Metadata

Metadata is a heavily overloaded term, but with regard to SAML (and Shibboleth), it refers to configuration data used to provision an SP or IdP to communicate with each other. Typically it exists in XML form, at least for publishing and interchange.

As a novice, it's going to be virtually impossible to approach learning about metadata unless you understand XML well enough to read and write it. We've found the w3schools site to be a good source of introductory material. Understanding namespaces is particularly important.

Technical Specs

There are several metadata schemas defined by different specifications or software, but Shibboleth is currently designed around the SAML 2.0 Metadata specification standardized by OASIS.

Shibboleth also supports "profiles" of this specification for use with other identity protocols, including SAML 1.x and WS-Federation. The SAML 1.x profile has also been standardized by OASIS.

Shibboleth also includes some extensions to the core metadata schema; for technical details refer to the Shibboleth Metadata Extension specification.

Help with Correctness/Validation

For information on what goes into ensuring metadata is properly formatted, refer to the MetadataCorrectness topic.

Use (and non-use) of Metadata

Shibboleth, in its current state, does not offer tools to import or export SAML metadata. Rather, it consumes the XML directly as a configuration mechanism that enumerates the set of trusted partners and tells the software how to communicate securely with them. The IdP software has support for communicating with "unknown" SPs to some degree, as long as the POST profile or binding is used. However, there is currently no facility in the SP for accepting assertions from "unknown" IdPs. This will trigger relatively common "metadata lookup" errors.

The IdP consumes metadata by looking for entities that act in SP roles. Conversely, the SP consumes metadata by looking for entities that act in IdP roles. In other words, each type of provider needs metadata about its opposite. In a few isolated cases, the IdP also relies on metadata about itself.

In all cases, it is critical that the metadata supplied to other systems be "in sync" with the configuration of the system it describes, or many different kinds of errors will result. In fact, most of the errors that people encounter with SAML software are caused by mismatches of this nature.

General Structure

Entities and Groups

SAML metadata files are rooted in one of two elements:

- `<md:EntityDescriptor>` (describes a single deployment)
- `<md:EntitiesDescriptor>` (describes a group of deployments)

The latter, which simply wraps one or more of the former, is more common in production Shibboleth use because it enables a bunch of IdPs or SPs to be described at once, and then signed as a unit. This is a common way for federations to supply metadata about their members. It's also frequently used in deployments where metadata has to be maintained more or less by hand, because it allows a bunch of information to be maintained in one file, with the entire set of metadata supplied to an IdP or SP with a single configuration element. It's usually easier that way than to individually supply metadata about a single entity at a time using multiple metadata sources.

An "entity" is just a server that's running SAML software to perform some task, such as an IdP or an SP. Each entity has a unique name, an entityID, that distinguishes it from any other. You are responsible for choosing an appropriate URL to use as an entityID, and both your configuration and the metadata you publish will contain that value. If they don't match, many problems will result.

Incomplete EntityDescriptor Example

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https://webauth.example.org/idp
/shibboleth">
... example content not shown ...
</md:EntityDescriptor>
```

Shibboleth-Specific Tip

With the example above, you would need to ensure that your 2.x IdP included a `provider` setting of "https://webauth.example.org/idp /shibboleth" in the various `<RelyingParty>` definitions.

Roles

Once you get past the "entity" level, the main information unit is the "role". An entity is structured as a sequence of one or more "role descriptors", followed by optional contact/organization information for support purposes. The role elements make up the "meat" of the metadata you need to create/supply. In the case of Shibboleth, the most important roles tend to be `<md:IDPSSODescriptor>` and `<md:AttributeAuthorityDescriptor>` in the case of metadata about IdPs and `<md:SPSSODescriptor>` in the case of metadata about SPs.

For details and examples at this level of detail, please refer to the MetadataForIdP and MetadataForSP topics.

One extremely important piece of information common to all role elements is the `protocolSupportEnumeration` XML attribute, which **MUST** be present. This attribute contains a space-delimited collection of URIs that represent general classes of protocol support for the role in question. There are URIs defined by the various standards and profiles to represent the fact that an entity acting in a role "supports" a particular protocol family, such as SAML 2.0 or the Shibboleth profile of SAML 1.1.

Shibboleth-Specific Tip

If you omit this attribute or put the wrong values into it, you will find that the metadata you're attempting to supply is essentially invisible for all intents and purposes. If the right protocol string isn't included, the software will skip right over the information as if it weren't there.

A summary of the values typically encountered follows:

Protocol Family	URI to Include in `protocolSupportEnumeration`	Roles?
SAML 2.0	urn:oasis:names:tc:SAML:2.0:protocol	IdP, AA, SP
SAML 1.1	urn:oasis:names:tc:SAML:1.1:protocol	IdP, AA, SP
SAML 1.0	urn:oasis:names:tc:SAML:1.0:protocol	IdP, AA, SP
Shib 1.x SSO Request	urn:mace:shibboleth:1.0	Shib IdP
WS-Federation	http://schemas.xmlsoap.org/ws/2003/07/secext	Shib IdP, Shib SP

Incomplete Example of IdP Supporting SAML 2.0 and Shib/SAML 1.1

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https://webauth.example.org/idp
/shibboleth">

  <md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol urn:oasis:names:tc:SAML:
1.1:protocol urn:mace:shibboleth:1.0">
  ... role content not shown ...
  </md:IDPSSODescriptor>

</md:EntityDescriptor>
```

