NativeSPAttributeExtractor

The <AttributeExtractor> element configures the component used by the SP to turn SAML content into "attributes", the internal/neutral representation of information stored with user sessions.

While there are no specifically "mandated" points at which extractors run, the SP generally invokes extraction following the acceptance of assertions during SSO and as a result of attribute resolution from SAML-based sources such as an Attribute Authority. Actually performing the filtering process is typically up to an Assertion Consumer Service handler (in the case of attributes delivered during SSO) or an attribute resolver.

In general, extractors can be handed many different XML element types and are free to process them or ignore them as their implementation or configuration dictates.

Common Attributes

- type(string)
    - Plugin type name.

Chaining AttributeExtractor

Indicated by type="Chaining", wraps a set of extraction plugins so that they execute in sequence. Obviously, duplication may result of more than one extractor is configured to do essentially the same work.

With V2.4 and above, this is implied by any configuration with multiple <AttributeExtractor> elements, so is no longer explicitly needed.

Child Elements

- <AttributeExtractor>(one or more)
    - Embedded plugins to instantiate.

XML AttributeExtractor

Indicated by type="XML", implements an XML-based rule syntax for designating SAML attributes and name identifiers to decode into internal attributes. The plugin supports extraction of SAML attributes and name identifiers from the following SAML constructs (it does not know how to pull any other data from these elements, only attributes and name identifiers):

- <saml:Assertion>
- <saml:Attribute>
- <saml:NameIdentifier>
- <saml2:Assertion>
- <saml2:Attribute>
- <saml2:NameID>
- <saml2:EncryptedAttribute>

As of Version 2.2, it also supports a specialized form of extraction from the <md:RoleDescriptor> (and its various concrete subtypes) metadata element. This is described in more detail below.

The actual extraction process relies on a secondary layer of attribute decoder plugins that actually consume the XML content.

The XML attribute extractor's XML "portion" is a reloadable resource, which means that the XML content can be supplied inline, in a local file, or a remote file, and can be monitored for changes and reloaded on the fly. The root of the XML instance MUST be an `<am:Attributes>` element.

```
<Attributes xmlns="urn:mace:shibboleth:2.0:attribute-map" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Attribute name="urn:oid:2.5.4.3" id="cn"/>
    <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" id="eppn">
        <AttributeDecoder xsi:type="ScopedAttributeDecoder"/>
    </Attribute>
</Attributes>
```

Attributes

Inherits attributes supported by reloadable resources.

- `metadataPolicyId` (string) (Version 2.2 and Above)
  - Optional reference to a Security Policy to apply to SAML assertions processed in the `<EntityAtributes>` metadata extension (see below).
- `metadataAttributeCaching` (boolean) (defaults to true) (Version 2.5 and Above)
  - When false, disables the caching of decoded attribute information that is normally done to improve the efficiency of extracting attribute information from the `<EntityAttributes>` metadata extension (see below). The usual reason to turn this off is to support language-aware decoding of attribute values.

Child Elements

- `<am:MetadataProvider>` (optional) (Version 2.2 and Above)
  - Supplies a dedicated MetadataProvider for use in validating SAML assertions processed in the `<EntityAtributes>` metadata extension (see below).
- `<am:TrustEngine>` (optional) (Version 2.2 and Above)
  - Supplies a dedicated TrustEngine for use in validating SAML assertions processed in the `<EntityAtributes>` metadata extension (see below).
- `<am:AttributeFilter>` (optional) (Version 2.2 and Above)
  - Supplies a dedicated AttributeFilter for use in filtering data from SAML assertions processed in the `<EntityAtributes>` metadata extension (see below).
- `<am:Attribute>`(one or more)
  - An extraction rule.
    - `name`(string)
      - SAML Attribute/@Name or NameID/@Format to extract from.
    - `nameFormat`(string)
      - Optional SAML @AttributeNamespace/@NameFormat of extraction source (defaults to Shibboleth "standard" URI format constants specific to each SAML version).
    - `id`(string)
      - Primary internal ID of the extracted attribute.
    - `aliases`(deprecated) (space-delimited list of strings)
      - Additional internal IDs for the extracted attribute.
    - `<AttributeDecoder>`
      - Optionally specifies the attribute decoder to use. A simple/string decoder is used if not specified.

General Information

Each `<am:Attribute>` element installs a rule for extracting a named SAML attribute or name identifier into an internal attribute. The source of the attribute is identified with the `name` (and possibly `nameFormat`) XML attributes and internally tagged with the `id` (and possibly `aliases`) XML attributes. The `aliases` feature is deprecated and may be removed from a future release, consider revising to use the AttributeResolver.

