Configuring Shibboleth Delegation for a Portal

Enhancements and extensions to the Shibboleth software, and creation of a Delegated SAML Authentication Library have enabled a delegated authentication model among SAML-enabled services. The use case motivating this development was to enable portlets in a uPortal-based portal to access back-end services on behalf of portal users via Shibboleth and this delegation model. This document will give an orientation and specific configuration guidance to deployers who wish to set up a similar environment.

- Required Software
  - Shibboleth IdP v2.1.3+ with delegation plug-in
  - Shibboleth SP v2.2+
  - Delegated SAML Authentication Library
  - uPortal v3.1
  - What's going on
- Configuration constants
- Configuration Guidance
  - IdP Configuration
  - Portal's SP Configuration
  - Portlet's Delegated SAML Authentication Library Configuration
  - WSP's SP Configuration
- Trust Management and TLS Considerations

Required Software

We'll start with a high level review of the main components and their interactions that make delegation happen.

Shibboleth IdP v2.1.3+ with delegation plug-in

The delegation plug-in can be obtained at <http://svn.shibboleth.net/view/extensions/java-idp-delegation/>. It includes installation instructions in the form of a README file.

The plugin will **NOT** work with earlier versions.

Shibboleth SP v2.2+

Starting with this version, the SP software contains all of the necessary delegation-related enhancements as well as enhancements to integrate with the Delegated SAML Authentication Library. It is typically deployed in two roles in this delegation scenario: one instance protects the portal (or any user-facing web application making use of the library) and the other protects a back-end Web Service Provider (WSP).

Delegated SAML Authentication Library

This library extends the Apache HTTP Client (v4) in Java with an implementation of the SAML ECP profile. It is used by portlets that need delegated access to a WSP using a SAML-based delegation model. More information at https://wiki.jasig.org/display/UPM31/Delegated+Authentication+Integration+Library

uPortal v3.1

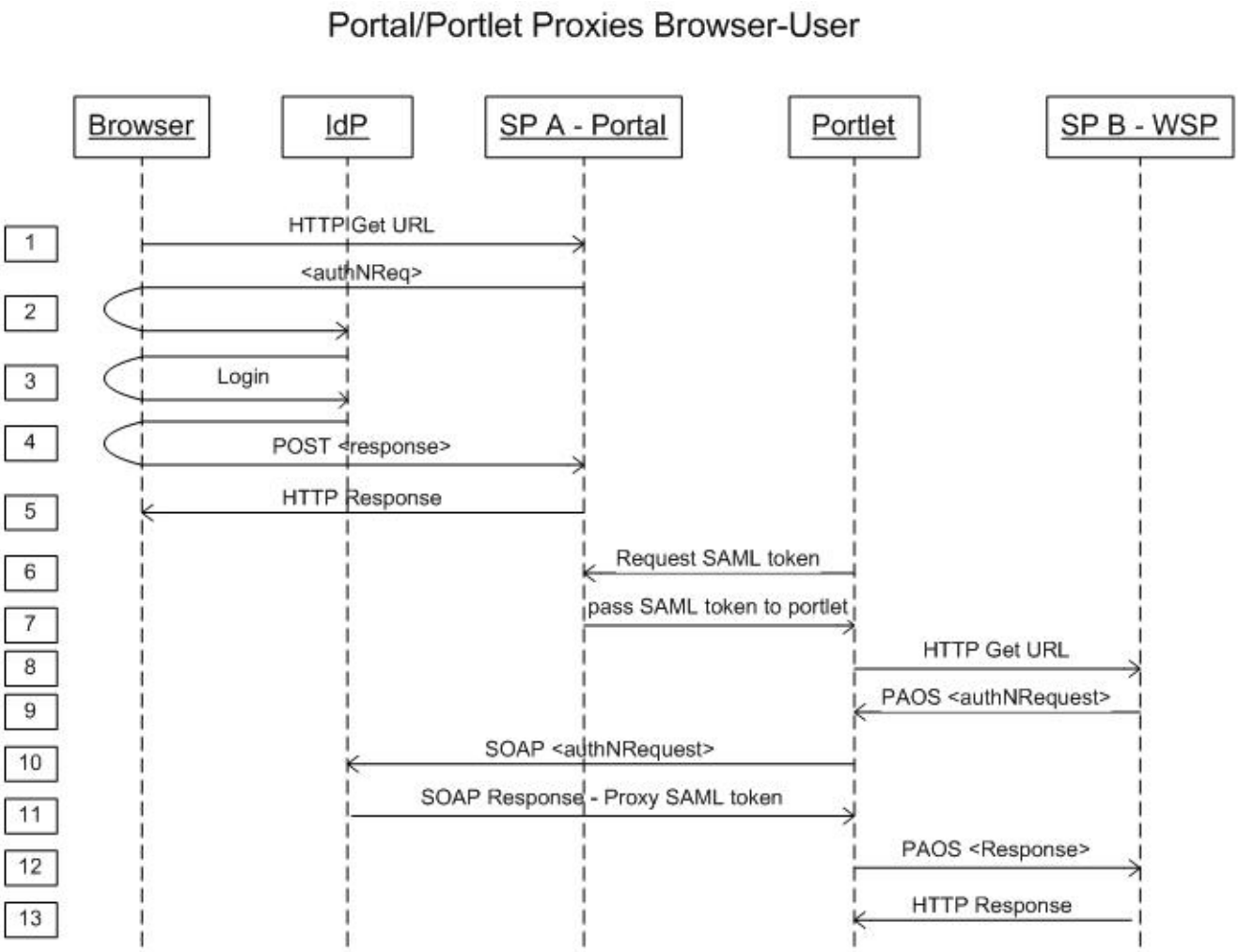
This version of uPortal contains enhancements enabling it to provide Shibboleth-delivered attributes, the "delegatable" SAML assertions delivered when users login to the portal, and selected portions of the relevant IdP's metadata, for use within the portal environment. Portlets using the Delegated SAML Authentication Library must request these from the portal through the portlet API.

