NameIDConsumptionConfiguration

Current File(s): *conf/c14n/subject-c14n.xml*

Format: Native Spring / Deprecated Custom Schema

Legacy V2 File(s): *conf/attribute-resolver.xml*

Overview

Mapping SAML identifiers into a user identity is one of the use cases for Subject Canonicalization. This mechanism is applied when a SAML 1 `<NameIdentifier>` or SAML 2 `<NameID>` element is passed into the IdP and needs to be mapped back into a username. The most common example is when an AttributeQuery message is received, and the IdP needs to recover the user's identity to pass into the attribute resolver. There are a few additional scenarios where this might happen, but they're substantially less common.

Each method of mapping SAML-carried identifiers into a username is implemented as a subflow and described using a descriptor bean that tells the IdP how to run that flow. The supplied flows handle all of the standard use cases for turning SAML Name Identifiers back into usernames, and little or no configuration is generally needed.

The *subject-c14n.xml* file includes the list of descriptors that describe the possible flows available, and some additional configuration. You might need to modify this file to enable the mapping of custom name identifier Formats into usernames, such as email addresses or DNs.

General Configuration

The **shibboleth.SAMLSubjectCanonicalizationFlows** bean is a list of the descriptor beans defining the SAML canonicalization flows available to run.

By default, it contains flows supporting legacy use of the attribute resolver, both memory- and crypto-based transient identifier reversal, and non-functional stubs for mapping custom string-based identifiers back into usernames, optionally applying regular expression transforms. A commented out flow also exists for reversing storage-managed SAML 2 persistent identifiers.

In the majority of cases, you shouldn't need to add to this list of flows, but if you were to build some kind of custom flow that perhaps relied on a web service or something of that nature, this is where it would be registered.

If you want to allow reversal of persistent IDs, they must be generated using a storage-based strategy (see NameIDGenerationConfiguration) and you must uncomment the "c14n/SAML2Persistent" bean reference here.

Finally, if you wanted to disable the capability to reverse map transient IDs in various ways, you could comment out or remove the relevant references here. A slight performance improvment might be gained, for example, by removing the flows corresponding to the transient-generation strategy you're not using.

Custom Identifier Mapping

It is generally a rare requirement to support reversing more customized Name Identifier formats, which are typically used for commercial SAML-enabled services that have no occasion to issue queries or other requests that would contain the identifiers.

In the event that this is required, the required components for this are already in place and just need to be turned on and configured by adjusting one or more of the other beans toward the end of the file.

The most important one is the **shibboleth.NameTransformPredicate** which controls the circumstances under which the custom mapping flow will be allowed to run. This is for security reasons, to prevent just any requester (even if trusted in general) from performing queries using a particular identifier. The bean itself must be a Predicate object. The example configuration demonstrates how to create a condition that requires the requester be one of a listed set of named entities, which given the rarity of this use case, may often be sufficient. But any Predicate can be used, even a script.

The **shibboleth.NameTransformFormats** bean is a list of the custom Name Identifier Format(s) that you want to support. The pre-existing list contains a number of Formats defined by SAML. You may need to add to it if you or some other party needs to exchange a custom Format.

Lastly, the custom mapping flow supports some simple transform capability. The **shibboleth.NameTransforms** bean is a list of regular expression and replacement string pairs that can transform the input value into a different username value rather than importing it directly.

Reference

Beans

Bean ID	Type	Function
shibboleth.SAMLSubjectCanonicalizationFlows	List<NameIDCanonicalizationFlowDescriptor>	List of flow descriptors enumerating the canonicalization flows to run on incoming Name Identifiers
shibboleth.NameTransformFormats	List<String>	List of Format values to run the "c14n/SAML2Transform" and "c14n/SAML1Transform" flows against
shibboleth.NameTransformPredicate	Predicate<ProfileRequestContext>	Activation condition for the "c14n/SAML2Transform" and "c14n/SAML1Transform" flows
shibboleth.NameTransforms	List<String,String>	List of regular expression and replacement string pairs to apply to the input to the "c14n/SAML2Transform" and "c14n/SAML1Transform" flows
shibboleth.AbstractSAML1C14NFlowBean shibboleth.AbstractSAML2C14NFlowBean	NameIDCanonicalizationFlowDescriptor	Template beans for defining additional flow descriptors

V2 Compatibility

In V2, the process of mapping SAML name identifiers back into usernames was managed using *attribute-resolver.xml* and `<PrincipalConnector>` plugins.

The IdP should load any existing 2.x *attribute-resolver.xml* file and configure itself in an expected manner, but the use of the legacy connectors is deprecated in this release, and deployers are urged to migrate their configurations to native Spring syntax after or while upgrading.

Notes

TBD