Assurance - Identity Provider Behavior

Identity Providers and Assurance

A typical IdP today tends to be configured with an authentication mechanism that is relatively indifferent to the identity of the user or service. A few deployments will implement the ability to choose from among a couple of different login technologies, and a smaller number might actually leverage the identity of the service to determine what to do, such as offering a PKI or multi-factor UI for a high-value application.

The use of assurance is an evolution in the maturity of these approaches that generally needs to know more about the service and the user to do the right thing. In this case, the right thing includes getting the right IAQ into an assertion's `<saml:AuthnContext>` element. It may also include limiting or altering the behavior of the UI, but this varies depending on the deployment.

As a couple of examples, consider two common cases:

- A deployment in which all users authenticate in a common way regardless of assurance level, such as a case in which all passwords are managed so as to meet InCommon Silver requirements. Users may or may not all be proofed at a level that meets a single assurance standard, but the authentication step is the same.
- A deployment in which the assurance level is in part determined by whether a two-factor solution is used in place of just a password, such as a case in which users must use a one-time password from a device for an assertion to carry the InCommon Siver designation.

If we consider the implications of the latter case, note that users will likely expect the option to stick with a password unless they interact with a service requiring the higher assurance level. This means the IdP needs to know how the user has already authenticated, if at all, and the identity of the service and its assurance requirements. Contrast that with the former case, in which a single mechanism is used regardless. Obviously this is simpler, at the cost of imposing tougher password requirements on all users.

Finally, consider that in either case, the user's actual assurance level may well depend on directory information recorded about the user reflecting the identity proofing done. That information might even change based on a remote password reset that temporarily lowers the assurance level. It is likely that the IdP will need to consult this information before determining the actual IAQ it can assert. This is in general not something that most software does "out of the box" today.

Runtime Implications

At the SAML layer, it has been assumed that if `<saml:AuthnContext>` is used to carry assurance information, then the best user experience would be obtained if the SP simply specifies its desired IAQ in its authentication request. This is complicated by a few considerations:

- Many SAML implementations, both IdP and SP, lack support for requesting or processing requested context information.
- Many large deployers of applications use overly-simplistic SAML gateways that hide the specific identity of the application being accessed. This means applications that have varying requirements don't work well behind a single gateway.
- SPs may want IdPs to provide an IAQ but still accept assertions without them in return for providing lesser service.
- SPs may want to request an IAQ only when a particular IdP **and** user supports it.
- SPs may support a variety of IAQs across different communities and may not know which to request.

To move forward we will be collecting use cases and identifying best practices as we learn more about the problem.

We know that most IdPs today will have to adjust their behavior to some degree to incorporate support for assurance. This will in most cases involve customizing the IdP via mechanisms specific to an implementation (see below for some more on this). Over time, we would expect to see some commonality in these extensions and can provide exemplars.

We know that a lot of implementations don't have full support for the SAML features involved and conservative choices will have to be made, though these may be balanced against the likelihood that doing anything may require some degree of customization anyway.

Questions

What does the standard require an IdP to do?

If an IdP receives a request from an SP that does not include a `<samlp:RequestedAuthnContext>` element (that is, doesn't ask for a particular IAQ), it is free to more or less choose any authentication approach and supply anything in the response that it chooses, as long as it's accurate in what it tells the SP.

Alternatively, an SP can ask for any of a set of "matching" authentication context classes (which we use to express IAQs), with the matching based on one of a set of predefined rules.

The most common, the default, and about the only one implemented at all, is "exact". It means "exactly" that. The response from the IdP must include one of the values requested, and it must match exactly, not by means of an "equivalence" test. For example, if the SP asks for "A", and "B" is just as good, the IdP cannot use "B", but only "A. Of course, the IdP is expected to only do this if it actually **can**. If the user can't authenticate well enough or their account doesn't qualify, then it can't fulfill the request. An error is defined in SAML to signal this result to the SP, which is expected to handle the outcome.

The other matching rules defined are "minimum", "better", and "maximum". Their definitions can be found in Section 3.3.2.2.1 of the SAML Core specification. They have not seen wide adoption, and most IdPs would require customization to handle them. "minimum", if any, is the one likely to see interest in the future.

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

Yes, and no, respectively. An SP can enumerate a set that it supports, but the IdP has to pick a single one. As a result, either rules within an application generally have to handle the relationships among IAQs, or it would need to request only a single one and require all IdPs to return only that value.

