, tenant1 for example:

1. Log in to the **Oracle WebLogic Server Remote Console**.
2. In the WebLogic Server Remote Console page, select **JDBC** > **Data Sources**. Click **New**.
3. In the **JDBC Data Source Properties** page:
   - In the **Name** field, enter the name of the unique JDBC data source. For example, `mds-owsm-tenant1`.
   - In the **JNDI** field, enter the name of the connection. For example, `mds/owsm/tenant1`.
   - For the **Database Type**, select `Oracle`.
   - For the **Database Driver**, select `Oracle Driver (thin)`
4. Click **Next**.
5. In the **Transactions Options** page, accept the default options and click **Next**.
6. In the **Connection Properties** page:
   - For **Database Name**, enter the Oracle SID. For example, rdbms.example.com.
   - For **Host Name**, enter the machine name of the database. For example, host.example.com
   - Enter the port number used to access the database.
   - Enter the user name as the Schema name created earlier, for example `TENANT1_MDS`. Enter the schema password for the database. Click **Next**.

7. In the **Test Database Connection** page, click **Test Configuration** to test the connection.
8. In the **Select Targets** page, select the server for which the JDBC data source is to be deployed.
9. Click **Finish**.

It creates data source for tenant1. Repeat the above procedure to create a data source for tenant2. Once the data source has been created in Oracle WebLogic Server, it can be used by an application module.

4.4.3 Setting up Data Sources for Tenants

Generic data sources are data sources that manage the actual database connections, with its underlying connection pool. As part of service instance provisioning, when a new service instance is created in a POD, generic data sources are created. Once all generic data sources are created, update the Switching properties of the proxy datasource.

The switching properties use the following format:

```
serviceInstanceId1=datasourceJNDIName1;serviceInstanceId2=datasourceJNDIName2;
```

Example: `serviceInstance1=mds/owsm/tenant1;serviceInstance2=mds/owsm/tenant2;`

Update Switching properties for Weblogic Server

Generic data sources can be created using WLST scripting shell or JMX. Once the data sources are created, follow the steps below to update proxy data source switching properties using WLST:

1. Add a new script UpdateSwitchingPropertiesProxyDatasource.py to your local service instance provisioning.
2. Copy the script provided in the code block below and make the following changes:
   - Update weblogic admin server credentials, replacing username/password with your weblogic server credentials: connect("**weblogic**","**password**","t3://"+example+":7001").
   - Update switching properties of a proxy datasource, changing updateProps provided the service instance Id and the underlying data source JNDI name.

     `updatedProps = switchingProps + '`**`;serviceInstance1=jdbc/instancejndi`**`'`

```
"""
Copyright (c) 2016, 2017, Oracle and/or its affiliates. All rights reserved.
This script updates Switching Properties of a Proxy JDBC Datasource.
"""

import sys, socket
import os
hostname = socket.gethostname()
connect("weblogic","password","t3://"+hostname+":7001")
edit()
startEdit()
cd ('/JDBCSystemResources/ProxyDS')
jdbcResource = cmo.getJDBCResource()
jdbcDataSourceParams = jdbcResource.getJDBCDataSourceParams()
switchingProps = jdbcDataSourceParams.getProxySwitchingProperties()
if (switchingProps != None):
        updatedProps = switchingProps + ';serviceInstance1=jdbc/instancejndi'
else:
        updatedProps = 'serviceInstance1=jdbc/instancejndi'
```

```
jdbcDataSourceParams.setProxySwitchingProperties(updatedProps)
save()
activate()
```

3. Execute the following command:

   ```
   java weblogic.WLST -i UpdateSwitchingPropertiesProxyDatasource .py
   ```

Update Switching properties for JavaSE application

Once the data sources are created, follow the steps below to update proxy data source switching properties:

- Edit the application server corresponding to context.xml and update switchingProperties value adding **';serviceInstanceId=jdbc/instancejndi'** to the existing value.

