SAML 2.0 Web SSO

SAML stands for Security Assertion Markup Language which is a XML based data format for exchanging authentication and authorization data between an identity provider and a service provider. The single most important requirement that SAML addresses is web browser single sign-on (SSO). Three main roles are defined in SAML Specification.

1. The Principal: This is typically the user who tries to access a protected resource or service of a service provider.
2. The Identity Provider: An Identity Provider (IdP) is responsible for authenticating users and issuing assertions which include authentication/authorization decisions and user attributes.
3. The Service Provider: A Service Provider(SP) consumes the assertions issued by Identity Provider and provides services to the principals.

The main use case scenario covered by SAML is the Principal (the user) requesting access to resource or service from the Service Provider. Then the Service Provider, using SAML, communicates with the Identity Provider to obtain identity assertion. The Service Provider makes the access control decision, depending on this assertion.

SAML 2.0 is the latest version of SAML, which uses security tokens containing assertions to pass information about a user between an identity provider and a service provider.

SAML 2.0 provides five main specifications:

- Core
- Bindings
- Profiles
- Metadata
- Conformances

This article provides more information about above specifications.

SAML 2.0 web browser-based SSO profile

SAML 2.0 Web Browser based SSO profile is defined under the SAML 2.0 Profiles specification.

In a web browser based SSO system, the flow can be started by the user either by attempting to access a service at the service provider or by directly accessing the identity provider itself.

If the user accesses a service from a service provider:

1. The service provider determines which identity provider to use (this is the case when there are multiple identity providers. SAML identity provider discovery profile may be used).
2. The service provider generates a SAML message and then redirects the web browser to the identity provider along with the message.
3. Identity provider authenticates the user.
4. The identity provider generates a SAML message and then redirects the web browser back to the service provider.
5. The service provider processes the SAML message and decides to grant or deny access to the user.

If the user accesses the identity provider directly, then only the steps 3, 4 and 5 are in the flow.

The message MUST contain an element which uniquely identifies the service provider who created the message. Optionally the message may contain elements such as Issuer, NameIDPolicy, etc. More information regarding the message can be found in SAML Core Specification.

The following diagram illustrates the scenario:
1. User try to access the SAML2 secured resource

2. User will be redirected with an <AuthnRequest> message to the Identity server

3. Response with the result of the authentication.

SAML 2.0 SSO assertion consumers

Service providers act as SAML assertion consumers. They have two basic functions:

1. Create messages and redirect users to the identity provider with the created message.
2. Process messages from the identity provider and make decisions based on them.

The following code is a sketch of a sample service provider servlet in a SAML 2.0 Web-Browser based SSO system.

```java
public class Resource extends HttpServlet {
    private static SamlConsumer consumer = new SamlConsumer();
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        requestMessage = consumer.buildRequestMessage();
        response.sendRedirect(requestMessage);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) {
        responseMessage = request.getParameter("SAMLResponse").toString();
        result = consumer.processResponseMessage(responseMessage);
    }
}
```
When a web user attempts to access the above servlet, its `doGet()` method is called. Inside the `doGet()` method, it generates an message and then redirects the user to the Identity Provider.

After authentication is completed by the Identity Provider, it does a POST call back to the above servlet with a message. Then the `doPost()` method of the servlet gets called and inside the `doPost()` method, it retrieves the message from the request and then the message is passed to the `SamlConsumer` instance for processing.

**<AuthnRequest> Message**

To create an `<AuthnRequest>` message using the OpenSAML library:

1. Add the OpenSAML library to the build path of the project. You can download the open SAML JAR file from here.
2. A sample `<AuthnRequest>` message can be found here.
3. According to SAML 2.0 specifications, the message must contain an element. Create the `Issuer` element first.

```java
String issuerId = "saml2.sso.demo";
IssuerBuilder issuerBuilder = new IssuerBuilder();
Issuer issuer = issuerBuilder.buildObject("urn:oasis:names:tc:SAML:2.0:assertion", "Issuer", "samlp");
issuer.setValue(issuerId);
```

4. Create the `<AuthnRequest>` next.

```java
// the issuerUrl is the url of the service provider who generates the message
String issuerUrl = "http://localhost:8080/saml2.sso.demo/consumer";
DateTime issueInstant = new DateTime();
AuthnRequestBuilder authnRequestBuilder = new AuthnRequestBuilder();
AuthnRequest authnRequest = authnRequestBuilder.buildObject("urn:oasis:names:tc:SAML:2.0:protocol", "AuthnRequest", "samlp");
authnRequest.setForceAuthn(false);
authnRequest.setIsPassive(false);
authnRequest.setIssueInstant(issueInstant);
authnRequest.setAssertionConsumerServiceURL(issuerUrl);
authnRequest.setIssuer(issuer);
authnRequest.setId(aRandomId);
authnRequest.setVersion(SAMLVersion.VERSION_20);
```

The message may contain many other elements like , etc. those elements can be created and added to the message in the same way.