Updated material at https://wiki.jasig.org/display/UPM31/00+-+Delegated+SAML+Authentication

What's going on

The following sequence diagram shows how the user's browser, the IdP, the SP protecting the portal, the portal itself, the portlet, and the SP protecting the back-end WSP all interact to enable the portlet to proxy the user to the WSP.

Note that the swimming lane labeled "SP A - Portal" sometimes refers to the Shibboleth SP itself, and sometimes to an action that uPortal takes. They are considered as one actor. The Portlet is given its own swimming lane to highlight the actions it takes. To simplify the development of delegation capability for this portal use case, it was decided that, from the SAML protocol perspective, the portlet, the portal, and SP A would be treated as a single logical security entity.



Note: The portal's SP and the portlet's Delegated SAML Authentication Library act as a single SAML entity for this use case.

1-5. Browser-user authenticates to portal. Per IdP configuration, SAML response token includes list of entityIDs for WSPs for which portal is authorized to delegate.

6-8. Portlet is "activated", gets user's SAML response token through portlet API, and requests a URL from WSP.

9-13. WSP issues authNRequest to IdP via Portlet. Portlet follows ECP profile to obtain SAML token to access Web Service Provider (WSP) as browser-user.

Configuration constants

For a given IdP, Portal SP, and WSP SP, the entityIDs and public keys need to be used and shared (via metadata) consistently across all the components. Most problems will result from a failure to do this. Use this space to record their values for a particular set of entities. The section following this will refer to them by the "shell variables" defined on the left hand side below.

$IdPEntityID	
$PortalEntityID	
$WSPEntityID	

Configuration Guidance

Each component - IdP, the Portal's SP, the Portlet, and the WSP's SP - has ordinary configuration needs unrelated to enabling delegation. A general prerequisite of the following guidance is that these are working in standard, non-delegated scenarios.

IdP Configuration

Prerequisite to these instructions, we assume that:

- All endpoints as shipped with IdP v2.1.3+ are enabled.
- The installation instructions for the delegation plug-in have been followed.
- Attribute filters correctly permit the release of attributes needed by each WSP.

Configuration objectives for the IdP are:

1. Allow the IdP to produce delegatable tokens for each "Portal" SP.
2. Allow the portal to act on SSO requests from WSPs.
3. Ensure that step 5(g) of the delegation plug-in's installation instructions was done correctly.

These are by no means the only steps, as outlined in the instructions linked above, but are the primary steps involving references to the other scenario components.

