EntityAttributesFilter

A filter of type `EntityAttributes` adds or removes SAML entity attributes to or from metadata in order to drive software behavior.

Add entity attributes to remote metadata

The `EntityAttributes` filter is typically used to add entity attributes to remote metadata at runtime. The filter is usually applied to an HTTP metadata provider such as the `FileBackedHTTPMetadataProvider` or the `DynamicHTTPMetadataProvider`.

Contents

The `<mdattr:EntityAttributes>` extension element is a container for entity attributes. Syntactically, an entity attribute looks like an ordinary user attribute. For example:

A simple entity attribute

```
<mdattr:EntityAttributes xmlns:mdattr="urn:oasis:names:tc:SAML:metadata:attribute" xmlns:saml="urn:oasis:names:
tc:SAML:2.0:assertion">
    <saml:Attribute Name="http://macedir.org/entity-category" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
        <saml:AttributeValue>http://refeds.org/category/research-and-scholarship</saml:AttributeValue>
    </saml:Attribute>
</mdattr:EntityAttributes>
```

In the previous example, the name and value of the entity attribute are `http://macedir.org/entity-category` and `http://refeds.org/category/research-and-scholarship`, respectively. Note that an entity attribute may be multi-valued.

To use the `EntityAttributes` filter, sequences of `<saml:Attribute>` elements are supplied as filter content. When a child element such as `<Entity` or `<ConditionRef>` or `<ConditionScript>` evaluates to true, the SAML attributes are applied to the corresponding entities as entity attributes. The software automatically adds or removes the parent `<mdattr:EntityAttributes>` extension element as needed.

Filter order is important!

This filter changes the content of the metadata and so a filter of type `EntityAttributes` should appear after any SignatureValidationFilter in the overall sequence of filters.

Position the EntityAttributes filter for efficiency

Deliberately position an `EntityAttributes` filter in the overall sequence of filters for optimal efficiency. In particular, a filter of type `EntityAttributes` should appear after the EntityRoleWhiteListFilter since the latter effectively removes entities from the input.

As of V3.4, an inbound attribute filtering feature (`<AttributeFilterRef>` or `<AttributeFilterScript>`) allows existing entity attributes to be removed prior to performing any additions. This allows locally applied tags to be blocked from use by external metadata sources, which prevents misuse.

Schema

The `<MetadataFilter>` element and the type `EntityAttributes` are defined by the `urn:mace:shibboleth:2.0:metadata` schema, which can be located at http://shibboleth.net/schema/idp/shibboleth-metadata.xsd.

The schema for the `<mdattr:EntityAttributes>` extension element is part of the OASIS SAML V2.0 Metadata Extension for Entity Attributes specification.

The embedded entity attribute is defined by the `urn:oasis:names:tc:SAML:2.0:assertion` namespace, the schema for which can be located at http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd. The latter namespace is usually associated with the `saml:` prefix.

Attributes

None.

Child Elements

The first two are optional, mutually exclusive, and must appear first:

Name	Description
`<AttributeFilterRef>` 3.4	Optional Bean ID of type Predicate<Attribute>, this is applied to all pre-existing extension attributes and any for which it evaluates **false** are removed prior to subsequent additions
`<AttributeFilterScript>` 3.4	The content of this element is an inline or local script resource that implements Predicate<Attribute>, which is applied to all pre-existing extension attributes. Any entity attribute for which it evaluates **false** are removed prior to subsequent additions.

Then, any of the following can be supplied in any order:

Name	Description
`<saml:Attribute>`	An attribute which is added to all the entities which match any of the following `<Entity>` or `<ConditionRef>` elements
`<Entity>`	The textual content is an entityID. All preceding attributes are added to the entity with this ID.
`<ConditionRef>`	The textual content is the Bean ID of type Predicate<EntityDescriptor>. All preceding attributes are added to the entities for which this returns true.
`<ConditionScript>` 3.4	The content of this element is an inline or local script resource that implements Predicate<EntityDescriptor>. All preceding attributes are added to the entities for which this returns true.

Examples

Add entity attributes to metadata

The following example adds the entity attribute "https://sp.example.org/tagname1" to entity "https://sp1.example.org", and both "https://sp.example.org/tagname1" and "https://sp.example.org/tagname2" to entity "https://sp2.example.org"

Add entity attributes to metadata

```
<MetadataFilter xsi:type="EntityAttributes">
    <saml:Attribute Name="https://sp.example.org/tagname1">
        <saml:AttributeValue>foo</saml:AttributeValue>
    </saml:Attribute>
    <Entity>https://sp1.example.org</Entity>
    <saml:Attribute Name="https://sp.example.org/tagname2">
        <saml:AttributeValue>foo</saml:AttributeValue>
        <saml:AttributeValue>bar</saml:AttributeValue>
    </saml:Attribute>
    <Entity>https://sp2.example.org</Entity>
</MetadataFilter>
```


Shibboleth IdP V3.4 (or later) is required

The remaining examples in this section require V3.4 or later.

