JavaSPRoadmap

This topic does not indicate a commitment to provide a Java implementation of the SP software. It only represents current thinking on the topic.

Java Service Provider

Current Landscape

Currently, individuals wishing to SAML-enable a Java web application with Shibboleth software must front-end their Servlet container with a supported web server (IIS, Apache, Netscape/iPlanet/Sun) and use the C++ SP implementation. In many cases this is fine because deployers often deploy their container behind such a web server for other reasons.

However, in some cases such a deployment environment is not ideal. For such instances it would be nice to have a Java-native implementation of the SP. Such an implementation might integrate with an application via:

- Servlet/Filter - The Servlet 2.x APIs, which includes Filters, are literally the standard for Java web applications. Therefore, this presents a reliable model for dealing with the HTTP request/responses, creating new communication endpoints, locating configuration data, and storing attribute data.
- Java Authentication Service Provider Interface for Containers (JASPI) - The JASPI standard has been around since 2007 but only recently has it begun to be adopted by containers (see next item for why). Because of its slow adoption targeting this API wouldn't provide much bang for the buck at this time.
- Servlet 3.0 - The latest version of the Servlet standard, finalized in December of 2009, now includes API calls to initiate and check the state of authentication. The general idea seems to be that invoking such APIs will trigger a JASPI module. However, because the standard is so new, implementations and deployments aren't there yet.
- Spring Security - Spring has become a defacto standard in Java applications. Spring Security offers a model for integrating various authentication mechanisms in to an application. It has a nascent SAML SP but it's pretty immature and is Spring Security specific. Additionally it can be cumbersome to integrate Spring Security with applications because of various assumptions made by the framework.
- Container/Application Integrated Mechanisms - Most containers provide some sort of implementation-specific mechanism for integrating new authentication mechanisms. Many of these have baked in assumptions about the general flow of an authentication interaction, mostly based of the idea of username/password mechanisms. Application-integrated mechanisms are rarely pluggable and where they are, they often make similar assumptions as the container developers did.

There are existing open source Java-based SAML SP implementations, however they are either poorly supported or are rather limited in what they support. These include:

- OpenSSO from Sun Microsystems, to be discontinued by Oracle
- OIO.SAML from the Danish Government
- Enterprise Sign On Engine (ESOE)

In addition, with commercial applications, there is a question about whether deployers will be willing to integrate a Java SP given that doing so would most likely void their support contracts.

Shibboleth Java SP

Given the discussion above, what might a Shibboleth Java SP look like? The following rough architecture tries to strike a balance between deployment complexity and feature set.

It seems clear that picking a single integration option as the one true option is probably not a good decision. So, instead we would probably try to create a library that was higher level than OpenSAML. Currently the idea would be to have two coarse grained components:

- Authentication Service - This would be responsible for determining if a request needed to be authenticated (similar to the C++ SP's request map functionality), creating the authentication request and processing the response (i.e. implementing the assertion consumer service endpoint logic).
- Attribute Service - This would be responsible for taking a SAML assertion, possibly querying the IdP, and spitting out attributes. This would be similar to the C++ SP's attribute resolver, extractor, and filter services.

In both services the security policies currently performed by the IdP and SP would be included (i.e. SAML condition validation, signature checking, etc.).

In terms of protocols we'd implement the Shib and SAML 2 SSO and SAML 1 and 2 Attribute query profiles. We'd leave out artifact resolution because it seems like it's not widely used and its inclusion would either end up adding a whole lot of complexity or prohibit clustering.

Such an SP library we believe would go 80-90% of the way to a usable SP. It would not address discovery and session initiation because those end up being very tightly coupled to the underlying technology (e.g. the way you'd return a discovery response to a Servlet/Filter setup is quite different than returning it to a Spring Security setup).

To both demonstrate the work necessary to go from the SP library to an SP implementation we would also produce an SP implementation that targeted the Servlet 2.x standard (JASPI and Servlet 3 just aren't widely enough available currently). This would probably also be what a significant portion of Java web apps would want today anyways.

Such a Servlet/Filter based SP would create HTTP sessions (expiration would still be managed by the container), make attributes available via the Session, and optionally HttpServletRequest, attributes. Discovery would be done using the new SP-local discovery service. Session initiation would be implemented in the Servlet.