CasProtocolConfiguration

File(s): conf/idp.properties, conf/relying-party.xml, conf/cas-protocol.xml

Format: Native Spring

Shibboleth IdP 3.0 supports most of the CAS protocol v2 specification including attribute release and CAS proxy support.

Notable features:

1. The XML response delivered by the /serviceValidate URI includes the <cas:attributes> extension supported by most CAS clients.
2. CAS proxy support with SSL/TLS trust configuration provided by Shibboleth IdP trust engine.
3. Simple metadata facility to configure verified relying parties modeled on Jasig/Apereo CAS service registry.

Notable requirements:

The CAS protocol requires a server-side StorageService implementation for both the CAS protocol ticket store and IdP session store. The following storage service implementations are suitable for use with CAS:

- MemoryStorageService (shibboleth.StorageService bean)
- JPAStorageService (HA compatible)
- MemcachedStorageService (HA compatible)

Notable limitations:

There is no support at present for the CAS /logout URI since global logout/single logout (SLO) in Shibboleth has been discouraged historically. IdP 3.0 has improved support for IdP-initiated SLO, which is conceptually similar to the behavior of the CAS /logout URI, and will be implemented on top of IdP capabilities in 3.2.0 (tentative).

CAS Quickstart

1. Configure suitable storage service.
2. Add CAS protocol profiles to the default relying party by editing conf/relying-party.xml.
3. Add verified CAS relying parties by editing conf/cas-protocol.xml.
4. Configure SSL/TLS trust (optional; only required for CAS proxy support).

Configure Storage Service

CAS requires a server-side storage service for CAS protocol tickets and IdP sessions since the ticket validation step of the protocol requires back-channel communication. The following properties define the storage service used for IdP sessions and CAS tickets, respectively:

- idp.session.StorageService
- idp.cas.StorageService

Modify conf/idp.properties accordingly to define suitable storage service beans for each subsystem. For simple deployments, the default in-memory storage service bean, shibboleth.StorageService, should suffice. HA deployments will require defining a custom JPAStorageService or MemcachedStorageService bean in conf/global.xml and referencing that.

Sample idp.properties excerpt for CAS

```properties
# Simple non-HA deployment case
idp.session.StorageService=shibboleth.StorageService
idp.cas.StorageService=shibboleth.StorageService
```

Enable CAS Protocol

The CAS protocol support is represented with a set of profile configuration beans, one for each of the primary CAS protocol URIs:

<table>
<thead>
<tr>
<th>Profile Bean Name</th>
<th>URI Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS.LoginConfiguration</td>
<td>/login</td>
</tr>
<tr>
<td>CAS.ProxyConfiguration</td>
<td>/proxy</td>
</tr>
<tr>
<td>CAS.ValidateConfiguration</td>
<td>/serviceValidate, /proxyValidate, /samlValidate</td>
</tr>
</tbody>
</table>
Each deals with configuration concerns around the related URI. The profiles are enabled as a function of relying party or relying party group as described in Profiles and Per-RelyingParty Behavior. While each profile may be enabled individually, the only configuration that provides meaningful behavior is one that includes at least \texttt{CAS>LoginConfiguration} and \texttt{CAS:ValidateConfiguration}. The \texttt{CAS:ProxyConfiguration} profile bean is truly optional since omitting it disables proxy support for the relying party, which is a meaningful configuration capability. The following configuration example demonstrates how to enable all CAS protocols for the default relying party.

### CAS enabled for default relying party

```xml
<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">
  <property name="profileConfigurations">
    <list>
      <ref bean="Shibboleth.SSO" />  
      <ref bean="SAML1.AttributeQuery" />  
      <ref bean="SAML1.ArtifactResolution" />  
      <ref bean="SAML2.SSO" />  
      <ref bean="SAML2.ECP" />  
      <ref bean="SAML2.Logout" />  
      <ref bean="SAML2.AttributeQuery" />  
      <ref bean="SAML2.ArtifactResolution" />  
      <ref bean="CAS>LoginConfiguration" />  
      <ref bean="CAS:ProxyConfiguration" />  
      <ref bean="CAS:ValidateConfiguration" />
    </list>
  </property>
</bean>
```

### Define Relying Party Metadata

Relying parties that authenticate to the IdP via the CAS protocol are called “services.” Two mechanisms are supported for registering services:

1. Service registry
2. SAML metadata

For IdP versions that support SAML metadata, by default the IdP checks all SAML metadata sources first then proceeds to the service registry if and only if no match is found.

