MetadataCorrectness

Recent changes to the Shibboleth implementation have revealed a number of issues with metadata production by federations that resulted in failures due to increased strictness in rejecting certain invalid XML constructs. To aid federations in the generation of correct metadata, this is a summary of known issues with implementations of interest.

General Statements

Before getting into the details, understand that to be "correct", a SAML metadata file needs to follow a number of syntactic and semantic rules. If it doesn't do so, a software product is free to reject it. You will find that different SAML products behave in vastly different ways, some very strict and some very accepting. The best approach you can take is to be very rigorous about what you produce, and be very wary of accommodating actual bugs in products that require you to "break" your metadata to get them to consume it. If you follow that path, you'll end up maintaining dedicated files in no time, so be sure that's what you want to do.

Obviously syntactic rules are much easier to document and enforce than semantic rules. It's also true that most of the time, it's the syntactic rules that will trip up software globally, while semantic problems are more likely to affect only particular entries in the metadata, which is generally less of a concern.

Syntax Rules Enforced by Standard

The specification itself dictates the following rules and you should not consider it a bug if a product rejects metadata that breaks them, because the standard is unambiguous about them:

- Obviously, metadata must be well-formed XML. It also must be **valid** with respect to the XML Schema defined by SAML for metadata (w/ errata).
- As part of the validity requirement, all content must follow the rules for the data types expressed in the SAML metadata schema. This includes things like characters permitted in ID attributes, date/time attributes, etc.
- SAML enforces additional "default" rules on string and URI data appearing in any XML, including metadata. These rules are right near the beginning of the standard in section 1 of the Core specification. In particular, they **outlaw** empty string or URI values, and require that URI values be absolute (meaning they have to carry a scheme, and cannot be merely a simple string, hostname, etc.). Simply put, empty XML elements or attributes in the metadata result in an incorrect document.
- SAML also requires that all dateTime values be expressed in absolute, UTC form, using the 'Z' character in the ISO field reserved for the time zone, with no time zone offset. Milliseconds MAY be included, but generally should be left out.

Syntax Rules Enforced by Common Sense

There are additional rules that common sense dictates ought to be avoided but are not currently unambiguously forbidden by the standard. It's possible that some of them might be added via the errata process at some point.

- The `<md:ContactPerson>` element should have at least one child element inside it, or be omitted. Simply put, an empty contact serves no purpose and doesn't supply any information to software and ought to be avoided.

Scopes and Default Attribute Values

An additional trouble spot that a few sites have run into is related to the <Scope> extension defined by the Shibboleth Project, which appears in a lot of metadata associated with Shibboleth IdPs. At the time the extension schema was created, the problems of default attribute values when creating signatures were not well understood, and unfortunately the schema includes one, the `regexp` boolean XML attribute that appears inside the `<Scope>` element.

To prevent signature failures in metadata consumers that perform schema validation before verifying signatures, be sure to include an explicit `regexp` attribute value in any `<Scope>` elements.

```
<!-- DO -->
<shibmd:Scope regexp="false">osu.edu</shibmd:Scope>

<!-- DON'T -->
<shibmd:Scope>osu.edu</shibmd:Scope>
```

Encoding of Special Characters

XML uses entity encoding for special reserved characters, that is, characters that have meaning within the grammar of XML itself. For example, '<' is encoded as '<'. In most cases if data isn't properly encoded the XML document will simple fail to parse. However, there are other encoding formats, such as URL encoding. It's important to always use the entity encoding within the XML document and never other forms of encoding, in particular in the endpoint URLs.

So the following is **wrong** because it uses URL encoding, note the %26

```
<AssertionConsumerService>https://example.org/?foo=value%26bar=value</AssertionConsumerService>
```

It should instead be, note the &

```
<AssertionConsumerService>https://example.org/?foo=value&amp;bar=value</AssertionConsumerService>
```

Extensions

In modern usage, SAML metadata often contains extensions of various kinds, defined in specifications that emerged after the original standard was done. Because of how schema validation is done in the presence of wildcards, it is usually the case that older software will (and should) ignore the presence of extensions that it has not been programmed to know about. It is always a bug if this is not the case.

However, over time, extensions may become "known" to software such that elements and attributes that used to be ignored may suddenly be recognized, and in such cases, failure to ensure correct syntax in extensions may break newer software that recognizes the error, even while being ignored by older software.

As a result, it is critical that metadata publishers that include extensions take special care that such extensions are correctly formed, to avoid problems in future versions of software that become aware of the extensions and cause a running system to break after an upgrade.