The `name` property corresponds to the `Name` XML attribute of a SAML `<Attribute>` element or the `Format` XML attribute of a SAML `<NameID>`/`<NameIdentifier>` element.

The Shibboleth SP by default will install rules using a `nameFormat` of `urn:mace:shibboleth:1.0:attributeNamespace:uri` and `urn:oasis:names:tc:SAML:2.0:attrname-format:uri` to accomodate all SAML versions. The `nameFormat` property can be omitted unless a different `NameFormat` is being used. This property is also omitted/ignored when extracting information from a `<NameID>`/`<NameIdentifier>` element.

The internal ID is typically short and is used throughout all other SP components (such as attribute filters).

Multiple `<Attribute>` elements can share the same `id` and `aliases` values; the implication is that a given internal ID may map to multiple externally-named sources.

Metadata Attribute Extraction (Version 2.2 and Above)

The plugin supports a standard metadata extension, the `<EntityAttributes>` element, which can be used to attach `<saml:Attribute>` and `<saml:Assertion>` elements to an `<md:EntityDescriptor>`. This allows information about a user's IdP to be exposed in the same way that a user's other attributes are.

To ensure they can be distinguished from more typical user data, the trigger for this feature is the `metadataAttributePrefix` application property. Setting this property is both a precondition for metadata attribute extraction and a value that is prepended to the internal attribute names that result. For example, a prefix of "Meta-" will turn an extracted attribute called "mail" into "Meta-mail".

In the case of bare `<saml:Attribute>` elements, often termed "tags", this is the entire picture. Any such elements found within an `<EntityAttributes>` extension are processed identically to user attributes with the same set of decoding and mapping rules. The extension may appear on both the `<md:EntityDescriptor>` and `<md:EntitiesDescriptor>` elements, and the plugin will walk the metadata tree from the role to the entity level up through any parent groups, and process each extension it finds.

At only the entity level, the use of embedded, signed SAML assertions is also supported, but this is quite a bit more complex, and may require additional configuration as follows.

A dedicated MetadataProvider can be defined for the plugin to use when evaluating the assertions. You can think of this as "meta-metadata", definitions of issuers of assertions about issuers of assertions. Issuers of these assertions are required to supply SAML 2.0 metadata with the `<md:AttributeAuthorityDescriptor>` role.

> ⚠ If you don't define a dedicated MetadataProvider for the plugin to use, it will reuse the metadata supplied to the SP as a whole. This may not be the desired behavior in some cases.

Optionally, you can also define an embedded instance of a TrustEngine. If one isn't supplied, the normal one defined to the SP will be used when verifying the signed assertions.

The evaluation process is controlled using a Security Policy that is typically specific to this purpose and is referenced by a `metadataPolicyId` attribute. Normally, rules that would apply to generic assertion processing within the SP, such as replay and freshness checking, do not apply to assertions found in metadata, so a separate policy has to be used.

Note that unsigned assertions are NOT permitted (you'd just use a `<saml:Attribute>` directly).

Finally, you can also define an embedded instance of an AttributeFilter. This enables a special "internal" filtering step to be applied to the attributes extracted from each assertion, separate from the other filtering performed by the SP. Specifically, this filtering step is performed with knowledge of the actual "issuer" of the attributes (the issuer of the embedded assertion). This step is also performed **before** the attributes are internally renamed and prefixed with the `metadataAttributePrefix`.

When the SP performs its standard filtering, the metadata attributes will have been renamed, and the "issuer" is presumed to be the user's IdP itself, because all of the attributes have been pooled by that time.

In other words, you should perform any specialized filtering of metadata-based attributes based on their source by using the dedicated filtering step here, and refer to the attributes by their "unprefixed" names. Other, generic filtering rules based on attribute values (e.g. checking syntax) can be applied using the standard filtering step, referring to metadata-based attributes by the "prefixed" names.

KeyDescriptor AttributeExtractor (Version 2.2 and Above)

Indicated by `type="KeyDescriptor"`, allows the signing/TLS or encryption keys advertised in an IdP's metadata to be exposed as attributes within the SP. This plugin executes only when extraction of an `<md:RoleDescriptor>` (or one of its concrete subtypes) is done, which is dependent on the use of the `metadataAttributePrefix` application setting.

Any public keys that apply (see below) are encoded as DER, using the SubjectPublicKeyInfo encoding commonly used in certificates, and then base64-encoded. At least one of the two attributes below must be specified.

