ShibbolethSSOConfiguration

File(s): *conf/relying-party.xml*

Format: Native Spring / Deprecated Custom Schema

Legacy V2 File(s): *conf/relying-party.xml*

Overview

The **Shibboleth.SSO** profile configuration bean enables support for the SAML 1.1 Browser Single Sign-On profile initiated via the legacy Shibboleth request protocol, which is documented in the UnsolicitedSSOConfiguration page.

Configuration

The most typical options used are described in more detail below, but not every obscure option is discussed. See the javadoc for all of the possible configuration options for this profile (note that many of them are inherited from parent classes).

Virtually all the configuration options below can be set via two different properties: a static property that explicitly sets the value to use and a lookup strategy or predicate property that takes a Function or Predicate and returns the value to use. The dynamic property is generally named "propertyNamePredicate" or "propertyNameLookupStrategy" for Boolean- and non-Boolean-valued properties respectively.

The examples shown are not specific to any particular profile configuration.

Common

Options common to most/all profiles:

Name	Type	Default	Description
securityConfiguration	SecurityConfiguration	Bean named **shibboleth.DefaultSecurityConfiguration**	An object containing all of the default security-related objects needed for peer authentication and encryption. See SecurityConfiguration for complete details.
disallowedFeatures [3.3]	Integer	0	A bitmask of features to disallow, the mask values being specific to individual profiles

Guidance

Modifying the security configuration is usually done to:

- specify an alternate signing or decryption key to use
- control signing or encryption algorithms (but for metadata you control, it's advisable to control algorithms by using an extension to specify supported algorithms).

Authentication

Options common to profiles that perform authentication:

Name	Type	Default	Description
postAuthenticationFlows	List<String>		Ordered list of profile interceptor flows to run after successful authentication
defaultAuthenticationMethods	List<Principal>		Ordered list of Java Principals to be used to select appropriate login flow(s) to attempt, in the event that a relying party does not signal a preference. See AuthenticationFlowSelection.
nameIDFormatPrecedence	List<String>		Ordered list of NameID Format(s) to select for use, in the event that a relying party does not signal a preference. This is nominally SAML-specific but may be adapted for use with other supported protocols as circumstances warrant.
forceAuthn [3.4]	Boolean	false	Disallows use (or reuse) of authentication results and login flows that don't provide a real-time proof of user presence in the login process

Guidance

The `postAuthenticationFlows` property is used to apply special processing to requests, such as attribute release consent, password expiration warnings, authorization checks, or other custom processing.

Examples of postAuthenticationFlows property

```
<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">

        <!-- Add consent to Shibboleth SSO profile. -->
        <bean parent="Shibboleth.SSO" p:postAuthenticationFlows="attribute-release" />

        <!-- Add consent, followed by expiring password check, to SAML 2 SSO profile. -->
        <bean parent="SAML2.SSO" p:postAuthenticationFlows="#{{'attribute-release', 'expiring-password'}}" />

        <!-- Return interceptors from a function bean (not shown). -->
        <bean parent="Shibboleth.SSO" p:postAuthenticationFlowsLookupStrategy-ref="InterceptorsFunction" />

</bean>
```

With the increased use of multi-factor authentication, it is more common to find RPs that can specify authentication requirements, but there are still many cases, particular with commercial services, in which it becomes necessary to force the use of specific login methods. This can be achieved using the `defaultAuthenticationMethods` property by specifying one or more corresponding Principals to trigger the use of stronger methods.

Note that you must also prevent a malicious actor from overriding this preference for a SAML 2.0 SP by manufacturing a request, via one of two means:

- For a SAML 2.0 SP that can sign its requests, its metadata can be modified with the `AuthnRequestsSigned` flag to indicate that its requests must be signed.
- Alternatively, the `disallowedFeatures` property may be set with the **SAML2.SSO.FEATURE_AUTHNCONTEXT** bean to block use of the SAML 2.0 `<RequestedAuthnContext>` feature.

At present, no other authentication profiles support a feature capable of requesting the authentication method.

Examples of defaultAuthenticationMethods property

```
<!-- NOTE: these example.org constants are examples and are not suitable for real use. -->
<bean id="MFASAML2Principal" parent="shibboleth.SAML2AuthnContextClassRef"
        c:_0="http://example.org/ac/classes/mfa" />
<bean id="MFASAML1Principal" parent="shibboleth.SAML1AuthenticationMethod"
        c:_0="http://example.org/ac/classes/mfa" />

<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">

        <!-- Require MFA with Shibboleth SSO profile. -->
        <bean parent="Shibboleth.SSO">
                <property name="defaultAuthenticationMethods">
                        <list>
                                <ref bean="MFASAML1Principal" />
                        </list>
                </property>
        </bean>

        <!-- Require MFA with SAML 2 SSO profile. -->
        <bean parent="SAML2.SSO" p:disallowedFeatures-ref="SAML2.SSO.FEATURE_AUTHNCONTEXT">
                <property name="defaultAuthenticationMethods">
                        <list>
                                <ref bean="MFASAML2Principal" />
                        </list>
                </property>
        </bean>

</bean>
```

The `nameIDFormatPrecedence` property is a common way of controlling the type of SAML NameIdentifier / NameID included in a response, a common requirement of many commercial services. It is in fact the **only** way to force the use of the ill-advised "`urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`" Format, which it must be noted is very rarely needed, despite frequent mis-documentation to the contrary.

