Shibboleth Identity and Service Provider Training Takeaway

1 Shibboleth Overview

- Shibboleth is a standards-based, open-source single sign-on (SSO) solution that enables people from multiple institutions to access network services shared in a circle of trust known as a federation.
- Shibboleth is an implementation of Security Assertion Markup Language (SAML). SAML gives Shibboleth interoperable SSO capabilities. Using SAML also means individual users in the federation need fewer passwords. And SAML provides better auditability features than traditional SSO solutions.
- Trust among federations is also established via the secure distribution and synchronization of SAML metadata, the information about the federation's Identity Providers (IdPs) and Service Providers (SPs). This metadata is used to scale trust relationships, facilitate SAML transactions to improve authentication, help apps make authorization decisions, and provide accountability and security. Each entity's metadata is keyed by its unique name, called an EntityID.
- Service providers specify what user and/or organizational attributes are required to access their services. And user privacy is protected at all times.
- The InCommon Trusted Access Platform packages Shibboleth IdP and SP as Docker container images to simplify installation and configuration. Containers are the latest way to deploy applications. A container image is your complete application in a package you can deploy and run across multiple environments no matter the infrastructure. You can think of container as highly scriptable virtual machines. They are abstractions at the application layer.

2 IdP Planning

- As an IdP operator, you need to integrate two major components from your infrastructure: an authentication source and an attribute store.
- The most common authentication methods supported natively by Shibboleth IdP are LDAP, Kerberos, and external authentication. LDAP and Kerberos can authenticate against Active Directory. External authentication is possible via an API. Central Authentication Service (CAS) can be leveraged through this API. LDAP is the typical attribute store, although Shibboleth also supports SQL. Shibboleth IdP's attribute resolver transforms LDAP and SQL attributes into SAML attributes.

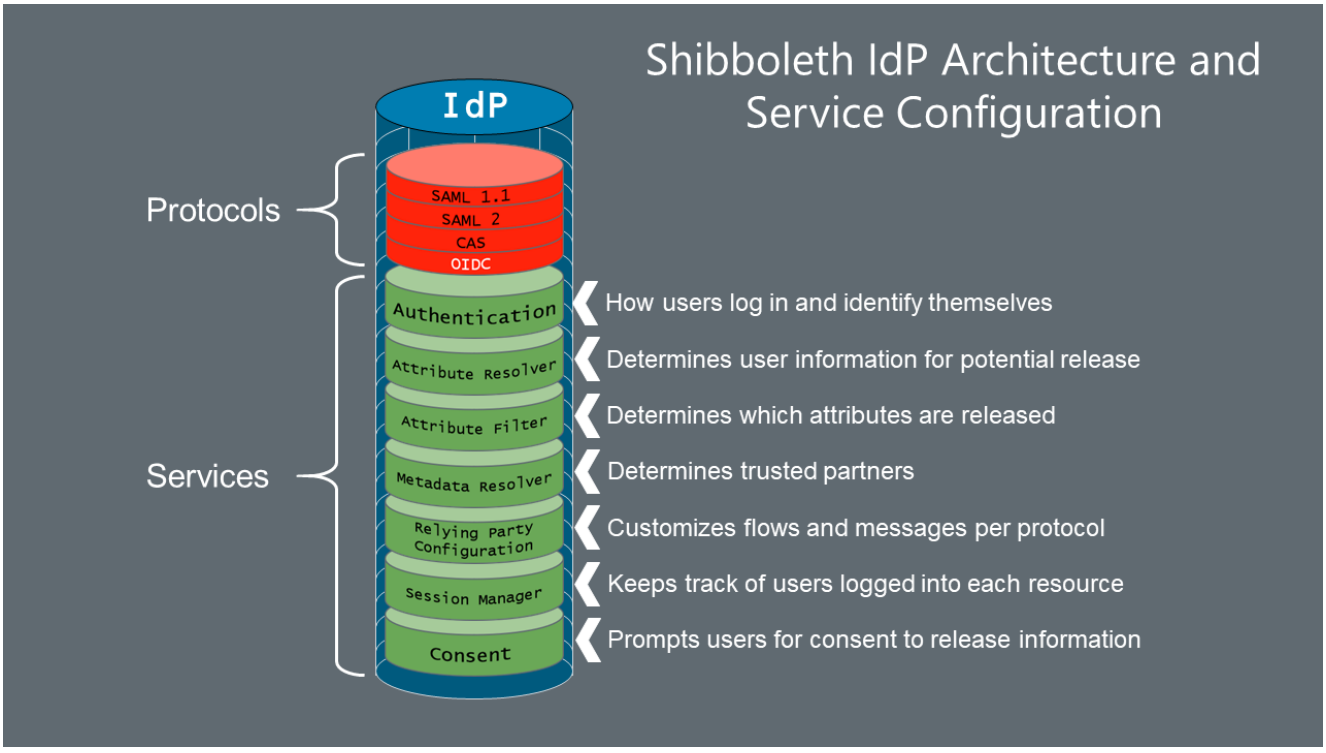

Figure 1 Shibboleth IdP is made up of a protocols layer and a services layer.

