ExternalAuthnConfiguration

Current File(s): conf/authn/external-authn-config.xml

Format: Native Spring

Legacy V2 File(s): conf/handler.xml

- Overview
- General Configuration
- API
  - Inputs
  - Outputs
- Reference
- Beans
- V2 Compatibility
- Notes
  - Duplicating the Flow
  - Dynamic Methods

Overview

The authn/External login flow supports the use of an arbitrary external (to the IdP) authentication mechanism. Its basic function is to provide a non-WebFlow-based integration strategy for new authentication mechanisms. As a rule, using WebFlow is better, easier, and safer, but it does require writing Java code in most cases. This is usually true for external mechanisms also, but they can in some cases be implemented with JSP alone.

By default, the External flow is defined to lack support for advanced controls such as passive or forced authentication, but this can be changed in authn.xml or by copying the External flow to a new user-specified variant and changing its properties.

This login handler usually requires additional code be written in order to trigger the external authentication system. If you're simply looking to authenticate based on the presence of REMOTE_USER, refer to the RemoteUser flow.

Once control is transferred to the external path, the IdP has no control over what happens, and it will implicitly trust any information passed back through the defined interface. If that information can be manipulated, security holes may result. The deployer takes full responsibility for the security of the overall exchange.

General Configuration

Use conf/authn/external-authn-config.xml to configure this flow.

The shibboleth.authn.External.externalAuthnPath bean defines the flow redirection path to the resource that's used to perform the external login, by default a context-relative location. It can be modified if needed, but must be a resource with access to the container session. Modify as needed to match the location of your external interface (see the documentation on flow redirects).

As of V3.3, you may dynamically derive the path to use, typically so that it can vary based on aspects of the request, by defining a bean named shibboleth.authn.External.externalAuthnPathStrategy of type Function<ProfileRequestContext,String>.

The shibboleth.authn.External.ClassifiedMessageMap is a map of exception/error messages to classified error conditions. You can make use of this map either by modifying it to understand the error or exception messages returned by your external code, or by using the map as is and passing back the literal tokens in the map from your external code via the exception or error message interface.

API

The ExternalAuthentication class makes up the interface between the external code and the IdP. The general flow is:

1. Call ExternalAuthentication.startExternalAuthentication(HttpServletRequest), saving off the result as a key.
2. Do work as necessary (reading request details from the attributes below). Any redirects must preserve the key value returned in step 1 because it must be used to complete the login later.
3. Set request attributes to communicate the result of the login back.
4. Call ExternalAuthentication.finishExternalAuthentication(String, HttpServletRequest, HttpServletResponse). The first parameter is the key returned in step 1.

A trivial JSP implementation of this interface is below. This works as is, but is obviously quite limited in its behavior.
External interface example in JSP

```jsp
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<%@ page import="net.shibboleth.idp.authn.ExternalAuthentication" %>
<%@ page import="net.shibboleth.idp.authn.ExternalAuthenticationException" %>