5. Next encode the message.
Marshaller marshaller =
    Configuration.getMarshallerFactory().getMarshaller(authnRequest);
Element authDOM = marshaller.marshall(authnRequest);

StringWriter rspWrt = new StringWriter();
XMLHelper.writeNode(authDOM, rspWrt);
String requestMessage = rspWrt.toString();

Deflater deflater = new Deflater(Deflater.DEFLATED, true);
ByteArrayOutputStream byteArrayOutputStream = new
    ByteArrayOutputStream();
DeflaterOutputStream deflaterOutputStream = new
    DeflaterOutputStream(byteArrayOutputStream, deflater);
deflaterOutputStream.write(requestMessage.getBytes());
deflaterOutputStream.close();

/* Encoding the compressed message */
String encodedRequestMessage =
    Base64.encodeBytes(byteArrayOutputStream.toByteArray(),
        Base64.DONT_BREAK_LINES);
String encodedAuthnRequest =
    URLEncoder.encode(encodedRequestMessage, "UTF-8").trim();

6. Construct the redirection URL.

    redirectionUrl = identityProviderUrl+ "?SAMLRequest=" + encodedRequestMessage;

7. Redirect the user to the identity provider.

    response.sendRedirect(redirectionUrl);

<Response> Message

The Identity provider must use HTTP POST or artifact binding to transfer the <SAMLResponse> message to the service provider. To read the <Response> message issued by the WSO2 Identity Server, do the following:

1. A sample <Response> message can be found here.
2. The response message must be fetched from the request.

    responseMessage = request.getParameter("SAMLResponse").toString();

3. The fetched "responseMessage" is unmarshaled and the SAML message is retrieved.
DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
documentBuilderFactory.setNamespaceAware(true);
DocumentBuilder docBuilder =
documentBuilderFactory.newDocumentBuilder();

byte[] base64DecodedResponse = Base64.decode(responseMessage);
ByteArrayInputStream is = new
ByteArrayInputStream(base64DecodedResponse);
Document document = docBuilder.parse(is);
Element element = document.getDocumentElement();
UnmarshallerFactory unmarshallerFactory =
Configuration.getUnmarshallerFactory();
Unmarshaller unmarshaller =
unmarshallerFactory.getUnmarshaller(element);
Response response = (Response) unmarshaller.unmarshal(element);

4. The retrieved SAML 2.0 Response message can be easily processed. For example, lets takes the User
Name or the Subject's Name Id.

String subject = response.get Assertions().get(0).getSubject() .getNameID().getValue();

5. Alternatively, if the response is signed by the IdP, you can retrieve the certificate.

String certificate =
response.getSignature().getKeyInfo().getX509Datas().get(0).getX509Certificates().get(0).getValue();

Likewise the message from the WSO2 Identity Server can be read easily.

Identity-agent-sso is an implementation of all the details discussed above, which can be used to implement
SSO enabled web applications. Travelocity is a sample SSO enabled web-app, which is implemented based
on Identity-agent-sso.

Relay state

The RelayState parameter is used so that the service provider can pass some value to the identity provider with the
AuthnRequest and get the same value back with the Response. This value can be any string and can be useful
for service provider application logic (when there is a failure, redirecting to the URL that comes as the RelayState
parameter is one way that this can be used).

- For a inbound request to the Identity Server, if the RelayState parameter is present, the Identity Server
  sends back the same value in the response.
- For federation using SAML2, the Identity Server uses the RelayState parameter to pass the session index,
  which is required to continue the authentication flow after receiving authentication response.

Identity provider initiated SSO

To initiate IdP Initiated SSO you need to perform a HTTP GET/POST to the following URL (assume the registered
service provider issuer ID is foo.com)

https://localhost:9443/samlsso?spEntityID=foo.com
This request will authenticate and redirect the user to the registered Assertion Consumer URL. Optionally, you can send in an 'acs' parameter as follows.

```
```

This request will authenticate and redirect the user to the URL in the 'acs' parameter itself.

Either you could have SP Initiated SSO only, or SP Initiated SSO and IdP Initiated SSO. You can't have IdP initiated SSO only. By design, SP Initiated SSO is more restrictive and secure. If a service provider is allowed to do IdP Initiated SSO, it would automatically imply that this service provider is allowed to do SP initiated SSO as well.

### Related Topics

- See [Configuring SAML2 Single-Sign-On Across Different WSO2 Products](#) to set up single sign on between different WSO2 products using SAML2.

**Note:** To make this work, IdP initiated SSO should be enabled in your IdP.