Attributes

- `signingId`(string)
  - If set, public keys marked for signing or TLS authentication will be placed into an attribute with the specified name.
- `hashId`(string)
  - If set, public keys marked for signing or TLS authentication will be placed into an attribute with the specified name. The DER-encoded keys are hashed before being base64-encoded.
- `hashAlg` (string) (defaults to "SHA1") (Version 2.3 and Above)
  - Optional name of hashing algorithm to use if the `hashId` setting is used. 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".
- `encryptionId`(string)
  - If set, public keys valid for encryption will be placed into an attribute with the specified name.

Delegation AttributeExtractor (Version 2.3 and Above)

Indicated by `type="Delegation"`, allows content from within a SAML DelegationRestriction condition to be extracted and passed to an application as an attribute. This allows for finer-grained control over delegation at an SP. The information that's eventually expressed in string form to the application is controlled by a `formatter` XML attribute that can reference specific content from within the `<del:Delegate>` elements in the condition.

Attributes

- `attributeId`(string)
    - Required setting that specifies the internal attribute name to be populated.

- `formatter`(string)
    - An expression containing any number of "substitution" variables starting with a '$' character that reference information from the `<del:Delegate>` element.

The set of `formatter` variables consists of:

- `$Name`, `$Format`, `$NameQualifier`, `$SPNameQualifier`, `$SPProvidedID`
    - Information derived from the corresponding content of the `<saml:NameID>` element found within the `<del:Delegate>` element. Typically delegates are SAML entities that are named by entityIDs and only the `$Name` property is relevant.

- `$ConfirmationMethod`
    - A SAML confirmation method URI that identifies how the delegate confirmed its identity to the IdP.

- `$DelegationInstant`
    - The time at which the delegate confirmed its identity to the IdP.

Assertion AttributeExtractor (Version 2.5 and Above)

Indicated by `type="Assertion"`, allows well-defined content from within a SAML assertion to be extracted and passed to an application as an attribute. This supplements older support for extracting a fixed set of information from the assertion and populating well-defined variables/headers (e.g., the Shib-Identity-Provider header and so forth).

Example equivalent to current standard headers

```
<AttributeExtractor type="Assertion"
    Issuer="Shib-Identity-Provider"
    AuthnInstant="Shib-Authentication-Instant"
    AuthnContextClassRef="Shib-AuthnContext-Class"
    AuthnContextDeclRef="Shib-AuthnContext-Decl"
    SessionIndex="Shib-Session-Index"
/>
```

Attributes

- `Consent`(string)
    - If set, used as the attribute ID for the value of the `Consent` attribute found in the response that delivered the assertion.

- `AuthenticatingAuthority`(string)
    - If set, used as the attribute ID for the value(s) of the `<AuthenticatingAuthority>` element(s) found in the assertion.

- `AuthnContextClassRef`(string)
    - If set, used as the attribute ID for the value of the `<AuthnContextClassRef>` element found in the assertion. Equivalent to the built-in Shib-AuthnContext-Class and Shib-Authentication-Method variables.

- `AuthnContextDeclRef`(string)
    - If set, used as the attribute ID for the value of the `<AuthnContextDeclRef>` element found in the assertion. Equivalent to the built-in Shib-AuthnContext-Decl variable.

- `AuthnInstant`(string)
    - If set, used as the attribute ID for the value of the `AuthnInstant` attribute found in the assertion. Equivalent to the built-in Shib-Authentication-Instant variable.

- `Issuer`(string)
    - If set, used as the attribute ID for the value of the `<Issuer>` element found in the assertion. Equivalent to the built-in Shib-Identity-Provider variable.

- `NotOnOrAfter`(string)
    - If set, used as the attribute ID for the value of the `NotOnOrAfter` attribute found in the assertion.

- `SessionIndex`(string)
    - If set, used as the attribute ID for the value of the `SessionIndex` attribute found in the assertion. Equivalent to the built-in Shib-Session-Index variable.

- `SessionNotOnOrAfter`(string)
    - If set, used as the attribute ID for the value of the `SessionNotOnOrAfter` attribute found in the assertion.

- `Address`(string)
    - If set, used as the attribute ID for the value of the `Address` attribute found in the assertion's `<SubjectLocality>` element.

- `DNSName`(string)

- If set, used as the attribute ID for the value of the `DNSName` attribute found in the assertion's `<SubjectLocality>` element.

