NativeSPSessions

The <Sessions> element broadly speaking controls how the SSO process is managed by the SP. It typically contains a number of child elements called "handlers" that act together to provide the core SAML functionality of the software. It also contains settings that govern how long SSO sessions last and how they're protected from certain kinds of attacks.

Each <ApplicationOverride> inherits the default settings and endpoints defined at the top level, and can contain its own <Sessions> element, if needed, to add additional endpoints. Note that if you supply a <Sessions> element in an override, none of its individual XML attribute properties will be inherited from the default element. You must supply all of them that you want to have non-defaulted values. In contrast, the elements from the parent <Sessions> element do get inherited/used if you don't override them. It's not terribly intuitive, an artifact of the code.

Simplified Protocol Configuration (Version 2.4 and Above)

Version 2.4 introduces a simplified syntax for configuring protocol-related handlers by expressing support in terms of "services" and "protocols". The currently supported services are:

- <SSO>
- <Logout>
- <NameIDMgmt>

A service is enabled by including the corresponding child element(s). Each service element contains a set of properties to configure the service, and the element's value is a list of protocol names that the SP supports. Each supported service element can appear only once, as this is a simplified approach that is meant to capture all of that function's behavior in a single element. More advanced cases may require reverting to older syntaxes, but this isn't usually necessary and you may want to ask on the support list before reverting to that.

These services and names are used to locate detailed information about the handlers to install based on a <ProtocolProvider> plugin. The implementation supplied with the SP relies on a separate configuration file to spell out the detailed information that used to appear within the <Sessions> element.

You can still plugin all of the older information by hand if you need unusual settings or more control over the internals. It's best to replace the new elements entirely to avoid unforeseen overlaps and interactions.

Handlers and Endpoint Construction

Handlers perform the work of the SP apart from the general request filtering work it performs for all requests. While handlers are essentially just mini-applications that could be built as CGI scripts, they're implemented inside the SP to avoid scripting language dependencies and to provide them with full access to the SP library API.

The SP uses a model in which handlers are dispatched based on "path info" that is appended to a base location that acts as a kind of virtual resource that the web server knows about and asks the SP software to implement. Sometimes this is referred to by the web server as a script mapping or handler.

Because the SP's handlers have to know which application is receiving a request, each application has to be assigned a unique "base location", which is called a handlerURL. By convention, this base location is "/Shibboleth.sso".

Often, each application spans a particular virtual host, and the base location is simply "/Shibboleth.sso" on that vhost. In more advanced cases, an application might live inside a subset of a virtual host's document tree. In that case, the base location has to be inside that document tree (e.g., "/path/Shibboleth.sso").

Each handler element has a Location XML attribute that, when appended to the base location, provides the full path to the handler. Combining that path with the virtual host's scheme, hostname, and port, you have the full URL of the handler:

> *http(s):// + hostname + [:port] + handlerURL + Location*

If you require to access the handler URL from your code (for example to trigger a login), the "Shib-Handler" attribute is availble set to the full path above (minus the Location) for a programatic way to access the path set in the configuration.

In turn, these endpoint locations are usually supplied to partner sites in Metadata. When they don't match the metadata, various errors will result.

Attributes

- handlerURL(a relative or absolute URL) (default is "/Shibboleth.sso" on 2.4+, required otherwise)
  - The base location on the server that dispatches requests to the handlers configured inside the <Sessions> element.
- handlerSSL(boolean) (defaults to true)
  - When true, only web requests over SSL/TLS will be processed by handlers. Other requests may be blocked, or possibly ignored (and usually result in a 404 error) depending on the web server, but will never be processed. This is useful for sites that want to protect SAML protocol traffic but leave actual content unencrypted.
- lifetime(time in seconds) (default is 28800)
  - Maximum duration in seconds that a session maintained by the SP will be valid. The actual time may be less than this value (if an IdP indicates it should be shorter) but will never be longer. Note that this will not influence sessions maintained by an application.

- `timeout`(time in seconds) (default is 3600)
  - Maximum inactivity allowed between requests in a session maintained by the SP. This inactivity applies only to requests to this SP and is not aware of activity between the browser and other web sites (or even other applications on this system). A value of 0 disables timeout checking.

- `maxTimeSinceAuthn`(time in seconds)
  - If set, the SP rejects assertions that indicate the user's actual time of authentication to the IdP is older than the difference between the current time and this value. Essentially this limits the amount of time between the act of authentication and the attempt to access the SP. This can be useful to ensure that the SAML 2.0 `ForceAuthn` flag was honored.

