Using SAML 1.1 Bearer Assertion for Authentication CaseUsing SAML 1.1 Bearer Assertion for Authentication Case

Files Prepare Run Troubleshooting Related Topics

About the Example

This is an example of SAML 1.1 Bearer with a standalone client. It demonstrates the minimum configuration and setup with WS-Trust to use a SAML assertion for authentication of a Web service application.

This example contains two WebLogic Server instances, which host the FlightService Web service and STSHttpsUNT.java, which is used as a Security Token Service (STS). The client TravelAgencyClient does not have a public/private key pair and gets the SAML token from the STS for authentication, with transport level message protection.

Overview of the Example

The example contains three parts: TravelAgencyClient, the remote STS, and the Web service FlightService.

The client TravelAgencyClient functions as follows:

- With the policy Wssp1.2-2007-Https-UsernameToken-Plain.xml, sends a WS-Trust Request Security Token (RST) message with Username token to the remote STS using HTTPS for message protection.
- Gets a SAML 1.1 Bearer assertion from the remote STS.
- Uses the SAML 1.1 Bearer assertion for authentication between TravelAgencyClient and FlightService.
- Calls FlightServcie.getName() to exchange string messages between TravelAgencyClient and FlightService.

The remote STS (STSHttpsUNT.java) uses the client side policy of Wssp1.2-2007-Https-UsernameToken-Plain.xml, which is an HTTPS transport level policy that requires a Username token for authentication. The remote STS receives a WS-Trust Request Security Token (RST) message, authenticates the user, and issues a SAML assertion in the Request Security Token Response (RSTR) message to the client.

The Web service FlightService.java has the predefined policy Wssp1.2-2007-Saml1.1-Bearer-Https.xml attached. FlightService.java receives the call from the client, authenticates the user by verifying the SAML 1.1 Bearer assertion, and returns a message to the client.

Invoking Remote Security Token Services

The STS provided in this sample is for demonstration of WS-Trust functionality only. It is not a supported product from WebLogic Server and has several limitations.

The standards-based WS-Trust protocol is used for the request and response messages between the client and remote STS. Therefore, the remote STS can be replaced with any WS-Trust based STS. In actual use involving SAML tokens, the remote STS could be .NET implementation or Oracle OpenSSO STS.

To direct the WS-Trust messages to another remote STS, all you need to do is to change the STS endpoint address. You can do this in TravelAgencyClient by changing the following code line:

```
context.put(WLStub.WST_STS_ENDPOINT_ON_SAML, "https://mysts.mycompany.com:7002/sts/usertokenoverssl");
```

or change samlStsURL in the properties.txt configuration file:

```
samlStsURL=https://mysts.mycompany.com:7002/sts/usertokenoverssl
```

Configuration and Setup

As described in Files, the WebLogic security providers are created using WLST scripts located in the config directory.

The Ant task wlserver defined in server.xml enables you to start, reboot, shut down, or connect to a WebLogic Server instance. The server instance may already exist in a configured WebLogic Server domain, or you can create a new single-server domain for development by using the generateconfig=true attribute.

The jwsc, wldeploy and clientgen ant tasks are also used in this example. For more information about these tasks, refer to the Web service example **Creating JAX-WS Web Services for Java EE** (listed under the **WebLogic Server Examples > Examples > API> Web Services** node in the Samples Viewer).

Viewing the Example Configuration in the WebLogic Server Administration Console

After you deploy this sample application as described in Build and Deploy the Example, you can use the WebLogic Server Administration Console to view the configuration.

- To see the deployed Web service and the attached policies: Deployments -> flightservice -> Expand Plus sign -> FlightService -> Configuration -> WS-Policy.
- To see the Web services security settings for the domain as set in configWss.py: domain (wl_server) -> Web Services Security -> default_wss. The credential provider is default_x509_cp. The token handlers are default_ut_handler and default_x509_handler.
- To see the credential mapping providers added by the example for the security realm as set in configWss.py: Security Realms -> myrealm -> Providers -> Credential Mapping. The providers are DefaultCredentialMapper, PKICredentialMapper, and SamlCredentialMapperV2.
- To see the credential mappings for the realm: Security Realms -> myrealm ->Credential Mappings -> PKI. Note the x509 credentials supplied to PartnerService map to WebLogic Server principal Alice.
- By default, the separate WebLogic Server server for the STS is located at http://localhost:7011. The WebLogic Server Administration Console for this server is available at http://localhost:7011/console/. (The user name and password for the console are in the properties.txt file.) To see the STS Web service and the attached policies: Deployments -> standalonests -> Expand Plus sign -> STSHttpsUNTService -> Configuration -> WS-Policy.
- To see the users added by this example: Security Realms -> myrealm -> Users and Groups. Note the users Alice, Cindy, and Bob.