```try {
    final String key = ExternalAuthentication.startExternalAuthentication(request);
    final String username = request.getRemoteUser();
    if (username != null) {
        request.setAttribute(ExternalAuthentication.PRINCIPAL_NAME_KEY, username);
    }
    ExternalAuthentication.finishExternalAuthentication(key, request, response);
} catch (final ExternalAuthenticationException e) {
    throw new ServletException("Error processing external authentication request", e);
}
```

**Inputs**

On first access to the external resource, the request attributes below will be set.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>opensamlProfileRequestContext</td>
<td>ProfileRequestContext</td>
<td>Access to full request context tree</td>
</tr>
<tr>
<td>forceAuth</td>
<td>Boolean</td>
<td>Whether the requester asked for re-authentication</td>
</tr>
<tr>
<td>isPassive</td>
<td>Boolean</td>
<td>Whether the requested asked for passive authentication</td>
</tr>
<tr>
<td>relyingParty</td>
<td>String</td>
<td>Name of the relying party requesting authentication</td>
</tr>
<tr>
<td>extended</td>
<td>Boolean</td>
<td>Whether this login flow has been invoked as an extension of another login flow</td>
</tr>
<tr>
<td>authnMethod (deprecated)</td>
<td>String</td>
<td>Identifier for an authentication method supported by the flow</td>
</tr>
</tbody>
</table>

Use of "authnMethod" is deprecated because login flows in V3 are more general than in V2 and are designed to handle more than one possible method at a time, so the full semantics of a request can only be understood by examining the request context tree. The value in this attribute will generally be an arbitrary one chosen from among the possible set that the flow is configured to support. To determine whether the RP requested a specific form of authentication, you must walk the context tree from the root down to the AuthenticationContext and then on to a RequestedPrincipalContext.

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalName</td>
<td>String</td>
<td>Name of authenticated subject to use as the login result</td>
</tr>
<tr>
<td>principal</td>
<td>java.security. Principal</td>
<td>Java Principal object to use as the login result</td>
</tr>
<tr>
<td>subject</td>
<td>java.security.Subject</td>
<td>Java Subject object to use as the login result</td>
</tr>
<tr>
<td>authnError</td>
<td>String</td>
<td>Error message to return in place of a successful login</td>
</tr>
<tr>
<td>authnException</td>
<td>Exception</td>
<td>Explicit exception object to return in place of a successful login</td>
</tr>
<tr>
<td>authnInstant</td>
<td>org.joda.time. DateTime</td>
<td>Exact time of authentication to report back</td>
</tr>
<tr>
<td>authnAuthorities</td>
<td>Collection&lt;String&gt;</td>
<td>Ordered collection of URLs identifying upstream/proxied authenticating &quot;authorities&quot; used to authenticate the subject</td>
</tr>
<tr>
<td>doNotCache</td>
<td>Boolean</td>
<td>If true, prevents the result from being saved for future use for SSO</td>
</tr>
<tr>
<td>previousResult</td>
<td>Boolean</td>
<td>If true, the &quot;new&quot; AuthenticationResult is created with the &quot;previousResult&quot; flag set to true (mainly impacts auditing)</td>
</tr>
</tbody>
</table>
Only one "result" or error attribute should be set by the external code. Setting more than one has unspecified behavior. In most cases, a simple principalName should be returned on success, but you can return the more complex objects to pass back additional information such as public or private credentials or custom principal data.

Reference

Beans

The beans defined in authn/external-authn-config.xml follow:

<table>
<thead>
<tr>
<th>Bean ID</th>
<th>Type</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>shibboleth.authn.External.externalAuthnPath</td>
<td>String</td>
<td>contextRelative:Authn /External</td>
<td>Spring Web Flow redirection expression for the protected resource</td>
</tr>
<tr>
<td>shibboleth.authn.External.externalAuthnPathStrategy</td>
<td>Function&lt;ProfileRequestContext,String&gt;</td>
<td>A constant function returning the bean value above.</td>
<td>A function that returns the redirection expression to use for the protected resource</td>
</tr>
<tr>
<td>shibboleth.authn.External.ClassifiedMessageMap</td>
<td>Map&lt;String,List&lt;String&gt;&gt;</td>
<td>various</td>
<td>A map between defined error/warning conditions and events and implementation-specific message fragments to map to them.</td>
</tr>
<tr>
<td>shibboleth.authn.External.resultCachingPredicate</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td></td>
<td>An optional bean that can be defined to control whether to preserve the authentication result in an IdP session</td>
</tr>
<tr>
<td>shibboleth.authn.External.addDefaultPrincipals</td>
<td>Boolean</td>
<td>true</td>
<td>Whether to add the content of the supportedPrincipals property of the underlying flow descriptor to the resulting Subject</td>
</tr>
<tr>
<td>shibboleth.authn.External.matchExpression</td>
<td>Pattern</td>
<td></td>
<td>Regular expression to match username against</td>
</tr>
</tbody>
</table>

V2 Compatibility

This flow is designed to be similar to, but is not compatible with, the 2.x External login handler. In most cases, converting will be a matter of changing a bit of code to use new packages and APIs. The names of the servlet request attributes on input and output are largely compatible with V2.

Notes

Duplicating the Flow

If more than one instance of the external interface is needed to support different requirements, you can create any number of external flows by copying the descriptor in general-authn.xml, assigning a custom name to each copy, and creating a duplicate flow. For example, to create a custom version called "authn/ext1", copy files like so:

```bash
$ cp system/flows/authn/external-authn-flow.xml flows/authn/ext1/ext1-flow.xml
$ cp system/flows/authn/external-authn-beans.xml flows/authn/ext1/ext1-beans.xml
```

Obvious edits to both files are then needed to correct the paths and bean names used to configure the flow. You can create any number of such flows as desired.

Dynamic Methods

Note that returning a Subject is often paired with setting the shibboleth.authn.External.addDefaultPrincipals bean to false, to dynamically establish Principal(s) representing the authentication method used without having them overwritten. For example, your External flow's supportedPrincipals property might be defined to include both password and multi-factor authentication Principals (meaning it supports both methods), but you can return the specific method used at runtime in the Subject.