  Example: `switchingProperties="1e8b59b2=jdbc/mds/owsm/tenant1;2efba34c=jdbc/mds/owsm/tenant2"`

4.5 Creating OWSM Security Artifacts

You can create OWSM security artifacts by using either online WLST commands or REST API.

See the following sections:

- Creating OWSM Security Artifacts by using WLST
- Creating OWSM Security Artifacts by using REST API

4.5.1 Creating OWSM Security Artifacts by using WLST

You can create OWSM security artifacts by using online WLST commands.

Create security artifacts for tenant1 and tenant2.

To create security artifacts for tenant1 with the `oracle/multi_token_rest_service_policy policy`, run the following WLST commands in sequence:

```
connectWSMRest('weblogic','password','example.com:22001')
startWSMTenantContext('tenant1')
beginWSMSession()
createWSMPolicySet('myPolicySet','rest-resource','Domain("*")')
attachWSMPolicy("oracle/multi_token_rest_service_policy")
commitWSMSession()
listWSMPolicySets()
endWSMTenantContext()
```

Note:

If you want to create any OWSM security artifacts like Global Policy Set, Token Issuer Trust, Configuration Customization, etc., then at first, run the connectWSMRest() command, and then wrap the corresponding OWSM commands between the startWSMTenantContext() and endWSMTenantContext() commands.

Note:

To see details about the commands see Web Services Custom WLST Commands in *WLST Command Reference for Infrastructure Components*.

To create security artifacts for tenant2 with the `oracle/http_jwt_token_service_policy` policy, run the following WLST commands in sequence:

```
connectWSMRest('weblogic','password','example.com:22001')
startWSMTenantContext('tenant2')
beginWSMSession()
createWSMPolicySet('myPolicySet','rest-resource','Domain("*")')
attachWSMPolicy("oracle/http_jwt_token_service_policy")
commitWSMSession()
listWSMPolicySets()
endWSMTenantContext()
```

4.5.2 Creating OWSM Security Artifacts by using REST API

While running REST Commands for creating OWSM artifacts, pass the tenant id value in the X-RESOURCE-IDENTITY-SERVICE-GUID header.

REST API to create OWSM policy set for tenant1:

```
curl -i -H "X-RESOURCE-IDENTITY-SERVICE-GUID:tenant1" -H "Content-Type:application/json"
-u username:password -X POST -d @- http://example.com:22001/wsm-pmrest/v2/policyset
[{
  "name": "myPolicySet",
  "type": "rest-resource",
  "scope": "Domain('*')",
  "policyReferences":[{
                     "uri":"oracle/multi_token_rest_service_policy",
                      "status":"enabled"
                      }
                      ]
}]
```

REST API to create OWSM policy set for tenant2:

```
curl -i -H "X-RESOURCE-IDENTITY-SERVICE-GUID:tenant2" -H "Content-Type:application/json"
-u username:password -X POST -d @- http://example.com:22001/wsm-pmrest/v2/policyset
[{
  "name": "myPolicySet",
  "type": "rest-resource",
  "scope": "Domain('*')",
  "policyReferences":[{
                        "uri":"oracle/http_jwt_token_service_policy",
                        "status":"enabled"
                        }
                        ]
}]
```

4.6 Enforcing Tenant Specific Policies at Runtime

Enforce tenant specific security policies during runtime by using the tenant id in the http header.

During runtime, OWSM enforces policy for a particular tenant, by reading the tenant id value received through http header (i.e. X-RESOURCE-IDENTITY-SERVICE-GUID).

5
Securing RESTful Web Services Using Basic Authentication

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to secure RESTful web services using basic authentication.

This chapter contains the following sections:

5.1 Use Case: Secure a RESTful Web Service Using Basic Authentication

You can develop a RESTful web service and secure it by attaching an Oracle Web Services Manager (OWSM) basic authentication policy.

