SessionInitiator

Summary

In previous versions of the Shibboleth *service provider* (SP), session initiation for applications was handled using a single property, `/SPConfig /Applications/Sessions/@wayfURL` (which could be overridden by applications). A limited lazy session capability also existed (as of Shibboleth 1.2). The overall design led to deployers of services that span multiple Shibboleth federations artificially creating multiple applications in the ShibbolethXml file to enable multiple WAYFs to be used, based on what resources were accessed.

As of Shibboleth 1.3, this notion of initiating sessions has been generalized and expanded to prepare for additional functionality in the future, as well as to simplify the development of more advanced applications and *identity provider discovery* (IdPDiscovery) mechanisms. In particular, a single application can take advantage of multiple *discovery services* (DiscoveryServices; see also WAYF) with the same URL-based approaches but less configuration machinery.

Session initiators (SessionInitiators) are *service provider handler* (ServiceProviderHandler) functions configured using the (duh) `/Applications /Sessions/SessionInitiator` element, of which there can be any number. Each can be assigned a name using the `id` attribute, and the default initiator can be identified with the `isDefault` attribute. They serve two purposes in the running SP:

- Handling required/automatic sessions by intercepting resource requests and returning an **`AuthnRequest`** directed to a particular DiscoveryServices (WAYF).
- Acting as a reception point for requests to initiate a lazy session, which may be directed to a specific Identity Provider (perhaps as a result of some local IdPDiscovery process) or as a default, to a particular DiscoveryServices (WAYF) as in the previous bullet.

If there is no `target` parameter associated with a request, the user will be sent to the URL declared in the `homeURL` attribute of the corresponding `<Application>` element.

Required Sessions

Most deployers rely on the Shibboleth filter to intercept requests for specific (or sometimes all) resources on a given host and ensure that the request is not served until/unless an **authenticated session** is established. This can often be enforced using either the `RequestMap` structure's `requireSession` attribute, or a platform-specific command such as the Apache `ShibRequireSession` command.

Previous versions of Shibboleth had inline code within the filter that redirected the browser to the `Application/Sessions/@wayfURL` location using the Shibboleth *authentication request profile* (AuthnRequest; the `shire/target/providerId` stuff). As of Shibboleth 1.3, this has been generalized for the future, but currently has the same behavior. The default `SessionInitiator` element's `wayfURL` attribute provides the location of the IdPDiscovery service to use and the `wayfBinding` attribute contains a URI that identifies the binding/profile to use for the `AuthnRequest` . Currently, only the Shibboleth profile is supported, with the value `urn:mace:shibboleth:1.0:profiles:AuthnRequest` . Future plugins and versions will support additional `wayfBinding` values.

An additional new feature of Shibboleth 1.3 is the RequestMap's new `requireSessionWith` attribute, and comparable `ShibRequireSessionWith` Apache command. The only difference is that instead of a true/false value, the value of this setting should match the **`id`** attribute of a `SessionInitiator` element in the corresponding application. This effectively allows for multiple WAYF services to be utilized based on which resource is being accessed.

Lazy sessions

For a specific definition of what a *lazy Session* (LazySession) is, refer to that topic. This section addresses the way to establish a session lazily using the session initiator "protocol" supported by Shibboleth 1.3. The presence of a `SessionInitiator/@Binding` attribute containing the URI value `urn:mace:shibboleth:sp:1.3:SessionInit` indicates support for this custom protocol within the Shibboleth SP. While it can be invoked by any entity on the network, since it amounts to a browser redirect, it is not intended as a standard, interoperable mechanism between federated systems, and not all Shibboleth implementations can be assumed to support it. Think of it as an internal (but documented and stable) API.

A SessionInitiator is a ServiceProviderHandler, which means that it exposes SP-specific functionality at a virtual resource location living within the web server's resource tree. As with other handlers, the `SessionInitiator/@Location` attribute is appended to the `Sessions/@handlerURL` value to compute the path at which the endpoint lives. To make use of the session initiator, the browser must send an HTTP GET request to that location with an optional set of query string parameters:

$ target: The resource to return the user to once a session is established (if not specified, the `Application/@homeURL` attribute determines the return location)

$ acsIndex: Matched against a `Sessions/AssertionConsumerService/@index` attribute to select the endpoint to return the SAML response, this is used to determine the value to place in the `shire` parameter when the Shibboleth `AuthnRequest` profile is used. In future versions (i.e. SAML 2.0), it may be placed directly into the AuthnRequest message. If not specified, the default (or first, if no default) *assertion consumer service* (AssertionConsumerService) is used.

$ providerId: The name of an IdP to request a session from. This bypasses the IdPDiscovery process (if for example an application has already carried out that step itself, or if the application has cached the source of an earlier session and simply wants to renew it). If not specified, the `SessionInitiator/@wayfURL` attribute determines the location to which the AuthnRequest is sent.

Use of MetaData

Of special mention is that when a `providerId` parameter is sent with the GET request to a SessionInitiator, SAML MetaData is used to determine how and where to send the resulting AuthnRequest message. The exact options in this step depend on the capabilities present in the Shibboleth version used. In Shibboeth 1.3, unmodified, only entities that support SAML 1.x and the Shibboleth AuthnRequest profile can be "located".

In practice, this means the `providerId` MUST correspond to an `EntityDescriptor/@entityID`, and it MUST contain an `IDPSSODescriptor` with `protocolSupportEnumeration="urn:oasis:names:tc:SAML:1.1:protocol urn:mace:shibboleth:1.0"` or `"urn:oasis:names:tc:SAML:1.0:protocol urn:mace:shibboleth:1.0"`. In that event, the first `SingleSignOnService` element with `Binding="urn:mace:shibboleth:1.0:profiles:AuthnRequest"` is used. A SAML 1.1 endpoint is favored over a SAML 1.0 endpoint.

As the reader can probably surmise, as additional protocols and request profiles/bindings are supported, in plugins or future releases, additional kinds of entities can be supported. Additional LazySession parameters might be defined in the future to influence or bypass precedence rules as things become more complex.