NativeSPAttributeDecoder

The `<AttributeDecoder>` element configures the component used to process the content of a SAML element undergoing extraction.

The decoder plugin is responsible for instantiating an object that subclasses the required internal class interface, while capturing any data necessary to properly expose the extracted value(s) to the SP and protected applications. It is the decoder plugin that actually understands how to process an attribute's values.

Common Attributes

- `xsi:type` (XML schema type / QName)
  - Plugin type name.

- `caseSensitive` (boolean) (default is true)
  - Allows the decoder to attach the proper case sensitivity setting to the attribute created. Attributes must carry this setting to enable proper comparison logic by access control plugins.

- `internal` (boolean) (default is false)
  - Allows the decoder to attach the proper internal-only setting to the attribute created. Attributes can carry this setting to hide themselves from CGI export.

- `hashAlg` (string) (Version 2.3 and Above)
  - Name of a hashing algorithm to apply to the decoded attribute's serialized values before caching them. Internally, this turns any source attribute object into a simple/string-valued attribute whose values are the result of hashing the original values that would have been placed into the mapped header(s). The algorithm names to use here are dependent on the cryptographic library that supplies the hashing. In the case of OpenSSL, they're simple names like "SHA1" or "SHA256".

- `langAware` (boolean) (defaults to false) (Version 2.5 and Above)
  - If true, the decoding process changes from "decode all values" to "find the value whose `xml:lang` attribute best matches the combination of the client's requested language preference(s) and any server-imposed defaults". If no match is found, a single unspecified value is selected. All other values are ignored.

String AttributeDecoder

Indicated by `xsi:type="StringAttributeDecoder"`, treats the SAML attribute's values as a simple string. No additional processing is done.

Scoped AttributeDecoder

Indicated by `xsi:type="ScopedAttributeDecoder"`, treats the SAML attribute's values as a two-part relational construct consisting of a left-hand side (the "value") and a right-hand side (the "scope").

During processing, both halves are tracked independently and exposed either as a flattened string or individually, depending on how the object is being used.

Typically, an attribute's scope gives an indication of the domain in which an attribute's value applies; for example, `staff@example.org` represents a staff member at Example Organization, and the scope is `example.org`. However, it may not be desirable to allow `staff@osu.edu` to be asserted by the Brown University IdP. The specialized processing of this decoder facilitates these kinds of distinctions.

With Shibboleth 2.0, scoped attribute processing accomodates multiple XML syntaxes for passing scoped values, including the legacy form supported by Shibboleth 1.x and parsing strings containing an arbitrary delimiter.

Attributes

- `scopeDelimiter` (character) (default is @) (Version 2.3 and Above)
  - The character used to delimit the value from the scope in a flattened source string.

NameID AttributeDecoder

Indicated by `xsi:type="NameIDAttributeDecoder"`, processes SAML attribute values that take the form of a `<saml:NameIdentifier>` or `<saml2:NameID>` element (or the equivalent schema type).

Since SAML identifiers are complex XML types that include XML attributes as well as content, a formatting string must be supplied to instruct the system how to represent the value internally as a flattened string, when necessary.

For historical reasons, the various parts of the XML structure are traditionally concatenated using `!!`, but any kind of formatting is possible.

```
<AttributeDecoder xsi:type="NameIDAttributeDecoder" formatter="$Name!!$NameQualifier!!$SPNameQualifier"/>
```

Attributes

- `formatter` (string) (default is "$Name!!$NameQualifier!!$SPNameQualifier")
    - A formatting string that turns the XML content into a flat string. The string contains one or more substitution tags consisting of a dollar sign ($) followed by the name of an XML attribute or the string "Name" (representing the XML element content). Other characters are echoed through to the constructed string.

- `defaultQualifiers` (bool) (default is false) (Version 2.2 and Above)
    - If true, the values of NameQualifier and SPNameQualifier will be defaulted, if not set by the source, based on the identity provider and service provider identities. Prior to version 2.2, this defaulting behavior was automatic and could not be disabled.

NameIDFromScoped AttributeDecoder (Version 2.1 and Above)

Indicated by `xsi:type="NameIDFromScopedAttributeDecoder"`, combines the features of the "Scoped" and "NameID" decoders above by processing values in "scoped" syntax, but returning a NameID-formatted value. Used for application compatibility by mapping different input syntaxes into one output format.

Attributes

- `formatter` (string) (default is "$Name!!$NameQualifier!!$SPNameQualifier")
    - A formatting string that turns the XML content into a flat string. The string contains one or more substitution tags consisting of a dollar sign ($) followed by the name of an XML attribute or the string "Name" (representing the XML element content). Other characters are echoed through to the constructed string.

- `format` (URI)
    - Optional value to set as the NameID "Format" attribute when constructing formatted values.

- `defaultQualifiers` (bool) (default is false) (Version 2.2 and Above)
    - If true, the values of NameQualifier and SPNameQualifier will be defaulted, if not set by the source, based on the identity provider and service provider identities. Prior to version 2.2, this defaulting behavior was automatic and could not be disabled.

