FlowsAndConfig

How it All Fits Together

Shibboleth has two major halves: an identity provider (IdP), and a service provider (SP). The identity provider supplies information about users to services, and the service provider gathers information about users to protect resources. In the typical use case, a web browser accesses a protected resource, authenticates at their identity provider, and ends up back at the resource logged in.

How does this actually happen, and how does it fit with IdP and SP configuration? What other pieces are involved?

1. User Accesses Protected Resource

A user tries to access a protected resource, causing the SP to intercept the request. The resource locations to protect can be defined in the web server configuration itself, such as `httpd.conf`, or in `shibboleth2.xml` (or a separate file) using the `<RequestMap>`.

2. SP Determines IdP and Issues Authentication Request

The SP will select a SessionInitiator to use based on this protection configuration, which in turn is responsible for determining which IdP the user will be referred to and what protocol to use. The providers signal their profile preferences to one another through the exchange of SAML metadata.

The process of determining the IdP to use is called IdP Discovery and can include a combination of configuration options, various web-based interactions, cookies, and other techniques. A SessionInitiator might supply a text entry box, refer the user to a locally or remotely deployed discovery service (DS), or select a fixed IdP based on the resource requested.

In some legacy configurations, the SP may use an older-style discovery mechanism called a "WAYF" service. In this case, the request for authentication is assumed to be a legacy Shibboleth 1.x request and is issued to the WAYF itself, which then relays the request to the IdP once the selection is made. This approach is limited to SAML 1.x use, does not interoperate with most other SAML implementations, and is not recommended for new installations.

Cookie Set by SP

During this step, the SP will preserve the original resource requested by the browser using a "relay state" mechanism, which is configured by a `relayState` property on the `<SessionInitiator>` element. The default mechanism does not rely on a cookie any longer, but many systems do, and send a state management cookie containing the resource URL to the client along with the request prepared for the IdP or DS/WAYF.

3. User Authenticates to the IdP

An authentication request is issued by the SP to the IdP as a result of the previous step. The format of this request depends on the protocol and binding /profile selected by the SP. The authentication request is passed through the browser, and the client is redirected (via GET or POST) to an endpoint at the IdP typically called a "Single Sign-On Service".

The IdP examines the request and decides how it would like to authenticate the user based on rules established for the SP in *relying-party.xml* and authentication in general. The user is redirected to a compatible login flow, authenticates (or tries to) using the method selected, and eventually control passes back to the profile implementation with their username determined.

Cookie(s) Set/Read by IdP

Typically the IdP will attempt to read and set one or more cookies during the authentication sequence, but the specifics vary based on the form of authentication used. In general, the IdP will establish a session cookie in order to track the client's progress through the request processing steps, as well as to maintain a longer term association for the purposes of SSO. Additional cookies could be involved in the authentication process depending on the login handler used.

4. IdP Issues Response to SP

The IdP now uses the principal's name, the SP, and the protocol and binding/profile selected to decide what information to send the SP and how to package it.

First, the IdP gathers a set of attributes for the user using the attribute resolver. It collects user data from all the backend sources, transforms it if necessary, and attaches encoders to each attribute.

These attributes are passed through the attribute filter, which may pare down the information to be included in the response. The set of attributes released most often depends on the SP and the principal. This protects the user's privacy. The resulting information could be as little as "someone authenticated successfully", or reveal any attribute you can imagine.

The user's information is packaged into a form suitable for the eventual response using the encoders attached earlier, typically in a SAML assertion. This assertion may be signed with the IdP's key and, in the case of a SAML 2.0 assertion, encrypted with the SP's key for security and privacy. The assertion (or a reference to it called an artifact) is placed into a response that is passed through the client browser for delivery back to the SP to an endpoint called an Assertion Consumer Service.

5. Back to the SP

The browser delivers the response from the IdP to an Assertion Consumer Service endpoint at the SP. The ACS implementation decodes the message, decrypts the assertion if necessary, and performs a variety of security checks. If everything is in order, then the SP will create a new user session after extracting attributes and other information from the message. Attributes are translated into a cacheable form using the SP's AttributeExtractor, passed through an AttributeFilter, and cached in the new session along with other relevant information.

Once the session is created, the SP determines where to send the browser by examining the "relay state" information returned by the IdP, if any.

Cookie Read by SP

The "relay state" information returned by the IdP, if any, will have been created by the SP and if using a cookie, will point to a specially named cookie that should accompany the authentication response supplied to the ACS endpoint in this step. This is the cookie set in Step 2 above. If this cookie is missing (or if no relay state exists at all), the associated application's `homeURL` property is substituted as a fall back.

Finally, having determined a resource location, the SP will redirect the browser to it.

Cookie Set by SP

The SP will associate the browser with the newly created session by sending a cookie containing a session key to the client as part of the redirection to the resource.

6. Back to the Protected Resource

In the final step, the browser is redirected to the protected resource accessed in Step 1, but this time the access occurs in the context of a session stored within the SP's SessionCache.

Cookie Read by SP

The cookie containing the session key set in the previous step is expected to accompany all subsequent requests for protected resources. This is why it's essential that the ACS endpoint and resource location share a virtual host; if they do not, the session cookie set in the previous step won't be returned by the browser, and looping typically results.

Assuming the session is recognized and validated by the SP, any applicable AccessControl plugins will be enforced, and assuming access is granted, the request will proceed, with any cached attributes attached to the request.