3 IdP Configuration

This table shows the Shibboleth configuration files and directories:

File	Purpose
access-control.xml	Use this file to control access to administrative pages such as stats, resolver testing tool, service-reload endpoints, etc.
attribute-filter.xml	Use this file to configure attribute release policies. The Shibboleth IdP has a very flexible rules engine. It allows you to release or block attributes to a single SP, a list of SPs, or all SPs in a federation. This is the file you'll touch most often.
audit.xml	Use this file to control how audit information is written to the audit logs. Edit this file if you want to add or remove information from Shibboleth's audit logs.
authn/	This is a directory in which you can configure authentication styles and control when a particular flow is to be used. Configuring this directory controls authentication behavior, including MFA scripting.
cas-protocol.xml	Use this file to configure features of the IdP's built-in support for Central Authentication Service (CAS) protocol.
credentials.xml	Use this file to configure SAML keys and certificates. You only need to touch this file if you need to perform a key rollover.
errors.xml	Configure how the IdP handles errors. You'll find this useful if users are hitting a condition that results in a very bad experience. For example, you could have a certain event type show a local page with a more helpful error message than simply sending the user on to the service with a SAML error.
global.xml	Empty by default, use this file to override the default behavior of low-level components such as session or storage management.
idp.properties	Use this file to affect a particular global setting before you decide to edit other config files. It's easier to make changes to the IdP's config with this file than it is to edit other .xml files.
intercept/	Use the intercept flows to modify processing flows. Some example flows you'll find in this directory include attribute consent, terms of use, and attribute release.
ldap.properties	Use this file to configure various LDAP authentication and attribute lookup settings.

File	Purpose
logback.xml	Configure all the IdP's logging activity with this file. The default settings work for most situations. You may want to make changes to this file if you need DEBUG logging for the LDAP authentication module or for your authentication events, or if you want to receive an email notification when the IdP logs a message with the level of ERROR.
metadata-providers.xml	Use this file to add SAML metadata sources and to configure how SAML metadata is validated.
relying-party.xml	Use this file to control which SAML profiles will be presented to which services and how the IdP will handle authentication for the various services. Aspects you can control with this file include whether to sign and encrypt assertions, specify the preferred authentication method, and what to use for the SAML NameID.
saml-nameid.xml	This file configures the generation of SAML NameIDs at a finer level of control than the saml-nameids.properties config file.
services.properties	The IdP will automatically reload many of its configuration files which allows you to make changes to the configuration without causing an outage. Use this file to configure services and settings which control the configuration reload policy.
services.xml	Use this file to control how various sub-systems within the IdP load their configuration. For example, you could configure the IdP to pull its relying-party.xml or attribute-resolver/filter.xml files from a central web server or sub-version repository instead of using the local copy in the directory.
session.manager.xml	Use this file to configure how the IdP handles user sessions.

4 IdP Operation

- Observer your IdP's health by looking for a "healthy" status in the `docker ps` output.
- The Linux-based IdP container writes most logging information to `<stdout>` and is available via the `docker logs <containerID>` command or by using docker-compose `docker-compose logs idp` from the same directory as the `docker-compose.yml` file.
- Periodically poll the IdP on its overall health by visiting the `/idp/status` URL.
- To troubleshoot issues, first check the log output by using the `docker logs <containerID>` or `docker-compose logs idp` command. You can also consult the Shibboleth wiki.

5 SP Planning

- The SP supports *active* and so-called *lazy* sessions. Active sessions require an SSO session on all visits. For lazy sessions, the application decides when to initiate a session (by redirecting to a special handler URL).
- Answer the following questions to determine how metadata will be managed:
    - How will IdPs/customers get your metadata?
    - How will you maintain your metadata?
    - From what sources will get trusted metadata?
    - How will changes to metadata be handled?
- Answer the following questions regarding attributes as part of planning your SP deployment:
    - What attributes will you require?
    - How will you use the required attributes?
    - Will you be storing these attributes?
    - How will you communicate your requirements to IdP operators?
    - How will your application handle situations in which some of the required attributes aren't released by the IdP?
- If your SP will have users from more than a single IdP, then your SP will need to handle IdP discovery. If your service works with many IdPs, you can use a discovery service such as the Shibboleth Embedded Discovery Service (EDS), a light-weight and easily-deployed solution. If you're only interacting with a few IdPs, need even more customization than the EDS provides, or simply want an option to bypass user-facing IdP discovery, you can use direct links to the Shibboleth SP's login handler.
- The Shibboleth SP is made up of two components: a web server module that just listens for incoming requests, and a service/daemon that takes a handoff from the web server to do the actual work of processing requests.