The `<RelyingParty>` element for the Portal must have a non-standard `<ProfileConfiguration>` element added to enable delegation support. To allow the portal to authenticate itself as the user when relaying requests from WSPs, the `allowTokenDelegation` attribute must be set to `"true"`, and a `<samldel:DelegationRestriction>` element for each WSP must be included. Also, for the portal use case, in which only one delegation "hop" is needed, the IdP is configured to create a proxy token that cannot itself be proxied. That's controlled with the `maximumTokenDelegationChainLength` attribute.

```
<RelyingParty id="$PortalEntityId" ... >
  ...
  <ProfileConfiguration xsi:type="samldel:LibertyIDWSFSSOSProfile"
    securityPolicyRef="shibboleth.ext.delegation.LibertySSOSPolicy"
    maximumTokenDelegationChainLength="1"
    allowTokenDelegation="true"
    signAssertions="always"
    encryptNameIds="never" >
    <samldel:DelegationRestriction>$WSPEntityId</samldel:DelegationRestriction>
  </ProfileConfiguration>
  ...
</RelyingParty>
```

A `<RelyingParty>` element for each WSP must have a non-standard `<ProfileConfiguration>` element to enable delegation **to** it. To prevent it from exercising further delegated access to other services, set `allowTokenDelegation` to `"false"`.

```
<RelyingParty id="$WSPEntityId" ... >
  ...
  <ProfileConfiguration xsi:type="samldel:LibertyIDWSFSSOSProfile"
    securityPolicyRef="shibboleth.ext.delegation.LibertySSOSPolicy"
    allowTokenDelegation="false"
    signAssertions="always"
    encryptNameIds="never" >
  </ProfileConfiguration>
  ...
</RelyingParty>
```

Step 5(g) of the delegation plug-in installation instructions requires a that a suitable IdP key be identified as **IdPValidationOnlyCredential**. It's important that this not be a new key, but the **same** key the IdP uses to sign the assertions it issues to the Portal SP. Typically, you use a single key for all of the IdP's needs.

```
<security:Credential id="IdPValidationOnlyCredential" xsi:type="security:X509Filesystem">
  <security:PrivateKey>usual IdP private key</security:PrivateKey>
  <security:Certificate>usual IdP certificate</security:Certificate>
</security:Credential>
```

Portal's SP Configuration

The sole configuration objective for this SP is to enable it to integrate with the Delegated SAML Authentication Library used by portlets acting as delegates. There is no other delegation-specific configuration required.

Integrating uPortal with Shibboleth to support delegation centers on enabling the SP to provide the portal's Delegated SAML Authentication Library with the SAML assertion from the user's initial login, and with the IdP's public key(s) from its metadata.

SAML assertions are exposed by the SP via the mechanism described in the https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPAssertionExport topic, by enabling the `exportAssertions` content setting. The `exportACL` property of the `<Sessions>` element limits access to the exported assertions by IP address. This may need to be modified to suit the local environment.

The IdP's public keys are extracted from the metadata provided to the SP by configuring an `<AttributeExtractor>` of `type="KeyDescriptor"`. The extracted keys are mapped to SP-cached attributes based on the `signingId` option on that plugin, and the `metadataAttributePrefix` property.

The example configuration below assumes that the default "application" configured for the SP is the Portal. It is also possible to associate these declarations with specific SP-defined applications via the usual override mechanism.

```
<RequestMapper type="Native">
  <RequestMap applicationId="default">
    <Host name="$PortalHostName" authType="shibboleth" requireSession="true" exportAssertion="true"/>
  </RequestMap>
</RequestMapper>
...
<ApplicationDefaults id="default" policyId="default" entityID="$PortalEntityID"
  ...
  metadataAttributePrefix="Meta-">
  ...
  <Sessions ...
    exportLocation="http://localhost/Shibboleth.sso/GetAssertion"
    exportACL="127.0.0.1" ... >
    ...
  </Sessions>
  ...
  <AttributeExtractor type="Chaining">
    <AttributeExtractor type="XML" path="attribute-map.xml"/>
    <AttributeExtractor type="KeyDescriptor" signingId="Signing-Keys"/>
  </AttributeExtractor>
  ...
</ApplicationDefaults>
```

Note that the AttributeExtractor configuration above, together with the `metadataAttributePrefix` setting, results in the IdP's public key(s) used for authentication being placed in an SP attribute named "Meta-Signing-Keys".