- `scopeDelimiter` (character) (default is @) (Version 2.3 and Above)
    - The character used to delimit the value from the scope in a flattened source string.

KeyInfo AttributeDecoder (Version 2.2 and Above)

Indicated by `xsi:type="KeyInfoAttributeDecoder"`, processes SAML attribute values that take the form of a `<ds:KeyInfo>` element (or the equivalent schema type).

Its current capability is to rely on a KeyInfoResolver plugin to transform the input data into a public key, after which it is DER-encoded into its SubjectPublicKeyInfo form and then base64-encoded.

Attributes

- `hash` (boolean) (default is false)
    - If set to true, the resulting DER-encoded key values are hashed via SHA-1 before being base64-encoded. Note that this is a different hashing operation than the generic one supported with the `hashAlg` attribute, described above.

- `keyInfoHashAlg` (string) (default is "SHA1") (Version 2.3 and Above)
    - Optional name of hashing algorithm to use if the `hash` option is enabled. The algorithm names to use here are dependent on the cryptographic library that supplies the hashing. In the case of OpenSSL, they're simple names like "SHA1" or "SHA256".

Child Elements

- `KeyInfoResolver` (optional)
    - Allows an alternate implementation to be supplied for mapping the data inside a `<ds:KeyInfo>` structure into a public key. The default implementation used, if no plugin is specified, is an "inline" implementation that understands `<ds:KeyValue>` and `<ds:X509Certificate>` content.

XML AttributeDecoder (Version 2.2 and Above)

Indicated by `xsi:type="XMLAttributeDecoder"`, processes SAML attribute values by directly serializing them as XML and storing them in that form. If exported through the CGI interface, the serialized XML is base64-encoded for that purpose.

DOM AttributeDecoder (Version 2.2 and Above)

Indicated by `xsi:type="DOMAttributeDecoder"`, processes SAML attribute values as an arbitrary XML DOM tree.

This plugin is a somewhat experimental attempt at providing limited support for "rich" XML-valued attribute information. It works best on XML structures without embedded "repeating" elements. Only limited support for accessing the iterated data is included.

The main limitation of this mechanism is that for the SP to provide a useful function, it has to be able to simplify the XML into a string. Otherwise the application might just as well parse the SAML assertion directly, and that may often be the better approach. However, if the XML structure is sufficiently simple, the decoder plugin offers the ability to pull information out of it using a quick and dirty notation as follows:

1. XML attributes and child elements can be jointly accessed by specifying the attribute or element name (or its remapped name, see below).
2. Nested elements can be navigated by separating parent and child with a period (.) character.
3. Lists of repeating child elements can be indexed using the typical array notation ([n]) and the zero-based index of the child to access. Out of range accessors simply return nothing as a resulting string. A zero index is ignored if no list is present.
4. If a list is encountered without an array index specified, the first element in the list is accessed automatically.

As an example, consider this rich SAML 1.1 attribute:

```
<saml:Attribute AttributeName="https://example.org/personalprofile" AttributeNamespace="urn:mace:shibboleth:1.0:
attributeNamespace:uri">
    <saml:AttributeValue>
        <prof:Profile xmlns:prof="https://example.org/personalprofile">
            <prof:Name>
                <prof:First>John</prof:First>
                <prof:Last>Doe</prof:Last>
            </prof:Name>
            <prof:Email>doe@example.org</prof:Email>
            <prof:Email>jdoe@gmail.com</prof:Email>
        </prof:Profile>
    </saml:AttributeValue>
</saml:Attribute>
```

The following attribute declaration will produce a value of "John Doe, jdoe@gmail.com":

```
<Attribute name="https://example.org/personalprofile">
    <AttributeDecoder xsi:type="DOMAttributeDecoder" formatter="$Profile.Name.First $Profile.Name.Last,
$Profile.Email.[1]"/>
</Attribute>
```

Attributes

- `formatter` (string)
    - A required formatting string that turns the XML content into a flat string. The string contains one or more substitution tags consisting of a dollar sign ($) followed by a "path specifier", as described above. Other characters are echoed through to the constructed string.

Child Elements

- `Mapping` (optional)
    - Allows an XML attribute or element to be "mapped" into a shorter or qualified name in the resulting structured representation of the data. The purpose of this feature is to allow for namespace-qualified XML by allowing qualified names to be turned into local "tags" that don't require the two part structure of a qualified name.
        - `from` (QName)
            - A required XML attribute identifying the qualified attribute or element name to map.
        - `to` (string)
            - A required XML attribute identifying the internal name to use when storing the mapped attribute or element.

Base64 AttributeDecoder (Version 2.4 and Above)

Indicated by `xsi:type="Base64AttributeDecoder"`, processes SAML attribute values that are base64-encoded UTF-8 by decoding them back into UTF-8.

This decoder has no way to determine whether the underlying data is in fact UTF-8, so it should be used with trusted IdPs only and with caution. If the data is binary, it will be exposed to applications as raw octets up to the first null character in the decoded data.