NativeSPEnableApplication

There are many different ways to integrate the Shibboleth SP with an application. Some are very elegant, and some are really not elegant. Some are simple and fast, and others are complicated and will take a long time. The rule of thumb is to deliver the information Shibboleth supplies to the application in the manner the application expects it already, or to modify the application.

Many applications assume they have complete control over the authentication process, including all user interactions and the collecting of credentials. Single sign-on is the art of taking control from the application, and teaching it to rely on authentication (and often attributes) from the surrounding environment. However, sometimes you don't have this degree of control over the application, in which case integrating single sign-on becomes the art of taking attribute and authentication information and placing it where and how the application expects.

Unlike authentication systems that expose an API, the SP operates inside the web server, isolated from applications, and places authentication and attribute information into the web application environment using environment variables, or using custom HTTP request headers in less functional web servers.

You can see much of the information established by Shibboleth for use by an application by accessing that application's "Session" handler: visit, e.g., `https://your-host/Shibboleth.sso/Session` **after** running through the login process.

An application that has been enabled to rely on standard HTTP environment information such as "REMOTE_USER" is essentially already prepared for use with Shibboleth. (We maintain a partial list of common commercial applications that have been enabled already.)

You need to find a way to do two things:

1. **Start a Shibboleth session**. How will a Shibboleth-based session begin? Will it be automatic and mandatory on first access to a protected resource, or will the application dynamically request a session only when it's needed or requested (e.g., using a Login button or link)?
2. **Deliver environment/header variables to the application's authentication and authorization points**. This can be handled in some cases by having Shibboleth or the web server perform the authentication/authorization itself. Please read on.

Session Establishment

Shibboleth can automatically establish a session whenever a particular URL (or URL pattern) is accessed. This means that any user accessing that resource must be able to authenticate at an IdP trusted by the SP. To always require that a session exist, a `ShibRequestSetting requireSession 1` Apache directive is added either to the web server's configuration, or the `requireSession` property is added to the SP's `<RequestMap>`.

Applications can also request that a session be created on demand by redirecting a user to a local URL bound to a `<SessionInitiator>`. This lazy session initiation should be used carefully to avoid unintended access being granted. SWITCH maintains a demonstration site with excellent examples and instructions for use of lazy sessions.

For additional details, refer to the topic on protecting content.

Use of Shibboleth Authentication & Attributes

This is the primary step for integration of data. It's a principle of the Shibboleth SP design that web applications should never require or rely on custom programming specific to any authentication or authorization mechanism, relying instead on variables provisioned by the web environment.

Thus the SP has no API per se; it intercepts requests and sets environment or header variables before passing control to the application. If applications are developed in this fashion, then attribute, authentication, and authorization mechanisms are largely interchangeable with little or no application modification.

When there are touch-points specific to the SP required, such as the need to explicitly request authentication or invoke other advanced functionality directly, the SP exposes handlers as local resources that can be accessed via simple redirects or HTTP callbacks. These are certainly non-portable mechanisms specific to the SP, but they are language-independent and can be isolated to replaceable modules.

Environment / Header Variables

The most straightforward approach is to modify the application itself to use CGI variables such as "REMOTE_USER" or custom mappings. This can be done throughout the application, or up front, in the application's own session establishment step. From that point on, the application could rely on its own session and never interact with the SP again.

See the NativeSPAttributeAccess topic for full details. Example scripts for interacting with the Shibboleth environment are also provided by K.U. Leuven.

Provisioning the Application's Database or Session

Alternatively, an application can retain its existing authentication handling mechanism, but you can place something alongside the application that checks the variables presented by Shibboleth and transforms them into whatever session mechanism the application expects. This means the user's first destination after Shibboleth authentication is this entry script, and then once the script does its thing, the user is sent further on into the application. The application believes the user is authenticated per usual and remains oblivious to the SSO infrastructure. There are two common ways this is done.

1. Many applications check user identities against databases. A popular integration strategy is to create a script that provisions user information from the SP-supplied variables into the application's database. The user may be presented additional tokens to present to the application in the redirect, such as a one-time username/password, but sometimes implicit information like the IP address is used.
2. An application session can be created directly by creating or binding to the session cookie or whatever else the application relies on.

Gateways

A more "minimal" approach in terms of the use of the SP is to utilize a single SP deployed as a gateway in front of an existing or alternate SSO solution. The SP handles the processing of the SAML protocol (or any other SP-supported protocol) and bridges to some other protocol that is deployed with the application.

Some of the background is discussed here.

Access Control

Externally to an application, you can also leverage the SP for static access control. Included with the SP are a pair of plugins, a cross-platform XML-based mechanism and support for Apache .htaccess.