Files Used in the Example

Directory Location:
MW_HOME/wlserver_12.1/samples/server/examples/src/examples/webservices/saml/bearer11ssl
(where *MW_HOME* is the directory containing your WebLogic Server installation)

File Click source files to view code.	**Description**
FlightService.java	The Web service that requires a SAML 1.1 Bearer token for authentication.
TravelAgencyClient.java	The standalone client application that interacts with the remote STS with username token and gets the SAML 1.1 Bearer token returned.
STSHttpsUNT.java	Remote STS code that uses WS-SecureConversation's STS to issue SAML Token.
Wssp1.2-2007-Https-UsernameToken-Plain.xml	Policy of HTTPS transport level policy using Username Token for authentication.
build.xml	Ant build file that contains targets for building, deploying, and running the example.
server.xml	Ant build file imported in the build.xml which contains targets for creating, starting, stopping and removing the STS server.
wss-config.xml	Ant build file imported in the server.xml which contains targets for configuring the Examples domain.
sts-config.xml	Ant build file imported in the server.xml which contains targets for configuring the STS domain.
config/addApiAssertingPartySVRecieverV2.py	WLST script that adds SAML 1.1 Asserting party for the receiver in SAML Identity Asserter
config/addApiProvidersSVRecieverV2.py	WLST script that disables the SAML Identity Assertion and creates a new SAML Identity Assertion V2 provider
config/addApiProvidersSVSenderV2.py	WLST script that disables the SAML Credential Mapping Provider and creates a new SAML Credential Mapping Provider Version 2. SAML Credential Mapping Provider Version 2 provides greatly enhanced configuration options and is recommended for new deployments. SAML Credential Mapping Provider Version 1 is deprecated in WebLogic Server 9.1.
config/addApiRellyingPartySVSenderV2.py	WLST script that configures the SAML Credential Mapping Provider Version 2 by adding FlightService.java Web service as a Relying Party.
config/configWLSSecurity.py	WLST script that configures server's security by using certs/oasis.jks as the Custom Identity KeyStore, certs/cacerts as the Custom Trust KeyStore, and so forth.
config/configWss.py	WLST script that configures WS-Security, including setting PKI Credential Mapper, creating default WebServcieSecurity, creating credential provider for x.509, customing keystore for XML encryption and digital signature, creating token handler for x509 token, configuring UsernameToken Digest and enabling SSL. Remember to restart the WebLogic Server after executing this script
config/setPKICredMapper.py	WLST script that sets PKI Credential Mapper for Alice and Bob on the server side
certs/	Refer to the Web service example **Using SAML 2.0 Assertion for Authentication and Authorization Case** (listed under the **WebLogic Server Examples > Examples > API> Web Services** node in the Samples Viewer).
request.xml	A sample SOAP message sent by TravelAgencyClient to the STS. It is displayed in the output when you run the example.
reponse.xml	A sample of a SOAP message replied by the STS to the TravelAgencyClient

Prepare the Example

Prerequisites

Before working with this example:

1. Install WebLogic Server, including the examples.
2. Start the Examples server.
3. Open a command window and set up your environment.
4. Change to the `SAMPLES_HOME\server\examples\src\examples\webservices\saml\bearer11ssl` directory, where *SAMPLES_HOME* refers to the main WebLogic Server examples directory, such as `d:\Oracle\Middleware\wlserver_12.1\samples`.
5. Execute the following command from the shell where you set your environment:

   ```
   prompt> ant configwssecurity
   ```

   This command executes the configwssecurity target in wss-config.xml, which in turn executes four WLST scripts: configWss.py, configWLSSecurity.py, addApiProvidersSVSenderV2.py and addApiProvidersSVRecieverV2.py. These scripts configure WebLogic security, as described in Files.
6. Restart WebLogic Server.

   You **must** restart WebLogic Server after executing the WLST script to make the configurations take effect.
7. Execute the following command from the shell where you set your environment:

   ```
   prompt> ant runtime-config-server
   ```

   This command executes the runtime-config-server target in wss-config.xml. It creates the PKI Credential Mapper and sets up the SAML receiver runtime.
8. Open a new command window and set up your environment.
9. Change to the `SAMPLES_HOME\server\examples\src\examples\webservices\saml\bearer11ssl` directory, where *SAMPLES_HOME* refers to the main WebLogic Server examples directory, such as `d:\Oracle\Middleware\wlserver_12.1\samples`.
10. Execute the following command from the shell where you set your environment:

    ```
    prompt> ant create-sts
    ```

    This command creates another domain which contains a server hosting the STSHttpsUNT.java Web service. This domain resides at *SAMPLES_HOME*\server\examples\src\examples\webservices\saml\bearer11ssl\build\sts_stage\

Build and Deploy the Example

1. Use the command window where you executed step 7 in the prerequisites section and execute the following command:

   ```
   prompt> ant build
   ```

   This command compiles and stages the example.
2. In the same command window, execute the following command:

```
prompt> ant deploy
```

This command deploys the example on the `wl_server` domain of your WebLogic Server installation.