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 RESTful Web Services
- Securing RESTful Web Services and Clients
- "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 Remote Console.
- Verify the `HelloWorld` web service using a browser.

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

To implement RESTful web service using basic authentication, develop a RESTful web service and secure it by attaching an Oracle Web Services Manager (OWSM) basic authentication policy.

To implement this use case, complete the following steps in sequence:

- 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

5.2.1 Implementing RESTful Web Service Using Basic Authentication- Prerequisites

Before implementing RESTful Web Service by using basic authentication, download and install product components, configure WebLogic domain, start the Remote Server, and get the access to Oracle Enterprise Manager Fusion Middleware Control and Oracle WebLogic Server Remote Console.

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 Remote 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 Remote Console

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

5.2.2 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', 'password', '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.
```

5.2.3 Creating a RESTful Web Service

You can create a simple HelloWorld RESTful web service by using JDeveloper.

Procedure:

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*

5.2.4 Authenticating the User Using SecurityContext

You can authenticate a user by 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 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 + "!";
    }
}
```

5.2.5 Packaging With an Application Subclass

You can 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;
    }
}
```

5.2.6 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*.

5.3 Verifying the Use Case: RESTful Web Service

You can verify a RESTful web service from a browser. You can test basic and advanced features of your web service by using the Web Services Test Client or Test Web Service page in Fusion Middleware Control.

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 **password**.

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*.

5.4 Additional Resources for RESTful Web Services Use Case

Additional resources that provide 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 RESTful Web Services
- Securing RESTful Web Services and Clients
- "Developing and Securing RESTful Web Services" in *Developing Applications with Oracle JDeveloper*

6 Propagating Security Identity with RESTful Web Services

You can 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.

This chapter contains the following sections:

6.1 Use Case: Propagate Security Identity with RESTful Web Services

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to propagate security identity with RESTful web services.

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.

6.2 Implementing Use Case: Propagating Security Identity with RESTful Web Services

To implement the Propagating Security Identity with RESTful Web Services use case, you need to perform the following tasks in sequence: create, secure, and deploy a RESTful web service and a RESTful client, create a test user, and set up the Keystore Service (KSS).

- 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)

6.2.1 Propagating Security Identity with RESTful Web Services - Prerequisites

Before implementing the use case, download and install product components, configure WebLogic domain, start the Remote Server, and get the access to Oracle Enterprise Manager Fusion Middleware Control and Oracle WebLogic Server Remote Console.

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 Remote 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 Remote Console

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

6.2.2 Create, Secure, and Deploy a RESTful Web Service

To create, secure, and deploy a RESTful web service, perform the following tasks in sequence: create a RESTful web service, authenticate the user by using SecurityContext, modify the servlet name for the web project, and secure and deploy the 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

6.2.2.1 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";
    }
}
```

6.2.2.2 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;
  ```

```
            }
        }
```

6.2.2.3 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>
```

6.2.2.4 Securing the RESTful Web Service

To secure RESTful web services, you can attach one of the OWSM predefined security policies described in "OWSM Policies Supported for RESTful Web Services and Clients" 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*.

6.2.2.5 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*.

6.2.2.6 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 **password** 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*.

6.2.3 Create, Secure, and Deploy a RESTful Client

After deploying a RESTful web service, you should create, secure, and deploy a RESTful client to implement the use case of propagating security identity.

- 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

6.2.3.1 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*.

6.2.3.2 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();
   }
}
```

6.2.3.3 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>
```