Incomplete Example of SP Supporting SAML 2.0 and Shib/SAML 1.1

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https://service.example.org
/shibboleth">

  <md:SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol urn:oasis:names:tc:SAML:
1.1:protocol">
  ... role content not shown ...
  </md:SPSSODescriptor>

</md:EntityDescriptor>
```

Note that the order of the URIs in the attribute is irrelevant and does not represent any kind of priority.

Contacts and Organizations

Beyond the role level, the rest of the metadata about an entity consists of optional `<md:ContactPerson>` and `<md:Organization>` elements that provide information about who is standing up a particular entity and how to contact them.

For testing purposes, you will rarely if ever need to supply these elements, but they may be needed for production use. Organization metadata in particular often gets used by other software systems that consume metadata in order to present lists of entities with human-readable names. Examples of such systems include IdPDiscovery services or software to assist users in granting consent for login and release of attributes to SPs.

Incomplete Contact/Organization Example

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https://webauth.example.org/idp
/shibboleth">

... role(s) not shown ...

  <md:Organization>
    <md:OrganizationName xml:lang="en">Example Organization, Ltd.</md:OrganizationName>
    <md:OrganizationDisplayName xml:lang="en">Example Organization</md:OrganizationDisplayName>
    <md:OrganizationURL xml:lang="en">http://www.example.org/</md:OrganizationURL>
  </md:Organization>

  <md:ContactPerson contactType="support">
    <md:GivenName>Authentication Support</md:GivenName>
    <md:EmailAddress>webauth-support@example.org</md:EmailAddress>
  </md:ContactPerson>
  <md:ContactPerson contactType="administrative">
    <md:GivenName>John</md:GivenName>
    <md:SurName>Doe</md:SurName>
    <md:EmailAddress>jdoe@example.org</md:EmailAddress>
  </md:ContactPerson>

</md:EntityDescriptor>
```

Note that the three child elements of the `<md:Organization>` element **MUST** carry the special `xml:lang` attribute.

Methods for Supplying Metadata

Broadly speaking the Shibboleth IdP and SP support roughly similar mechanisms for acquiring metadata, though the details vary. You can see the technical specifics in the IdPMetadataProvider (V2) / MetadataConfiguration (V3) and NativeSPMetadataProvider topics, but in general, there are three approaches one tends to see:

Locally Maintained Batches

One simple approach involves maintaining the metadata in files that you copy to the IdP or SP server to load from the local filesystem. The idea behind this approach is to use some kind of out of band method to actually acquire the metadata "securely" and then place it onto the server. It can typically be refreshed and automatically re-read by the software when it changes. Note that with this approach, it becomes a separate responsibility to determine if the metadata is valid and how to update it when it changes.

Shibboleth-Specific Tip

On the IdP, you would typically copy the metadata to the **shibboleth-idp/metadata** folder. On the SP, it would typically be copied to the **etc/shibboleth** folder with the rest of the SP configuration files.

One common use for this approach is for an IdP that is serving a local stable of SPs across an organization. Often, the IdP administrator is directly registering the SPs somehow and may be manually creating metadata for them.

Remote Batches of Signed Metadata

This is the "classic" model of federation in Shibboleth, and relies on a third party of some kind to create, sign, and host metadata for a collection of systems. These batches of metadata are in many cases the most tangible evidence of the federation itself.

With this model, the IdP or SP can consume the metadata directly from a published URL and verify the signature at runtime using a well-known key or certificate, or in more advanced setups, based on validating a metadata signing certificate.

Distributed / Endpoint-Supplied Metadata

The most distributed model is to serve up metadata directly from the system it's describing. This can be done either by treating the source as a case of the previous option (a remote "batch" of 1 entity), or by actually resolving the entityID directly as a URL to the metadata.

While this is a useful approach for testing, in practice this is a very difficult model to use securely, because it still requires some means to verify that the metadata is correct. Relying on a third party signature is quite workable (it really just turns into the previous case), but in most cases when people talk about distributing the metadata in this way, they mean that the metadata is signed by the entity itself or not signed at all. That may require a great deal of additional public key infrastructure, and places a lot of trust in an entity to describe itself accurately.

Bootstrapping Trust

In Shibboleth, from an API point of view, there is no "questioning" the information in the metadata you supply. It is implicitly trusted, so any rules for evaluating the information have to be supplied inside the metadata source, as a filter, or implemented in some out of band way. This is a particularly important point because of the use of metadata by trust engines to make decisions about keys. By implicitly trusting metadata, the problem of how to bootstrap trust within the system is addressed.

Other SAML implementations may have completely different approaches to managing trust, usually based on PKI in some fashion. It can be very challenging to interoperate between arbitrary implementations with different models for managing trust.
