Assurance - Service Provider Behavior

- Service Providers and Assurance
- Runtime Implications
- Questions
  - What does the standard require an SP to do?
  - Can an SP ask for more than one IAQ and can the IdP return more than one?
  - Does the IdP always provide an IAQ?
  - Can I limit the IAQs that an IdP might assert?
  - Can I discovery what IAQs an IdP supports?
  - What <samlp:RequestedAuthnContext> matching rules are recommended, or acceptable?
  - How is an SP supposed to "know" that Silver is acceptable in lieu of Bronze?
  - How should an SP perform "escalation" to allow for increasing LoA in an application when a user transitions from a context that needs little or no assurance, to a context that requires a higher assurance?
- Advice to SP Deployers
- Software-Specific Information
  - Shibboleth
    - Using the IAQ
    - Limiting and filtering IAQs
    - Requesting an IAQ
    - Handling Errors

Service Providers and Assurance

A typical SP today tends to ignore identity assurance or authentication quality issues because contracts and other compensating controls are used to deal with risks "out of band". An occasional exception is the checking of "authentication method" values from a SAML assertion to require stronger authentication such as hardware tokens or certificates. These are most commonly all or nothing exercises, usually internal to an organization, and thus subject to a lot of explicit configuration by the IdP to support.

A partial goal of the Assurance Program is to move more of these considerations "in band" to support more complex requirements and in particular to support applications that need to negotiate for higher assurance in real time or that may have differing requirements based on what the user is trying to do.

At least in SAML, the theoretical basis for using the `<saml:AuthnContext>` construct to represent assurance is that the protocol already includes features for requesting and negotiating the right result. In practice, this becomes complex as the use case becomes more complicated; some of the reasons can be found in the parallel discussion regarding Identity Provider Behavior.

Runtime Implications

Some SAML SP implementations do not support the use of `<samlp:AuthnRequest>` messages at all, or do not allow for the use of the `<samlp:RequestedAuthnContext>` element to specify the SP's requirements. In such a case, IdPs would have to allow for out of band configuration of their behavior based on the identity of the SP. It may also be possible to supplement the SP with application code that can generate a request on behalf of the broken implementation.

In turn, the SP implementation may or may not have the ability to actually consume the IdP's asserted `<saml:AuthnContext>` information, or affect application behavior based on it. In that scenario, the SP is essentially back to relying entirely on out of band assumptions; this is essentially what most deployments do today.

So for our purposes let's assume these constraints don't hold, at least to some degree. Ideally then, SPs that require a particular assurance level (or one of a set) will initiate the assurance flow by including the desired identity assurance qualifier (IAQ) in the `<samlp:AuthnRequest>` message. SPs should understand that asking for a particular IAQ implies that the result may be a SAML error rather than a successful authentication response, because the IdP may be unable to comply. (Errors are always possible of course, but they are more likely in the presence of `<samlp:RequestedAuthnContext>`.) To avoid such an error, the SP and the IdP need to agree on a catch-all value that means "or anything you can handle". InCommon recommends using `urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified` for this purpose.

SPs will receive IAQs (either in response to a specific request, or sent unsolicited) in assertions from IdPs. Exactly one IAQ is available, but some IdPs may provide values that predate the assurance program (often a signal that the user authenticated with a password) or may provide an IAQ that the SP does not recognize.

SPs should use metadata to check that the IdP is authorized to assert the IAQs being asserting.

Questions

What does the standard require an SP to do?

If the SP intends to request one or more qualifiers, then it has to determine what it's prepared to accept, and where appropriate, configure itself to include the acceptable values in its `<samlp:AuthnRequest>` messages.

It should also be prepared to handle errors that result from that change. As with most authorization failures, users typically won't understand why these happen or how to fix them (and true authorization implies that there isn't always a fix).

Can an SP ask for more than one IAQ and can the IdP return more than one?

Yes, and no, respectively. An SP (if supported by software) can enumerate a set that it supports, but the IdP has to pick a single one. As a result, rules within an application generally have to handle the relationships among IAQs, if they exist, unless it's prepared to request only a single one and require all IdPs to return only that, and always do so.

Does the IdP always provide an IAQ?

Not technically, but some form of authentication context is always supplied. An IdP may use an IAQ, or might fall back to older practice, passing specific technologies back instead. If the SP doesn't specify anything, it must be prepared for anything.

Can I limit the IAQs that an IdP might assert?

The standard does not itself address policy issues associated with controlling when an IAQ might be asserted, or by whom. Some software may provide such features. Furthermore, while IdPs are obviously expected to follow the standard, the only prevention mechanism an SP has to limit what it might get is ultimately to check and refuse (or limit) access.

The federation, however, does provide an explicit statement that associates the IAQs an IdP has been certified for in its metadata. The mechanism for this is described in Assurance Technical Implementation Considerations.

Can I discovery what IAQs an IdP supports?