Shibboleth

At least with respect to Shibboleth, we seek to be strict about what we accept. One reason is that we're a security product, and another is that we have a lot of code that needs to assume the metadata is correct in order to avoid a lot of extra checks for invalid or incomplete information in the rest of the code.

Note that the information below presumes the use of the standard/built-in plugins that handle metadata consumption. All versions of the software include some degree of modularity in that area, but there are no known cases of significant customization on this.

Identity Provider

Version 1.3.x

Mandatory Schema Validation

XML schema validation is done as a matter of course and as a result, any violations of schema validity will prevent metadata from loading. Few, if any, additional checks are performed, but you will probably find that various exceptions arise at runtime if other constraints not enforceable through the schema are violated, such as empty values.

Handling of `validUntil` / `cacheDuration`

The `validUntil` attribute is understood and enforced at any level at which it appears in a metadata file. The `cacheDuration` attribute is ignored.

Handling of `OrganizationURL`

All `OrganizationURL` element values must be correctly formatted absolute URLs or the identity provider will throw a `java.net.MalformedURLException`. For example, placeholder strings such as "unspecified" will cause this behaviour because they don't contain a protocol component such as "http:".

This applies to both the identity provider proper and the `metadatatool` application used to download and verify metadata.

Large numbers of namespaces in metadata

(SIDPO-35): The `metadatatool` application relies on an underlying library for signature verification which throws an array bounds exception in situations where a moderately large number of namespace prefixes are simultaneously in scope. For more details, see this bug report, which indicates that the problem can arise with as few as ten prefixes in scope.

If you are preparing metadata which uses large numbers of SAML extensions each with their own namespace, one way of avoiding triggering this problem in `metadatatool` appears to be to redefine the default namespace on appropriate elements rather than using namespace prefixes, at the cost of a larger metadata file size.

Version 2.x

Schema Validation

Schema validation is off by default and is enabled via a metadata provider filter. Federation operators therefore cannot know for certain whether particular IdPs will fail or not based solely on validation errors, but most will not.

Handling of `validUntil` / `cacheDuration`

All `validUntil` and `cacheDuration` attributes within a metadata document are inspected with the shortest determining when the IdP will next attempt to refresh the metadata. For example, if the shortest `cacheDuration` in the document is 30 minutes and the nearest `validUntil` date is 5 days away than the metadata will be refreshed in 30 minutes. Some metadata providers may provide additional parameters to further control metadata refresh behavior.

Native Service Provider

Version 1.3.x

Mandatory Schema Validation

XML schema validation is done as a matter of course and as a result, any violations of schema validity will prevent metadata from loading. Few, if any, additional checks are performed, but you will probably find that various exceptions arise at runtime if other constraints not enforceable through the schema are violated, such as empty values.

`<ds:KeyInfo>` Extensions

If the SP is running on top of a version of the XML-Security-C library prior to version 1.5.0, there is a bug causing exceptions when loading a `<ds:KeyInfo>` element containing any sort of extension. Their presence causes a failure, even though they're legal. Versions of the SP prior to 1.3.2 actually crash because of an uncaught exception from the library.

Note that simple XML mistakes like leaving off the namespace prefix from a known child element such as `<KeyName>` or `<X509Data>` can result in the child element being treated as an extension even though the intent was otherwise.

In general, you can assume that most deployments prior to 1.3.2 are vulnerable to this bug.

Handling of `validUntil` / `cacheDuration`

The `validUntil` attribute is understood and enforced at any level at which it appears in a metadata file. The `cacheDuration` attribute is ignored.

XML Comments

Another recently discovered problem is triggered by the use of XML comments (<!—>) inside of metadata elements that are text-valued, particularly `<ds:KeyInfo>` elements such as `<ds:X509Certificate>`.

If the comment precedes the intended content, and is preceded by a linefeed, the XML-Security-C library (prior to version 1.6.0) will fail to process the element's value, sometimes with a software exception. This branch of the SP has an exacerbating limitation that results in failure to process an entire metadata file if a single `<ds:KeyInfo>` element within a `<md:KeyDescriptor>` fails to load. So a single comment can be fatal to the entire file.

For a variety of reasons, including the fact that signatures over the metadata will not cover any comments, it is suggested to remove all comments from metadata prepared for consumption by software.