- `checkAddress`(boolean) (default is true)
  - The IdP will place the IP address of the user agent it authenticated into the assertions it issues. When true, the SP will check this address against the address of the client presenting an assertion before creating a session. While useful for security, NAT and proxy usage (as well as IPv6 support on only either the webserver hosting the IdP or the SP) often make this setting a source of errors.

- `consistentAddress`(boolean) (default is true)
  - When true, the SP will remember the IP address used when creating a session and ensure that all subsequent access associated with this session come from the same address. This can help protect against cookie theft and is less likely than the `checkAddress` setting to block legitimate access.

- `cookieName`(string)
  - Rarely needed, this can be used to override part of the names used for cookies maintained by the SP. The name is used in combination with other values that are part of the SP implementation and not documented. It is not meant to allow full control over the name but can be useful in some scenarios in which virtual server domains overlap.

- `cookieProps` (string) (default is `"; path=/; HttpOnly"`)
  - If set to a custom string, the string is appended to the cookie values maintained by the SP. Used to attach custom meta-properties like `path` or the `secure` and `HttpOnly` flags to the cookies. A common value for SSL-only use is `"; path=/; secure; HttpOnly"`. As of V2.5, this property can be set to a pair of built-in values, `"http"` and `"https"`, which expand to the default and SSL-only properties respectively.

- `idpHistory`(boolean) (default is false)
  - When true, the SP will maintain a SAML "discovery" cookie in the browser by adding any identity provider successfully used to the cookie.

- `idpHistoryDays`(time in days)
  - Determines how long the history cookie will persist. If not set, the cookie will expire when the browser closes.

- `exportLocation`(an absolute or relative URL)
  - This is the handler location supporting local lookup of raw SAML assertions cached with a session (see NativeSPAssertionExport). When absolute, you can override the usual insertion of the hostname and force the use of "localhost". When relative, it expands much like any handler's `Location` attribute. Note that using an absolute hostname like "localhost" results in an application-specific location that can't be inherited by `ApplicationOverride` elements. You'll need to supply a custom <`Sessions`> element with its own value for this attribute as well as all the other usual settings.

- `exportACL`(space-delimited list of IP addresses) (default is 127.0.0.1)
  - The set of requesters allowed to query the SP for assertions using the mechanism described in the NativeSPAssertionExport topic. This should almost always be left as "127.0.0.1" (or "::1" for IPv6) to prevent third parties from viewing sensitive data about users. No other security mechanism is currently supported. IPv6 addresses must be provided in the non-dotted format with lower case letters and without padding '0's. E.g. "2001:620:0:40:c62c:3ff:abc:123".

Version 2.1 and Above:

- `cookieLifetime` (time in seconds)
  If set, cookies used for session management will be created with the designated lifetime. When omitted, which is the default, such cookies are in-memory only and do not persist across browser restarts (assuming various session restore features aren't in use). Note that this will not affect "transitory" cookies used for maintaining state across redirects.

Version 2.2 and Above:

- `postData`(string)
  - Allows for the preservation of form POST data (with Content-Type application/x-www-form-urlencoded) across SSO on supported platforms and identifies the mechanism for preserving that data. Currently this **MUST** take the form "ss:<id>" where <id> is the ID of a configured StorageService plugin.

- `postTemplate` (path to file)
  - Path to an HTML template with the markup necessary to carry replayed form POST data and if possible prevent accidental replay if the back button is hit. Refer to the default postTemplate.html file as a guide.
- `postLimit` (number of bytes)
  - A maximum number of bytes to allow when saving off POST data. Over this limit, a warning in the log will appear, but the data will not be saved.

- `postExpire` (boolean) (default is true)
  - If set, causes the HTTP response containing to-be-replayed form POST data to be expired and non-cacheable by the server using the usual mix of response headers. Can be adjusted to accommodate varying strategies for back-button/replay prevention.

Version 2.4 and Above:

- `relayState`(string)

- Controls how information associated with requests for authentication, primarily the original resource accessed, is preserved for the completion of the authentication process. If not specified, the resource URL is passed by value to the IdP, when possible. A value of "cookie" causes the URL to be saved in a cookie, to protect the user's privacy. A third option, which is recommended, is to use the SP's persistent storage by specifying a value of the form "ss:id", where id references a `<StorageService>` element, typically "ss:mem". As of V2.5, the "cookie" option can include a ":n" suffix, where n specifies the number of cookies to permit before purging old ones, defaulting to 25.