Portlet's Delegated SAML Authentication Library Configuration

First read the following four pages from the uPortal v3.1 wiki:

Delegated SAML Authentication

Making the SAML assertion available to uPortal portlets

Delegated SAML Authentication Library for Portlet Developers

Using UW Web Proxy Portlet with Delegated SAML Authentication Library

Of particular note, the Portlet needs access to the private key used by the Portal's SP so it can successfully authenticate to the IdP. Here's how, excerpted from above:

```
<bean id="HttpManagerBean" class="edu.wisc.my.webproxy.beans.http.ShibbolethEnabledHttpManagerImpl" scope="
prototype">
<property name="spPrivateKey" value="some path"/> <!-- This property and the one below are optional to provide
client-side TLS authentication to the IdP -->
<property name="spCertificate" value="some path"/>
<property name="portalEntityID" value="$PortalEntityID"/> <!-- This property is required -->
</bean>
```

WSP's SP Configuration

The sole configuration objective for this SP is to enable it to accept delegated SAML tokens, but in this case only those presented by the Portal.

The <SessionInitiator> with type="SAML2" must have the ECP property set to "true" to enable it to interact properly with the Delegated SAML Authentication Library (because it's not a browser).

The pre-configured <PolicyRule> with type="Conditions" must be given a new child <PolicyRule> with type="Delegation" to enable the SP to accept delegated assertions for SSO. The new rule uses a <del:Delegate> element and a match attribute so that only delegated assertions received from the Portal will be accepted. (Non-delegated assertions from other sources would still be accepted.)

The example configuration below assumes that the default "application" configured for the SP is the WSP. It is also possible to associate these declarations with specific SP-defined applications via the usual override mechanism.

```
<ApplicationDefaults id="default" policyId="default"
  entityID="$WSPEntityID">
  ...
  <SessionInitiator type="Chaining" Location="/Login" isDefault="true" id="Intranet" relayState="cookie"
entityID="$IdPEntityID">
    <SessionInitiator type="SAML2" acsIndex="1" template="bindingTemplate.html" ECP="true"/>
    <SessionInitiator type="Shib1" acsIndex="5"/>
  </SessionInitiator>
  ...
</ApplicationDefaults>
...
<SecurityPolicies>
  <Policy id="default" validate="false">
    <PolicyRule type="MessageFlow" checkReplay="true" expires="60"/>
    <PolicyRule type="Conditions">
      <PolicyRule type="Audience"/>
      <!-- Enable Delegation rule to permit delegated access. -->
      <PolicyRule type="Delegation" match="oldest"
        xmlns:del="urn:oasis:names:tc:SAML:2.0:conditions:delegation">
        <del:Delegate>
          <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">$PortalEntityID</saml:NameID>
        </del:Delegate>
      </PolicyRule>
    </PolicyRule>
    <PolicyRule type="ClientCertAuth" errorFatal="true"/>
    <PolicyRule type="XMLSigning" errorFatal="true"/>
    <PolicyRule type="SimpleSigning" errorFatal="true"/>
  </Policy>
</SecurityPolicies>
```

The use of match="oldest" above isn't really material. The other values of "anyOrder" and "newest" both amount to the same thing when delegation is only one tier deep. We simply need to declare one of them.

Trust Management and TLS Considerations

Most of the trust management issues that arise when delegation is used are the same as in more traditional cases. In each SSO interaction, the SP and IdP are assumed to have metadata about each other containing the authorized public keys to accept. During the delegation step, the Portal SP acts as the client, and the relevant SP that the IdP cares about is the **WSP**, and not the Portal.

By default with Shibboleth 2.x and the use of SAML 2.0, communication between the IdP and SP is now typically confined to the "front channel", meaning the browser. Direct, back-channel interaction is not usually involved. A SAML delegation scenario, though, turns an SP into a client, and requires SOAP-based messages between that SP (in this case the Portal system) and the IdP.

Because SOAP message signing is quite complex, TLS/SSL is more typically used to secure these exchanges, and in fact the current implementation assumes TLS/SSL is used in order to maintain confidentiality and ensure that assertions containing user data are sent to an authenticated IdP and not an attacker. As a result, the Delegated SAML Authentication Library also required knowledge of the IdP's public key(s) so that it can authenticate the IdP server before sending anything to it. The library is modular with respect to the source of this information, but the Portal SP software is commonly used to obtain it by utilizing the metadata it already has on hand for the IdP.

On the IdP side, TLS/SSL support is handled by the servlet container. In the case of Tomcat, it requires special configuration ***and*** adding a Shibboleth-specific library and configuration to the JRE itself. See the "Supporting SOAP Endpoints" section of this topic for details.