Metadata AttributeExtractor (Version 2.5 and Above)

Indicated by `type="Metadata"`, allows well-defined content from within SAML metadata to be extracted and passed to an application as an attribute. This plugin executes only when extraction of an `<md:RoleDescriptor>` (or one of its concrete subtypes) is done, which is dependent on the use of the `metadataAttributePrefix` application setting.

```
<AttributeExtractor type="Metadata" errorURL="errorURL" DisplayName="displayName"/>
```

Attributes

- `AttributeProfile`(string)
  - If set, used as the attribute ID for the value(s) of the `<md:AttributeProfile>` element(s) found in the metadata.
- `errorURL`(string)
  - If set, used as the attribute ID for the value of the `errorURL` attribute found in the metadata.
- `DisplayName`(string)
  - If set, used as the attribute ID for the value of the `<mdui:DisplayName>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `Description`(string)
  - If set, used as the attribute ID for the value of the `<mdui:Description>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `InformationURL`(string)
  - If set, used as the attribute ID for the value of the `<mdui:InformationURL>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `PrivacyStatementURL`(string)
  - If set, used as the attribute ID for the value of the `<mdui:PrivacyStatementURL>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `OrganizationName`(string)
  - If set, used as the attribute ID for the value of the `<md:OrganizationName>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `OrganizationDisplayName`(string)
  - If set, used as the attribute ID for the value of the `<md:OrganizationDisplayName>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.
- `OrganizationURL`(string)
  - If set, used as the attribute ID for the value of the `<md:OrganizationURL>` element found in the metadata. The element chosen is the one that best matches client and/or SP language preferences.

Child Elements

- `<ContactPerson>`(zero or more)
  - Each element defines a rule for extracting `<md:ContactPerson>`element information from the metadata.
    - `id`(string)
      - Name of the attribute to create.
    - `contactType`(one of the SAML-defined contact types)
      - Indicates which type of contact to locate in the metadata for extraction. The first match is used.
    - `formatter`(string)
      - A formatting string used to configure an attribute decoder of `xsi:type="DOMAttributeDecoder"`. For example, "$GivenName $SurName".
- `<Logo>`(zero or more)
  - Each element defines a rule for extracting `<mdui:Logo>` element information from the metadata. The `height` and `width`settings, if provided, are used to find the logo with the smallest total difference in size between the inputs.
    - `id`(string)
      - Name of the attribute to create.
    - `height`(unsigned integer)
      - Input to logo size matching.
    - `width`(unsigned integer)
      - Input to logo size matching.
    - `formatter`(string)
      - A formatting string used to configure an attribute decoder of `xsi:type="DOMAttributeDecoder"`. For example, "".

GSS-API AttributeExtractor (Version 2.5 and Above)

Indicated by `type="GSSAPI"`, implements an XML-based rule syntax for designating GSS-API naming extensions to decode into internal attributes. Using this plugin requires that the **plugins.so** extension library be loaded via the `<Extensions>` element in the `<OutOfProcess>` element.

GSS-API names or contexts can be processed by encoding the exported data in base64, and wrapping in an `<am:GSSName>` or `<am:GSSContext>` element respectively, to meet the constraints of the API, which are based around XML as input.

The attribute extractor's configuration is a reloadable resource, which means that the XML content can be supplied inline, in a local file, or a remote file, and can be monitored for changes and reloaded on the fly. The root of the XML instance MUST be an `<am:Attributes>` element.

```
<Attributes xmlns="urn:mace:shibboleth:2.0:attribute-map" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <GSSAPIAttribute name="urn:ietf:params:gss-eap:radius-avp urn:x-radius:1" id="radius-1"/>
</Attributes>
```

Attributes

Inherits attributes supported by reloadable resources.

Child Elements

- `<am:GSSAPIAttribute>`(one or more)
  - An extraction rule.
    - `name`(string)
      - GSS-API attribute name to extract from.
    - `id`(string)
      - Primary internal ID of the extracted attribute.
    - `aliases`(space-delimited list of strings)
      - Additional internal IDs for the extracted attribute.
    - `authenticated`(boolean) (defaults to "true")
      - If true, only authenticated GSS-API naming attributes are processed.
    - `scopeDelimeter`(character)
      - If set, all values of the naming attribute must contain the character, and it is used to split the value into a two-part construct using a ScopedAttribute object.
    - `binary`(boolean) (defaults to "false")
      - If set, this overrides the `scopeDelimiter` option, and causes the attribute's value to be base64-encoded and handled with a BinaryAttribute object. The unencoded value can be accessed natively in C++ code, but the serialized values are left encoded.

General Information

Each `<am:GSSAPIAttribute>` element installs a rule for extracting a GSS-API naming attribute into an internal attribute. The source of the attribute is identified with the `name` XML attribute and internally tagged with the `id` (and possibly `aliases`) XML attributes.

The internal ID is typically short and is used throughout all other SP components (such as attribute filters).

Multiple `<GSSAPIAttribute>` elements can share the same `id` and `alias` values; the implication is that a given internal ID may map to multiple externally-named sources.
