Overview

The SAML2.SSO profile configuration bean enables support for the SAML 2.0 Browser Single Sign-On profile (the most common profile used today with Shibboleth). This includes support for “unsolicited” or “IdP-initiated” SSO via the request format documented [here](#).

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disallowedFeatures 3.3</td>
<td>Integer</td>
<td>0</td>
<td>A bitmask of features to disallow, the mask values being specific to individual profiles</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postAuthenticationFlows</td>
<td>List&lt;String&gt;</td>
<td>Ordered list of profile interceptor flows to run after successful authentication</td>
<td></td>
</tr>
<tr>
<td>defaultAuthenticationMethods</td>
<td>List&lt;Principal&gt;</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>nameIDFormatPrecedence</td>
<td>List&lt;String&gt;</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>forceAuthn 3.4</td>
<td>Boolean</td>
<td>false</td>
<td>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</td>
</tr>
</tbody>
</table>
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

```xml
<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

```xml
<!-- 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 />` or `<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

```xml
<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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalAudiencesForAssertion</td>
<td>Collection&lt;String&gt;</td>
<td></td>
<td>Additional values to populate into audience restriction condition of assertions</td>
</tr>
<tr>
<td>includeConditionsNotBefore</td>
<td>Boolean</td>
<td>true</td>
<td>Whether to include a NotBefore attribute in assertions</td>
</tr>
<tr>
<td>assertionLifetime</td>
<td>Duration</td>
<td>PT5M</td>
<td>Lifetime of assertions</td>
</tr>
<tr>
<td>signAssertions</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>false</td>
<td>Whether to sign assertions</td>
</tr>
<tr>
<td>signResponses</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>varies by profile</td>
<td>Whether to sign responses</td>
</tr>
<tr>
<td>signRequests</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>false</td>
<td>Whether to sign requests</td>
</tr>
</tbody>
</table>

**Guidance**

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

The `additionalAudiencesForAssertion` 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.
SAML 2.0

Options common to SAML 2.0 profiles:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encryptionOptional</td>
<td>Boolean</td>
<td>false</td>
<td>Whether to automatically disable encryption if the relying party does not possess a suitable key</td>
</tr>
<tr>
<td>encryptAssertions</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>varies by profile</td>
<td>Whether to encrypt assertions</td>
</tr>
<tr>
<td>encryptNameIDs</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>varies by profile</td>
<td>Whether to encrypt NameIDs</td>
</tr>
<tr>
<td>encryptAttributes</td>
<td>Predicate&lt;ProfileRequestContext&gt;</td>
<td>false</td>
<td>Whether to encrypt attributes</td>
</tr>
</tbody>
</table>

Guidance

The encryption options are generally set correctly for each different profile; see the notes on the individual profile pages. We provide Spring support so you can just set them to "true" or "false" as though they are boolean-valued, but they also directly support the more dynamic approach of deriving the value with a bean.

Note that when the conditions to encrypt various constructs evaluate to true, the IdP will fail the request if it is unable to perform the encryption, for whatever reason. This is overrideable using the encryptionOptional property, which allows the IdP to encrypt if it can but continue otherwise. If you carefully control your metadata sources, which you should do in any case, you should be able to trust that any SP lacking an encryption key is incapable of encryption anyway, making the property safe to enable.

Artifact

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactConfiguration</td>
<td>SAMLArtifactConfiguration</td>
<td>Bean named shibboleth.DefaultArtifactConfiguration</td>
<td>Customizes the use of SAML artifacts</td>
</tr>
</tbody>
</table>

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 SAML 2.0 Browser SSO profile:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximumSPSessionLifetime</td>
<td>Duration</td>
<td>0</td>
<td>If non-zero, attempts to limit length of session with SP via SessionNotOnOrAfter attribute</td>
</tr>
<tr>
<td>skipEndpointValidationWhenSigned</td>
<td>Boolean</td>
<td>false</td>
<td>Whether to skip validation of response location via metadata if the request was signed</td>
</tr>
</tbody>
</table>

Guidance

The skipEndpointValidationWhenSigned option is attractive in many enterprise scenarios if you prefer to maintain some degree of security but avoid registration of metadata containing every individual SP endpoint, which adds a lot of overhead in massively vhosted-environments.

It can also add a degree of insulation from SP changes, but in practice systems that are likely to change endpoint locations but don't support metadata-based change control are likely to misunderstand the need to keep entityIDs stable also.

There are a variety of settings related to delegation that are not shown above but can be found in the relevant API documentation.

Notes

The default value of signResponses for this profile is "true", in keeping with modern best practice. As long as one of the response or assertion are signed, use of the profile is "safe" in terms of authentication integrity, but there are vulnerabilities in XML Encryption that make signing responses advisable when the most common encryption algorithms are used. Some of the backstory around the signing defaults is discussed in this thread.
The default value of `encryptAssertions` for this profile is "true".

If you encounter a relying party that accepts an unsigned response and assertion 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.