Both options can work, but have their advantages and disadvantages. If the SP handles the problem, then it must be configured with all of the possible IAQs it can accept, and may fail to accomodate new values until modified to do so. Relying on IdPs to accept all the values they can satisfy relieves the SP of this problem, but means more complex rules (and probably custom code) for the IdP. In practice, while there are more SPs than IdPs, SPs tend to be easier to modify with new values.

It is possible in theory to define "super-IAQs" that operate above the level of a single one and express combinations or alternatives, but in practice this doesn't scale well, and produces essentially the same outcomes anyway.

Does an IAQ always have to be asserted by an IdP?

Not technically, but **some** form of authentication context does. An IdP could use an IAQ, or could fall back to older practice, passing specific technologies back instead. For consistency, using the SAML-defined constant "urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified" is a good idea.

Advice to IdP Deployers

- Expect that your deployment will need to change, just as other elements of your IdM infrastructure have to change to accomodate assurance.
- Think about whether you expect to have a single authentication method for all assurance levels or more than one, and plan accordingly.
- Ensure that the determination of assurance level for a given user can be determined based on information your IdP is likely to have access to, such as the authentication method and your directory information.
- Use the "urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified" constant as a default context class in the absence of assurance and migrate any local applications looking for other "default" values to that value instead.
- Document how you expect assurance to work for your IdP so that you can collaborate with other campuses in similar situations using the same software.

Software-Specific Information

Shibboleth

The documentation available for the IdP's authentication support is available on the Shibboleth wiki..

For information about configuring Shibboleth IdP to support specific use cases to support InCommon's profiles, see contributed documentation on this wiki:

- IdP Configuration for Username-Password with Silver Assurance

In a nutshell, the IdP defines a concept called "authentication method" that is analagous to the concept of an authentication context class, and thus an assurance qualifier. "RelyingParty" definitions in the IdP can carry a default method that stands in for a specific requested value from an SP. The result is used to identify one of a set of installed login handlers to use for the request. The built-in handlers are limited to very simple use cases in which a fixed value is passed back from the handler identifying a single method, and that method ends up in the resulting assertion.

The "exact" matching type is supported, and a given login handler can be associated with multiple methods so that different requests can succeed. However, the built-in handlers cannot themselves pass back any but a fixed value, which will simply cause an error if the request was for a different value. In other words, some of the machinery supported is specifically designed for use by custom login handlers and not the built-in examples. The exception is the handler designed for plugging in "external" authentication systems like a separate SSO system. This approach allows for some degree of customization already.

In addition, the IdP does two things worthy of note:

- It treats the "unspecified" constant noted earlier as a wildcard that matches any supported authentication method. This is done for consistency with the handling of the "unspecified" `<NameID> Format`.
- It favors methods that are "active" for the user's session (i.e., methods the user already recently performed) when looking for a matching method to satisfy a request.

The latter means that SSO is more likely, but that specifying a stronger method first doesn't necessarily mean it will get used, even if the user could in theory satisfy it. It also means that including "unspecified" in a request will imply that any active authentication method for the user will potentially get used before any stronger method would have the chance to be evaluated, even if the stronger method were placed earlier in the requested list.

As of V2.3.4, all handlers must signal the method used back to the IdP, to prevent certain security problems when handlers of different kinds are available. Older versions work, but are likely to behave correctly only in certain cases depending on the mix of handlers installed.

The simplest way to extend the IdP and ensure control over what happens is by installing a single custom login handler that itself contains all of the logic needed to differentiate what to do based on the SP, the request, and the user. The handler can be associated with as many "methods" as necessary to cover all of the possible values involved, and the IdP will know to invoke it in all cases. Alternatively, if the release is new enough, you could isolate support for different methods into different custom handlers if you prefer that approach.

In any case, if the resulting assurance value depends on information from among the user's attributes, then a custom login handler is probably required, because none of the existing handlers have support for invoking the attribute resolution step in the IdP.

Resources and Help

There is an unsupported example of a custom login handler that includes attribute resolution on the Contributions page under "Stateless Cluster SSO". It does not explicitly deal with assurance, but is itself extensible.

Questions about how to use the IdP to support assurance can be directed to the users@shibboleth.net mailing list.