Not in general. The IdP's SAML metadata may provide a hint (see previous question), but the policy statement from the federation that indicates what an IdP has been certified to do does not imply that the IdP will in fact provide a given IAQ for any particular user or transaction. In other words, you can winnow the set to exclude non-certified IdPs, but not further. This is in general impossible since the question may depend on the user and other factors beyond the IdP's control.

What `<samlp:RequestedAuthnContext>` matching rules are recommended, or acceptable?

In practice, the only (even minimally) supported matching rule one will tend to see is "exact", in which the Idp MUST select one of the set requested or return an error. Over time we may see the need for other values, particularly "minimum", but the most likely outcome for the time being would be failure.

How is an SP supposed to "know" that Silver is acceptable in lieu of Bronze?

More generally, how does an SP know the relationship between different IAQs? The answer is that for now this is local policy, probably influenced by community and federation advice. In fact, it may not be true in all cases that higher assurance is acceptable; consider if it costs the IdP or SP money per transaction in a particular case.

So SPs will need to rely on local policy to decide how to handle incoming IAQs. For example if the SP requires InCommon Bronze, does not specify this, but receives InCommon Silver, that is probably acceptable. If an SP is not able to apply judgements to what it receives, it should probably specify exactly what values it will accept in its request.

How should an SP perform "escalation" to allow for increasing LoA in an application when a user transitions from a context that needs little or no assurance, to a context that requires a higher assurance?

SAML doesn't address this specifically because there's really nothing to support at that layer; it's really a matter for the SP and application to address by ensuring that crossing such a boundary results in a second request to the IdP. The main concern is really state. If the application's state is maintained separately from the SAML implementation, then simply re-invoking the SAML authenticaton step with different request content may be sufficient.

Advice to SP Deployers

The specific practices to follow depend largely on the use case for assurance that one has. We are collecting material on use cases identified by the community as valuable.

A concrete piece of advice across all use cases is to document as extensively as possible what your service's requirements, assumptions, and expectations are with regard to assurance so that IdPs can evaluate their systems, and understand the errors that their users might be experiencing. One would expect in the short run that most assurance-requiring services are going to involve some degree of setup and testing, so this documentation will be critical. Think of it as akin to the material used to define attribute requirements for users.

Software-Specific Information

Shibboleth

Using the IAQ

The SP extracts and makes available the IdP's asserted IAQ for a session in a variable called "Shib-AuthnContext-Class". The exact name and syntax for accessing it will vary by programming language (see https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPAttributeAccess).

Limiting and filtering IAQs

The SP also supports extracting the "tag" applied by InCommon to the IdP so that applications can build authorization rules based on it to control which IdPs can assert an IAQ. This relies on an advanced feature called "metadata attribute extraction".

1. Edit the SP's **attribute-map.xml** configuration file. Add the following new element to the config:

```
<Attribute
    name="urn:oasis:names:tc:SAML:attribute:assurance-certification"
    id="assurance-certification"/>
```

This corresponds to the `<saml:Attribute>` element in the IdP's metadata.

2. Edit the SP's **shibboleth2.xml** configuration file. Add the following attribute to the `<ApplicationDefaults>` element:

```
metadataAttributePrefix="Meta-"
```

You will now have an `<ApplicationDefaults>` element with (more or less) the following:

```
<ApplicationDefaults
    id="default" policyId="default"
    entityID="https://example.org/shibboleth"
    REMOTE_USER="persistent-id targeted-id eppn"
    metadataAttributePrefix="Meta-">
```

This will add new environment variables with the prefix `Meta-` and allow the SP software to automatically populate the server environment with IAQ certifications from the IdP's metadata. This is useful for the SP to programmatically determine which assurance attributes are valid from the IdP, or to build authorization rules. These variables or headers behave exactly as user attributes do, and can be used interchangeably (the prefix is used to tell the difference).

Requesting an IAQ

To populate the `<samlp:RequestedAuthnContext>` element in the SP's requests, the `AuthnContextClassRef` content setting can be used.

Content settings are properties that can be set in a number of places in the SP to associate settings with resources, commonly via the Apache `ShibRequestSetting` command or the `<RequestMap>` in **shibboleth2.xml**, or by passing them as parameters to "/Shibboleth.sso/Login" if the SP's lazy session feature is being used to generate requests.

Currently, this mechanism only allows for a single IAQ to be requested. To include multiple values in a request, the AuthnRequest "template" mechanism described in the SessionInitiator documentation can be used.

Handling Errors

In the case that an IdP does not support the SP's requested IAQ for whatever reason, the user may be left at the IdP (in which case the error is up to the IdP to handle), but in most cases, the IdP will return a SAML error response to the SP. Sometimes, but not always, this response will contain SAML status information identifying the problem.

The Shibboleth SP only minimally handles this, by surfacing it as a "FatalProfileException" and displaying an error template. Most of the time, such errors represent unusual or rare conditions, which makes them different from this one. As a result, we recommend that deployers experiment with the SP's `redirectErrors` content setting and route errors to an application script to deal with. Complete documentation on error handling is available.