Version 2.0

Schema Validation

Schema validation is off by default and is enabled via a configuration setting (see NativeSPMetadataProvider). Federation operators therefore cannot know for certain whether particular SPs will fail or not based solely on validation errors, but most will not.

<md:EncryptionMethod> Bug

The OpenSAML-C 2.0 release contains a bug that prevents the loading of metadata containing an <md:EncryptionMethod> element within a <md:KeyDescriptor> element. This isn't commonly used, and is ignored entirely by the SP, but some commercial products are known to generate metadata containing them.

Handling of validUntil / cacheDuration

The validUntil attribute is understood and enforced at any level at which it appears in a metadata file.

The cacheDuration attribute was improperly handled by the underlying libraries and may produce unpredictable results, including failures to parse valid metadata. Its use should be avoided if you're supporting this version of the SP.

XML Comments

Another recently discovered problem is triggered by the use of XML comments (<!—>) inside of metadata elements that are text-valued, particularly <ds:KeyInfo> elements such as <ds:X509Certificate>.

If the comment **follows** the intended value, and is preceded by a linefeed, the SP libraries will fail to process the element's value. Unlike older SP versions, such problems will generally be confined to the affected element, or at least the containing role or entity.

For a variety of reasons, including the fact that signatures over the metadata will not cover any comments, it is suggested to remove all comments from metadata prepared for consumption by software.

Version 2.1

Schema Validation

Same as version 2.0.

Handling of validUntil / cacheDuration

The validUntil attribute is understood and enforced at any level at which it appears in a metadata file.

The cacheDuration attribute was improperly handled by the underlying libraries and may produce unpredictable results, including failures to parse valid metadata. Its use should be avoided if you're supporting this version of the SP.

XML Comments

Same as version 2.0.

Version 2.2/2.3

Schema Validation

Same as version 2.0.

Brute Force Validation

In 2.3, or in 2.2 when schema validation is **NOT** enabled, the OpenSAML-C library performs a number of additional checks on the data as the XML is processed that will prevent successful loading of metadata containing some of the syntactic errors noted earlier in this topic. This includes things like empty elements or attribute values and in some cases missing required elements.

These checks were supposed to be performed in the original 2.0 release but a bug was preventing that until 2.2.

Handling of validUntil / cacheDuration

The validUntil attribute is understood and enforced at any level at which it appears in a metadata file.

The cacheDuration attribute is understood only at the **root** of a metadata file and was intended to affect the resulting expiration if doing so would expire the metadata sooner than the validUntil constraint. This was a bad idea in retrospect, but in practice rendered the attribute meaningless because the refresh/reload behavior caused the "clock" to be reset even if the attempt to refresh the metadata failed.

XML Comments

Same as version 2.0.

Version 2.4

Schema Validation

Same as version 2.0.

Brute Force Validation

When schema validation is **NOT** enabled, the OpenSAML-C library performs a number of additional checks on the data as the XML is processed that will prevent successful loading of metadata containing some of the syntactic errors noted earlier in this topic. This includes things like empty elements or attribute values and in some cases missing required elements.

This validation includes certain extensions that were unknown to previous releases, such as:

- `<idpdisc:DiscoveryResponse>`
- Extensions for Algorithm Support
- Extensions for UI/Login

As a result, additional errors will be detected by this version.

Handling of `validUntil` / `cacheDuration`

The `validUntil` attribute is understood and enforced at any level at which it appears in a metadata file.

The `cacheDuration` attribute is understood only at the **root** of a metadata file and is now used sensibly to alter the refresh policy of a metadata file, but has no effect on validity. It can be used to tell an SP to reload metadata more frequently than its default settings would otherwise do.

XML Comments

Same as version 2.0.

Approaches and Tools

In most cases, to validate SAML Metadata you will need:

- a local copy or the URL of a metadata file (e.g., **example-metadata.xml**)
- access to the XML Schema (XSD) file saml-schema-metadata-2.0.xsd
- software to perform the validation process

As explained above, publishers need to ensure that all metadata conforms to technical specifications: XML documents need to be well-formed, schema-valid, and conformant with the additional requirements imposed by the specifications. Of course, metadata will also often contain a digital signature that must be verifiable.

A note on Debian GNU/Linux
While there is mention of Debian and derivatives (Ubuntu, etc.) in this section, only those GNU/Linux distributions listed on NativeSPLinuxInstall are officially supported by the Shibboleth project. Debian packages are supported by the Debian project and/or the DebianShibboleth Team.