Run the Example

To run the example, follow these steps:

1. Complete the steps in the "Prepare the Example" section.
2. In your development shell, run the `TravelAgencyClient.java` Java application by running the following command from the main example directory (`SAMPLES_HOME\server\examples\src\examples\webservices\saml\bearer11ssl`):

   ```
   prompt> ant run
   ```

Check the Output

If your example runs successfully, you will get output similar to the following message in the command shell from which you ran the client application:

```
[java] <WSEE:1>timestamp(maxAgesSecs=60) verified<SecurityMessageInspector.doMessageAge:755>
[java] <WSEE:1>Inspecting message authentication identity ...<SecurityMessageInspector.checkMessage:175>
[java] <WSEE:1>Identity is not required.<SecurityMessageInspector.inspectIdentity:803>
[java] <WSEE:1>Inspecting signature ...<SecurityMessageInspector.checkMessage:268>
[java] <WSEE:1>Signature is not required.<SecurityMessageInspector.inspectIntegrity:855>
[java] <WSEE:1>Inspecting encryption  ...<SecurityMessageInspector.checkMessage:281>
[java] <WSEE:1>Encryption is not required.<SecurityMessageInspector.inspectConfidentiality:1097>
[java] <WSEE:1>SOAP Security Message has been verified<SecurityMessageInspector.inspectWssMessage:83>
[java] Flight Service return value: -->Request from [Travel Agency client to Flight Service], response message from Oracle FlightService HTTPS)


BUILD SUCCESSFUL
        Total time: 30 seconds
```

Stop and remove the STS server

1. Execute the following command from the shell where you set your environment:

   ```
   prompt> ant stop-sts
   ```

   This command stops the STS server.

   ```
   prompt> ant remove-sts
   ```

   This command uninstalls the STS server.

   At the end, close the command window where the STS server runs.

Troubleshooting

For generalized troubleshooting tips for the examples, see the following link:

- General Troubleshooting Tips for Examples

Troubleshooting SAML Scenarios

Note: These steps apply to any SAML scenarios.

1. If you have a proxy server between the client and STS or the client and the Web service, you need to set up the following properties in your client:

   ```
   // setup for proxy server
   System.setProperty("proxySet", "true");
   System.setProperty("proxyHost", proxyHost);
   System.setProperty("proxyPort", proxyPort);
   ```

2. If you receive an exception similar to the following while setting up SSL certificates, the workaround is to have a copy of the keystore with certificates under $JAVA_HOME/jre/lib/security.

   ```
   javax.xml.ws.WebServiceException: javax.net.ssl.SSLKeyException: [Security:090542]Certificate chain
   received from www-proxy.us.oracle.com - 148.87.19.20 --> adc2170719.us.oracle.com was not trusted causing
   SSL handshake failure. Check the certificate chain to determine if it should be trusted or not. If it
   should be trusted, then update the client trusted CA configuration to trust the CA certificate that
   signed the peer certificate chain. If you are connecting to a WLS server that is using demo certificates
   (the default WLS server behavior), and you want this client to trust demo certificates, then specify
   -Dweblogic.security.TrustKeyStore=DemoTrust on the command line for this client.
   [java]
   ```

3. Use the following property to get more debug information on the client side: `System.setProperty("javax.net.debug","all");`
4. Ensure that you have the appropriate certificates imported into the keystores. You can use the keytool utility to import and check the list of certificates in a keystore.

   Import a certificate into a keystore:

   ```
   keytool -import -trustcacerts -keystore cacerts -alias WssIP -file WssIP.cer
   ```

   List all certificates in a keystore:

   ```
   keytool -list -v -keystore cacerts
   ```

Related Topics

(Internet connection required.)

- Getting Started With WebLogic Web Services Using JAX-WS
- Getting Started With WebLogic Web Services Using JAX-RPC

- Programming Advanced Features of WebLogic Web Services Using JAX-WS
- Programming Advanced Features of WebLogic Web Services Using JAX-RPC
- Securing WebLogic Web Services
- WebLogic Web Services Reference