6.2.3.4 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>
```

6.2.3.5 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*.

6.2.3.6 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 **password**.

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

6.2.4 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.

6.2.4.1 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.

6.2.4.2 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**

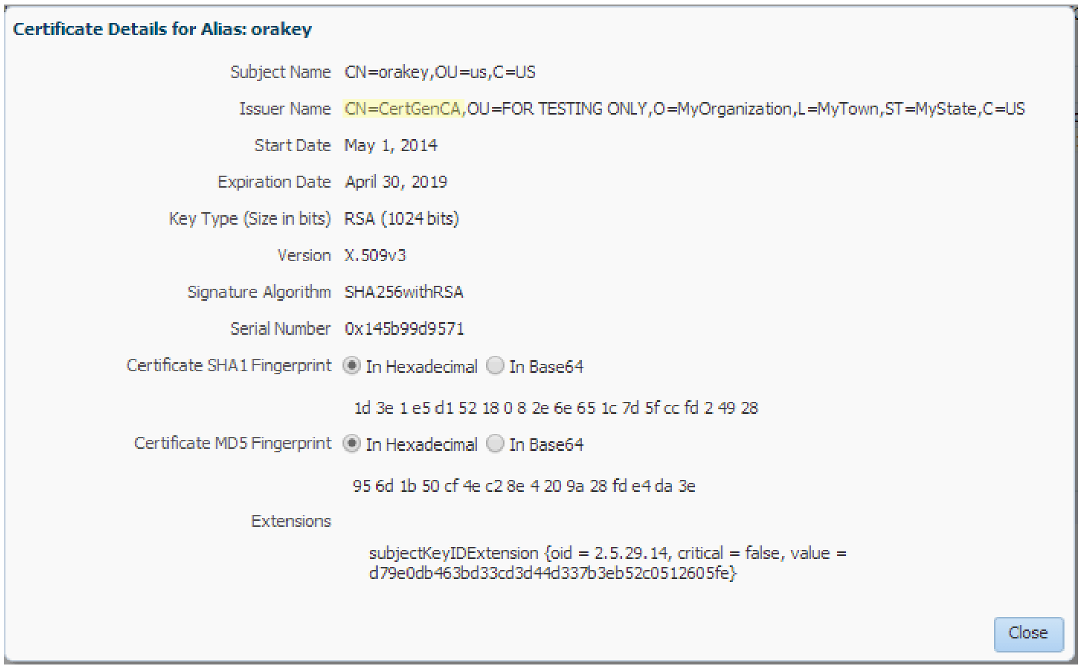
   For the complete procedure, see "Using the OPSS Keystore Service for Message Protection" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

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 6-1   Certificate Details for Alias: orakey Dialog



Certificate Details for Alias: orakey

Subject Name	CN=orakey,OU=us,C=US
Issuer Name	CN=CertGenCA,OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US
Start Date	May 1, 2014
Expiration Date	April 30, 2019
Key Type (Size in bits)	RSA (1024 bits)
Version	X.509v3
Signature Algorithm	SHA256withRSA
Serial Number	0x145b99d9571
Certificate SHA1 Fingerprint	● In Hexadecimal ○ In Base64
	1d 3e 1 e5 d1 52 18 0 8 2e 6e 65 1c 7d 5f cc fd 2 49 28
Certificate MD5 Fingerprint	● In Hexadecimal ○ In Base64
	95 6d 1b 50 cf 4e c2 8e 4 20 9a 28 fd e4 da 3e
Extensions	
	subjectKeyIDExtension {oid = 2.5.29.14, critical = false, value = d79e0db463bd33cd3d44d337b3eb52c0512605fe}

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*.

6.2.5 Creating a Test User

After deploying a RESTful web service and client, create a test user.

To create a test user:

1. Invoke the WebLogic Server Remote 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 **password**.

   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.

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

You can verify the Propagating Security Identity with RESTful Web Services use case from a browser. You can test basic and advanced features of your web service by using the Web Services Test Client or Test Web Service page in Fusion Middleware Control.

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 **password** or **testuser** and **password**.

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*.

7

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

You can configure web services federation with Microsoft ADFS 2.0 STS as the Identity Provided STS (IP-STS) and OWSM as the Relying Party (RP-STS).

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

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

Components

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

Additional Resources on Oracle Web Services Manager

- *Overview of Oracle Web Services Manager*
- *Securing Web Services*
- *Managing and Troubleshooting Oracle Web Services Manager*
- Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

This use case demonstrates the steps required to:

- Attach the appropriate OWSM security policies to enforce message-level protection using SAML bearer authentication. You must attach the following service policy :

  `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 OWSM is used as the RP-STS.

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

For more information on how to implement this use case, see Use Case: Implementing Web Services federation with Microsoft ADFS 2.0 STS as IP-STS and OWSM as RP-STS.

7.1 Use Case: Implementing Web Services federation with Microsoft ADFS 2.0 STS as IP-STS and OWSM as RP-STS

To implement the use case, complete the following tasks in sequence: configure OWSM as the RP-STS, configure Microsoft ADFS 2.0 STS as the IP-STS, and configure the Web Service Client.

- Generating Federation Metadata Document for the RP-STS
- Configuring the Web Service
- 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 Microsoft ADFS 2.0 STS is provided. For detailed information about how to perform these configuration steps, refer to the documentation:`http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

7.1.1 Generating Federation Metadata Document for the RP-STS

You must generating a federation metadata document for the RP-STS using the `exportFederationMetadata` command or the REST API.

To generate an unsigned federation metadata document using the WLST command, do the following:

1. Connect to the running instance of the server in the domain for which you want to generate the document as described in Accessing the Web Services Custom WLST Commands in *Administering Web Services*.
2. Run the `exportFederationMetadata` command to generating an unsigned federation metadata document.

   ```
   exportFederationMetadata(federationFile, metadataType, issuer, signMetadata ,
   [signAliases=None], [encAliases=None])
   ```