### Configure Proxy Trust (Optional)

When the service provides a \texttt{pgtIurl}/protocol parameter at ticket validation time and the service is authorized to proxy, the IdP attempts an HTTPS connection to that URL. (The IdP will immediately reject a non-HTTPS proxy callback URL.) The proxy callback endpoint must present a certificate that is trusted by the IdP, which requires explicit configuration; in other words, the IdP does not use the default Java truststore for trust material since that would not provide adequate security. The IdP configuration machinery for proxy trust is designed to force deployers to consider what hosts they trust. While this approach requires more effort, it is justified on a few counts:

1. Proxying is a substantial increase in authorization afforded to a service.
2. The callback endpoint receives a credential, so authentication and trust are paramount.
3. TLS authentication is the sole authentication mechanism used to authenticate proxy callback endpoints.

Since certificate trust underpins the last point, it requires adequate treatment to garner meaningful security. The IdP offers two approaches to proxy trust configuration in order of decreasing security:

1. Registration of service-specific end-entity certificates via SAML metadata
2. PKIX validation of end-entity certificates based on a set of CA trust anchors.
   a. Configure via \texttt{relying-party.xml} \texttt{3.0,3.1,3.2,3.3}
   b. Configure via \texttt{cas-protocol.xml}\texttt{3.4.2}

The second approach only provides meaningful security when you have a small number of certificate authorities that issue Web server certificates with a high degree of identity vetting. If that requirement is not met, configuring end-entity certificates via metadata is the recommended approach.

### Alternate \texttt{cas:user} in the validation response (Optional)

CAS validation responses include the user’s username. This is generally the normalized username the user logged in with. It is possible to substitute the username in the response with a value from another IdP attribute using the \texttt{p:userAttribute} attribute of \texttt{CAS:ValidateConfiguration} profile in a Relying Party Override.

In this example, 3 services will get the value of the studentId attribute instead of the standard username. Note that these services are identified by Regex strings instead of standard literal strings.
External client configuration

As CAS is configured as a new profile for the default relying party, CAS clients should be configured with these values:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cas_host</td>
<td>idp.example.org</td>
</tr>
<tr>
<td>$cas_context</td>
<td>/idp/profile/cas</td>
</tr>
</tbody>
</table>

This example is based on PHP CAS Client from Jasig.

Ticket Service

As of version 3.3.0, a stateless ticket service component, EncodingTicketService, is the default ticket producer and consumer. This component stores ticket metadata in the random part of the ticket identifier using a DataSealer component. This component has a couple advantages:

- No dependency on a StorageService component
- Back-channel limitations are removed such that there is no requirement for shared storage between IdP nodes

This component should be used where possible. Note that deployers that need CAS proxy support still require stateful ticket storage, so the advantages of EncodingTicketService are diminished in that case and the original component SimpleTicketService is recommended.

It should be noted that some CAS clients cannot accommodate the long identifiers produced by EncodingTicketService; phpCAS and mod_auth_cas are notable clients that can't support the identifiers as of this writing.

The choice of ticket service is controlled by configuration in conf/cas-protocol.xml. Simply comment/uncomment the appropriate configuration for the ticket service you want to use. The default configuration for 3.3.0 and later is shown below:
Ticket Service Configuration in cas-protocol.xml

<!--
The default ticket service as of 3.3.0 serializes ticket data into the opaque section of the ticket ID
for service tickets and proxy tickets. Proxy-granting tickets still require server-side storage, and
a StorageService defined by the idp.cas.StorageService is used. Thus for deployers that do not require
CAS proxy capabilities, no stateful storage mechanism is required; that means no memcached or database
is required for HA deployments that want CAS (without proxy) support. A notable limitation of the new
component is that the one-time use feature of service and proxy tickets is not available due to the lack
of a ticket-tracking mechanism. Instead, tickets expire when their expiration period is exceeded.
If this limitation is of concern, one may consider decreasing ticketValidityPeriod on the profile
configuration from the default 15000ms.
-->
<alias name="encodingTicketService" alias="shibboleth.CASTicketService" />

<!--
Uncomment the following element and comment out the above to enable the previous default ticket service
that uses a StorageService for ticket persistence. Use this if the one-time use limitation of
EncodingTicketService is problematic and can't be mitigated by decreasing ticketValidityPeriod.
-->
<!--<alias name="simpleTicketService" alias="shibboleth.CASTicketService" /> -->