Remove entity attributes from metadata

The following example removes unauthorized entity attributes from the input. The metadata filter uses an `<AttributeFilterScript>` to remove any and all entity attributes that might be used to subsequently configure a SAML protocol at runtime. It does this by matching on a common prefix of the overall set of Shibboleth profile URIs.

Remove unauthorized entity attributes from metadata

```
<MetadataFilter xsi:type="EntityAttributes">

    <!-- remove unauthorized entity attributes -->
    <AttributeFilterScript>
        <Script>
        <![CDATA[
            // an implementation of Predicate<Attribute>
            //
            // if the name of the entity attribute starts with
            // a common prefix of the set of Shibboleth profile
            // URIs, the function returns false, which removes
            // the entity attribute from its entity descriptor
            //
            // the input argument is of type:
            // org.opensaml.saml.saml2.core.Attribute
            //
            (function (attribute) {
                "use strict";

                // Shibboleth profile URI prefix
                var prefix = "http://shibboleth.net/ns/profiles";

                // check the parameter
                if (attribute === null) { return true; }

                // check a prefix of the attribute name
                return ! attribute.getName().startsWith(prefix);
            }(input));
        ]]>
        </Script>
    </AttributeFilterScript>

</MetadataFilter>
```


Protect your metadata-driven configuration

An IdP that configures itself on-the-fly using entity attributes should include the previous filter in the overall sequence of filters. The previous filter should appear before any entity attributes are added by subsequent filters. OTOH, if the metadata source is completely trustworthy (e.g., a local metadata source), the previous filter is not necessary. See the MetadataDrivenConfiguration topic for more info.

Add entity attributes that affect signing operations

The following example uses entity attributes to override default signing operations during web browser SSO.

Add entity attributes that affect signing operations

```
<!--
    By default, responses are signed but assertions are not.
    This filter enables signing of both responses and assertions for select entities.
    For other entities, only assertions are signed.
-->
<MetadataFilter xsi:type="EntityAttributes">

    <!-- sign assertions -->
    <saml:Attribute Name="http://shibboleth.net/ns/profiles/saml2/sso/browser/signAssertions" NameFormat="urn:
oasis:names:tc:SAML:2.0:attrname-format:uri">
        <saml:AttributeValue xsi:type="xsd:boolean">true</saml:AttributeValue>
    </saml:Attribute>

    <!-- sign both responses and assertions for the following entity -->
    <Entity>https://sp.example1.org</Entity>

    <!-- do not sign responses -->
    <saml:Attribute Name="http://shibboleth.net/ns/profiles/saml2/sso/browser/signResponses" NameFormat="urn:
oasis:names:tc:SAML:2.0:attrname-format:uri">
        <saml:AttributeValue xsi:type="xsd:boolean">false</saml:AttributeValue>
    </saml:Attribute>

    <!-- sign assertions but do not sign responses for the following entities -->
    <Entity>https://sp.example2.org</Entity>
    <Entity>https://sp.example3.org</Entity>

</MetadataFilter>
```

Add an entity attribute that disables encryption

The following example uses an entity attribute to disable encryption on a specific set of entities in the input. A `<ConditionScript>` adds the entity attribute (`encryptAssertions`) to a collection of entities specified by a Spring bean (`customObjectRef="MyEntityCollection"`) defined elsewhere in the configuration. The collection is exposed to the script via a `custom` object of type `Collection<String>`. Each element of the collection is an `entityID`.

The script takes an argument of type `Collection<String>` (i.e., the `custom` object) and returns a function that implements `Predicate<EntityDescriptor>`. The resulting predicate is applied to the `input` object. The entity attribute is added to the entity descriptor if (and only if) the predicate evaluates to true.

Add an entity attribute that disables encryption

```
<!--
    By default, SAML assertions are encrypted.
    This filter disables encryption for select entities.
-->
<MetadataFilter xsi:type="EntityAttributes">

    <!-- this particular entity attribute disables encryption -->
    <saml:Attribute Name="http://shibboleth.net/ns/profiles/encryptAssertions" NameFormat="urn:oasis:names:tc:
SAML:2.0:attrname-format:uri">
        <saml:AttributeValue xsi:type="xsd:boolean">false</saml:AttributeValue>
    </saml:Attribute>

    <!-- add the entity attribute to a predefined collection of entities -->
    <ConditionScript customObjectRef="MyEntityCollection">
        <Script>
        <![CDATA[
            // this function takes a custom object of type Collection<String>
            // and returns an implementation of Predicate<EntityDescriptor>;
            // the predicate is then applied to the input object
            //
            // the custom argument is of type:
            // java.util.Collection<String>
            //
            // the input argument is of type:
            // org.opensaml.saml.saml2.metadata.EntityDescriptor
            //
            (function (entityIDs) {
                "use strict";

                // return a trivial implementation of Predicate<EntityDescriptor>
                if (entityIDs === null) {
                    return function (entity) { return false; };
                }

                // return an implementation of Predicate<EntityDescriptor>
                // that depends on a custom object of type Collection<String>
                return function (entity) {
                    if (entity === null) { return false; }
                    return entityIDs.contains(entity.getEntityID());
                };
            }(custom))(input);
        ]]>
        </Script>
    </ConditionScript>
</MetadataFilter>
```