   In the following example, unsigned federation metadata document is generated for Service provider and the role descriptor does not have an encryption key.

   ```
   wls:/wls-domain/serverConfig> exportFederationMetadata('/home/ABC/Downloads/
   FederationMetadata.xml','SP','www.example.com')
   ```

   This is URL for the service.

   See, exportFederationMetadata in *WLST Command Reference for Infrastructure Components*

   To generate an unsigned federation metadata document using the REST API, see Export Federation Metadata Document Method in *Oracle Fusion Middleware REST API for Managing Credentials and Keystores with Oracle Web Services Manager*.

7.1.2 Configuring the Web Service

To implement the use case configure web services federation with Microsoft ADFS 2.0 STS as the Identity Provided STS (IP-STS) and Web Service as the Relying Party (RP-STS)., first you need to configure 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 and configure the WS-Trust for the Relying Party (RP-STS) in OWSM. To do so, run the WLST command:
   a. Connect to the running instance of the server in the domain for which you want to generate the document as described in Accessing the Web Services Custom WLST Commands in *Administering Web Services*.
   b. Run the `importFederationMetadata` command to import the signing certificate for the Microsoft ADFS 2.0 STS endpoint into the OWSM keystore and configure the WS-Trust for the Relying Party (RP-STS).

      ```
      importFederationMetadata(federationFile,nameIdAttribute=None,
      [filterValues=None],userAttribute=None,userMappingAttribute=None)
      ```

      For example:

      ```
      wls:/wls-domain/serverConfig> importFederationMetadata('https://example.com/
      FederationMetadata/2007-06/Federation.xml',"Unique_name",['filter'],'mail','uid')
      ```

      This is the federation metadata document URL of Microsoft ADFS 2.0 STS.

      For more information see, importFederationMetadata in *WLST Command Reference for Infrastructure Components*
3. Define the OWSM 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*.

7.1.3 Configuring Microsoft ADFS 2.0 STS as the IP-STS

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, you need to configure Microsoft ADFS 2.0 STS as the IP-STS.

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`.)

Perform the following steps:

1. From the AD FS 2.0 console, expand **Trust Relationships**, right-click the **Relying Party Trusts folder** and then select **Add Relying Party Trust** to open the **Add Relying Party Trust Wizard**.
2. Confirm that the endpoint is enabled.
3. Add the OWSM instance acting as the IP-STS as a relying party using the ADFS 2.0 management console.
   a. On the **Select Data Source** page, click **Import data about the relying party from a file**, and then click Next.

  b. Click Browse and navigate to the directory where the federation metadata file is located.

4. Configure ADFS 2.0 STS for claims-based authentication using the ADFS 2.0 management console.
   a. On the **Select Rule Template** page, select the option**Send LDAP Attributes as Claims** as the rule type.
   b. On the **Configure Rule** page, enter Name ID as the **Claim rule name**, select Active Directory option as the **Attribute store**, SAM-Account-Name as the **LDAP Attribute**, and Name ID as the **Outgoing Claim Type**.

7.1.4 Configuring the Web Service Client

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, finally you need to configure the web service client.

To configure the web service client:

1. Ensure that you have create JAX-WS Client Application. For more information, see Creating JAX-WS Web Services and Clients in the *Developing Applications with Oracle JDeveloper*.
2. Creating a Web Service Proxy using JDeveloper by completing the following steps:
   a. Right-click the JAX-WS Client Application you have created and select **New** and then **From Gallery** .
   b. In the New Gallery, expand the Business Tier node and select **Web Services** in the Categories list. Select the **Web Service Client and Proxy** item and click **OK**.
   c. The **Create Web Service Client and Proxy** page is displayed.
   d. In the **Select Web Service Description** page, specify the location of the WSDL service (For example: `https://www.example.com:8002/JaxWsWssStsIssuedBearerTokenWithADFSWssUNOverSsl/JaxWsWssStsIssuedBearerTokenWithADFSWssUNOverSslService?WSDL`) and select Copy WSDL Into Project and click Next.
   e. In the **Asynchronous Methods** page, select **Don't generate any asynchronous methods** and click **Finish**.
3. Attach the policy `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_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 ADFS 2.0 STS endpoint. For example:

   ```
   http://http://m1.example.com/adfs/services/trust/13/usernamemixed
   ```