In most cases, it is better to control the Format selected by including a `<NameIDFormat>` element in the SP's metadata. In the event that you don't control the metadata, you can inject the required element by applying a metadata filter.

Examples of nameIDFormatPrecedence property

```
<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">

        <!--
        Both constants below evaluate to the string "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified",
        and are interchangeable, they're just illustrations of different ways to reference the same string.
        -->

        <!-- Use "unspecified" NameIdentifier with Shibboleth SSO profile. -->
        <bean parent="Shibboleth.SSO">
                <property name="nameIDFormatPrecedence">
                        <list>
                                <util:constant static-field="org.opensaml.saml.saml1.core.NameIdentifier.
UNSPECIFIED" />
                        </list>
                </property>
        </bean>

        <!-- Use "unspecified" NameID with SAML 2 SSO profile. -->
        <bean parent="SAML2.SSO">
                <property name="nameIDFormatPrecedence">
                        <list>
                                <util:constant static-field="org.opensaml.saml.saml2.core.NameIDType.
UNSPECIFIED" />
                        </list>
                </property>
        </bean>

        <!-- Return formats from a function bean (not shown). -->
        <bean parent="Shibboleth.SSO" p:nameIDFormatPrecedenceLookupStrategy-ref="FormatsFunction" />

</bean>
```

SAML

Options common to SAML profiles:

Name	Type	Default	Description
additionalAudiencesForAssertion	Collection<String>		Additional values to populate into audience restriction condition of assertions
includeConditionsNotBefore	Boolean	true	Whether to include a `NotBefore` attribute in assertions
assertionLifetime	Duration	PT5M	Lifetime of assertions
signAssertions	Predicate<ProfileRequestContext>	false	Whether to sign assertions
signResponses	Predicate<ProfileRequestContext>	varies by profile	Whether to sign responses
signRequests	Predicate<ProfileRequestContext>	false	Whether to sign requests

Guidance

It isn't too common to need any of these options, and they should be changed only with care.

The `additionalAuduencesForAssertion` and `includeConditionsNotBefore` settings provide ways to work around bugs in other systems. You should never use these settings without obtaining a commitment from the other system's owner to fix their bugs.

The `assertionLifetime` setting does **not** involve control over the session with the relying party, it's only relevant in delegation scenarios.

The signing options have a complex history, which is one reason they are not themselves just boolean-valued. We provide Spring support so you can just set them to "true" or false" as though they are, but they also directly support the more dynamic approach of deriving the value with a bean.

The `signResponses` default varies by profile, see the notes on the individual profile pages.

If you need to enable the `signAssertions` option, and you control the SP's metadata, you should generally add the `WantAssertionsSigned` flag to it in place of using this option.

Artifact

Options common to SAML profiles that may transmit messages via SAML Artifact (a pass by reference instead of value, followed by a callback).

Name	Type	Default	Description
artifactConfiguration	SAMLArtifactConfiguration	Bean named **shibboleth.DefaultArtifactConfiguration**	Customizes the use of SAML artifacts

Guidance

You shouldn't really need to modify this, as artifacts are rarely used anymore, and if they are, the default configuration suffices. The main reason you might change it is to switch a SAML 1.1 SSO configuration from Type 1 to Type 2 artifacts, but that's very obscure. If it ever comes up, we will provide an example.

With SAML 2.0, there is a valid case for customizing the configuration on a per-node basis by exposing dedicated resolution endpoints on each node, and making sure a node issues artifacts that will be resolved by that node. This is already exposed for you via the **idp.artifact.endpointIndex** property.

Profile-Specific

Options specific to the Shibboleth / SAML 1.1 SSO profile:

Name	Type	Default	Description
includeAttributeStatement	Boolean	false	Whether to "push" attributes during SSO

Guidance

The historical default for the Shibboleth profile of SAML 1.1 was to issue only authentication information through the normal channel and rely on a back-chanel to query for attributes, due to the lack of support for XML Encryption in SAML 1.1.

This a very commonly modified setting because of the gradual deprecation of the use of the back channel and support for attribute queries. With the very limited use of SAML 1.1, it's usually simpler to forgo supporting queries and simply push attributes for the few legacy systems left, relying on the TLS protections between the client and servers to protect the user's data from passive observation.

Note that the value of this setting is ignored when SAML artifacts are used, it's always true in those cases because the data is passed over a back-channel anyway.

Notes

The default value of `signResponses` for this profile is "true", and it is unsafe to change this value. If you encounter a relying party that accepts an unsigned response that is transmitted via POST (and not artifact), you have identified an insecure implementation and should report the issue immediately while following your local security incident response process.