Version 2.5 and Above:

- `redirectLimit`(see below for allowable values) (default is "none")
  - Prevents the injection of redirect locations after login or logout that don't meet specific criteria, to prevent misusing the SP to carry out phishing attacks.
    - "none"
      - The default, matching previous behavior, does no limiting.
    - "exact"
      - Requires that the destination match the exact scheme, hostname, and port of the handler performing the redirect.
    - "host"
      - Requires that the destination match the hostname of the handler performing the redirect.
    - "whitelist"
      - Requires that the destination's "scheme://hostname[:port]/" matches one of a whitespace-delimited URL prefixes. Allows explicit permission to send the client off-host to external systems. The list of URL prefixes are set via a `redirectWhitelist` setting.
    - "exact+whitelist"
      - Combines the "exact" and "whitelist" policies above, without requiring explicit enumeration of the handler's vhost.
    - "host+whitelist"
      - Combines the "host" and "whitelist" policies above, without requiring explicit enumeration of the handler's host.

- `idpHistoryProps`(string) (defaults to "; path=/;")
  - May be used to override the standard `cookieProps` setting for the SAML "discovery" cookie enabled by the `idpHistory` flag. Primarily this is useful if there's a need for the discovery cookie to be accessible to client side scripts (while leaving the `HttpOnly` property enabled for other cookies).

Child Elements

For Version 2.4 and above, the usual content of this element will be one or more "service" elements, along with an optional set of custom `<Handler>` elements.

The defined service elements, in the order they must appear is:

- `<SSO>` (zero or one) (Version 2.4 and Above)
  - Enables support for one or more Single Sign-On / Authentication protocols.

- `<Logout>` (zero or one) (Version 2.4 and Above)
  - Enables support for one or more Logout protocols.

- `<NameIDMgmt>` (zero or one) (Version 2.4 and Above)
  - Enables support for one or more NameID Management protocols. Rarely used.

Prior to Version 2.4, the child elements defined for this element were the various kinds of handlers supported by the SP. A number of different pre-defined elements are used for handlers with particular significance, often lifted directly from the SAML Metadata schema. A generic element is also used for extension handlers with a more varied nature.

Handlers can appear in any order and need not be grouped into like types, although this is useful for clarity. Some handler types are "indexed", which means they carry labels for referencing elsewhere, and can be marked with an `isDefault` attribute. Order is only significant when determining a "soft" default; that is, if no handler of a particular type is marked as a default, the first one will be used as such.

- `<SessionInitiator>`
  - Initiates sessions by creating an a request for authentication specific to a particular SSO protocol, or invoking some kind of IdP discovery mechanism. Generally superseded in 2.4+ by the `<SSO>` service element.

- `<LogoutInitiator>`
  - Terminates a session by invoking some kind of logout process, which may be local to the SP or global to the SSO environment. Generally superseded in 2.4+ by the `<Logout>` service element.

- `<md:AssertionConsumerService>`
  - Incoming entry point for messages carrying SAML SSO assertions to initiate a user session. The terminology is borrowed from the SAML spec (as is the actual element). Almost always superseded in 2.4+ by the `<SSO>` service element.

- `<md:ArtifactResolutionService>`
  - Incoming SOAP endpoint for the resolution of SAML 2.0 artifacts into protocol messages. This is used when transmitting **outbound messages** as artifacts by the SP, which is not common. (Artifacts issued by an IdP are processed by other endpoints.) The terminology is borrowed from the SAML spec (as is the actual element). Almost always superseded in 2.4+ by the various service elements.

- `<md:SingleLogoutService>`
  - Incoming entry point for single logout protocol messages from an IdP (acting in its role as a session authority). The terminology is borrowed from the SAML spec (as is the actual element). Almost always superseded in 2.4+ by the `<Logout>` service element.

- `<md:ManageNameIDService>`

    - Incoming entry point for NameID management messages from an IdP. The terminology is borrowed from the SAML spec (as is the actual element). Almost always superseded in 2.4+ by the `<NameIDMgmt>` service element.

- `<Handler>`
    - Generic endpoint for non-specific functionality implemented by the SP or an extension library.