Well-Formedness

Verifying that XML files are at least well-formed can often be used to quickly catch simple typos (e.g., when manually editing metadata or configuration files), such as missing or incorrectly-placed closing tags, nested comments (which are illegal in XML), etc.

`xmlwf` from the `expat` package is good for this purpose. Always check all XML files for well-formedness after editing.

But well-formedness alone is not enough, so you'll want to schema-validate all metadata you produce (and possibly what you consume if you're managing metadata update outside of your SAML software).

Schema Validation

Most tools support the loading of XSD files via the network. In those cases you can simply point them to http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd when a reference to the XSD file is required.

This schema file is also included with the OpenSAML package provided by the Shibboleth project (RPM package name `opensaml`, Debian package `opensaml2-schemas`), which by default installs the SAML 2.0 Metadata schema to **/usr/share/xml/opensaml/saml-schema-metadata-2.0.xsd**

You can also refer tools to a local copy, either from one of those packages or by downloading the XSD file and placing it somewhere convenient. The URL is used in the examples below to indicate that loading the schema over the network is possible.

XmlSecTool

Using the contributed XmlSecTool to check for Schema-validity (also checks for well-formedness):

```
xmlsectool.sh --validateSchema --schemaDirectory /usr/share/xml/opensaml/ --inFile example-metadata.xml
```

`xmllint`

On Red Hat, `xmllint` is included in the package `libxml2`, which will already be installed if you're using the Red Hat Network (since it's required by `rhn-client-tools`). On Debian and derivatives, `xmllint` can be installed via `apt-get install libxml2-utils`.

```
xmllint --schema http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd \
--noout example-metadata.xml
```

You can also check a remote XML document against a remote XSD Schema, e.g. testshib's metadata file:

```
xmllint --schema http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd \
--noout http://www.testshib.org/metadata/testshib-two-metadata.xml
```

Shibboleth Service Provider

The `XML` MetadataProvider used by most deployers can optionally schema-validate the metadata it attempts to load, and will prevent loading of schema-invalid Metadata, as well as report complaints to the log (or console) during startup or configuration checks. Simply include the `validate="true"` attribute in the `<MetadataProvider>` element:

```
<MetadataProvider type="XML" uri="http://federation.org/federation-metadata.xml"
validate="true" backingFilePath="federation-metadata.xml" reloadInterval="7200" />
```

Whether validation is enabled or not, the SP will apply a variety of runtime correctness checks, as described earlier under "Brute Force Validation".

To manually check all configured Metadata resources on GNU/Linux systems use:

```
shibd -t
```

For Windows-based systems use:

```
shibd -check
```

See NativeSPshibd for a complete documentation of all `shibd` options.

Signature Verification

In addition to syntactical checking, you might also want to manually check the digital signature on a given metadata instance.

XmlSecTool

Using the contributed XmlSecTool to verify a signature:

```
xmlsectool.sh --verifySignature --certificate metadata-signing.crt --inFile example-metadata.xml
```

`xmlsec1`

The XMLSec Library – based on `libxml2` – can be used to verify a signature:

```
xmlsec1 --verify --pubkey-cert-pem metadata-signing.crt example-metadata.xml
```

or possibly

```
xmlsec1 --verify --id-attr:ID urn:oasis:names:tc:SAML:2.0:metadata:EntitiesDescriptor \
  --pubkey-cert-pem metadata-signing.crt example-metadata.xml
```

On Red Hat install via `yum install xmlsec1 xmlsec1-openssl`. Note that on a RHEL5.5 system I additionally had to create a symlink for the openssl engine library, e.g.:

```
cd /usr/lib && ln -s libxmlsec1-openssl.so.1 libxmlsec1-openssl.so
```

On Debian and derivatives install via `apt-get install xmlsec1`.

`samlsign`

The samlsign command line tool from the OpenSAML 2 toolkit (also included in the `opensaml-bin` RPM packages; for Debian and friends it's in `opensaml2-tools`) is a useful tool for verifying signatures (as well as creating them, in a pinch).

Shibboleth Software

Both the Shibboleth IdP and Shibboleth SP support the use of "filters" to perform digital signature verification when loading metadata. For the Shibboleth SP, the same procedure documented above involving shibd configuration checks can be used to manually evaluate the result of the filtering process.

oXygen XML Editor

Version 10.3 of the oXygen XML editor takes the URL of a signed XML file (such as your federation metadata) and simply returns the result. Example usage: `Tools Verify Signature Input URL:` `http://example.org/metadata.xml`.