4. Create a policy from `oracle/sts_trust_config_service_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*.

5. 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 WSDL Uri to the Web Service 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*.

8

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

You can refer to the use case description, solution summary, components involved, and the linked documentation resources 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).

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

Additional Resources on Oracle Web Services Manager

- *Overview of Oracle Web Services Manager*
- *Securing Web Services*
- *Managing and Troubleshooting Oracle Web Services Manager*
- Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

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.

For more information on how to implement this use case, see Use Case: Implementing Web Services federation with Microsoft ADFS2.0 STS.

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

To implement the use case, complete the following tasks in sequence: configure the Web Service, configure Oracle STS as the RP-STS, configure Microsoft ADFS 2.0 STS as the IP-STS, and configure the Web Service Client.

- 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 Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

8.1.1 Configuring the Web Service

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, first you need to configure 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*.

8.1.2 Configuring Oracle STS as the RP-STS

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, you need to configure Oracle STS as the RP-STS.

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

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.

8.1.3 Configuring Microsoft ADFS 2.0 STS as the IP-STS

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, you need to configure Microsoft ADFS 2.0 STS as the IP-STS.

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`.)

Perform the following steps:

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.

8.1.4 Configuring the Web Service Client

To implement the use case Web Services federation with Microsoft ADFS2.0 STS, finally you need to configure 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*.

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

You can refer to the use case description, solution summary, components involved, and the linked documentation resources 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).

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

Additional Resources on Oracle Web Services Manager
Additional resources provides more information about the technologies and tools used to implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS.

- *Oracle Web Services Manager Predefined Policies*
- *Wss11 Issued Token with Saml Holder of Key with Message Protection Client Policy*
- Microsoft ADFS 2.0 STS: `http://technet.microsoft.com/en-us/library/adfs2(v=ws.10).aspx`

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.

For more information on how to implement this use case, see Use Case: Implementing Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS.

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

To implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS: first configure the web service, then configure Microsoft ADFS 2.0 STS as the RP-STS, followed by configuring Oracle STS as the IP-STS, and in the end configure the Web Service Client.

- 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

9.1.1 Configuring the Web Service

To implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS, first you need to configure 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*.

9.1.2 Configuring Microsoft ADFS 2.0 STS as the RP-STS

To implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS, after configuring the web service, you need to configure Microsoft ADFS 2.0 STS as RP-STS.

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

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

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.

9.1.3 Configuring Oracle STS as the IP-STS

To implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS, after configuring the web service and RP-STS, you need to configure Oracle STS as the IP-STS.

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

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.

9.1.4 Configuring the Web Service Client

To implement the use case configuring web services federation with Oracle STS as IP-STS and Microsoft ADFS 2.0 STS as RP-STS, finally you need to configure 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.

10 Configuring SAML HOK Using WS-Trust with OpenSSO STS

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to configure SAML holder-of-key (HOK) with message protection using WS-Trust with OpenSSO STS.

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

Additional Resources on Oracle Web Services Manager

- *Oracle Web Services Manager Predefined Policies*
- *Overview of SAML Holder of Key and SAML Bearer as Issued Tokens*
- `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 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.

For more information on how to implement this use case, see Configuring SAML HOK Using WS-Trust with OpenSSO STS

10.1 Configuring SAML HOK Using WS-Trust with OpenSSO STS

To implement the use case configuring SAML HOK with message protection using WS-Trust with OpenSSO STS, first configure OpenSSO STS and then configure SAML HOK by using WS-Trust with OpenSSO STS.

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

10.1.1 Configuring OpenSSO STS to Implement SAML HOK

To implement the use case configuring SAML HOK with message protection using WS-Trust with OpenSSO STS, first configure 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 **password**.
5. Under the On Behalf of Token section, select **IdapService** 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, consider the Certificate Management Server maintained by the OpenSSO QA team at https://example.com.

      c. Access the Certificate Management Server at `https://example.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://example.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.

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

After configuring OpenSSO STS, configure 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/ wss11_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 `password`. 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*.

11

Configuring SAML Sender Vouches Using WS-Trust with OpenSSO STS

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to configure SAML sender vouches using WS-Trust with OpenSSO STS.

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

Additional Resources on Oracle Web Services Manager

- *Understanding SAML Sender Vouches as Issued Tokens*
- *About SAML Configuration*
- `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 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.

For more information on how to implement this use case, see Use Case: Implementing SAML Sender Vouches Using WS-Trust with OpenSSO STS.

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

To implement the use case first configure OpenSSO STS, and then configure SAML sender vouches message protection using WS-Trust with OpenSSO STS.

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

11.1.1 Configuring OpenSSO STS to Implement SAML Sender Vouches

To implement the use case Configuring SAML Sender Vouches Using WS-Trust with OpenSSO STS, first configure Open SSO 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 **password**.
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://example.com`.

    c. Access the Certificate Management Server at `https://example.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://example.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.

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

After configuring OpenSSO STS, configure 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 `password`.

> **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*.

12

Configuring SAML Bearer Using WS-Trust with OpenSSO STS

You can refer to the use case description, solution summary, components involved, and the linked documentation resources to configure SAML bearer using WS-Trust with OpenSSO STS.

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

Additional Resources on Oracle Web Services Manager

- *Overview of Oracle Web Services Manager*
- *Securing Web Services*
- *Managing and Troubleshooting 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 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.

For more information on how to implement this use case, see Use Case: Implementing SAML Bearer Using WS-Trust with OpenSSO STS.

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

To implement the use case configure OpenSSO STS, and then configure SAML bearer message protection using WS-Trust with OpenSSO STS.

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

12.1.1 Configuring OpenSSO STS to Implement SAML Bearer

To implement the use case SAML Bearer Using WS-Trust with OpenSSO STS, first configure 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 **password**.
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.

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

After configuring the OpenSSO STS, configure 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 `password`. 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*.