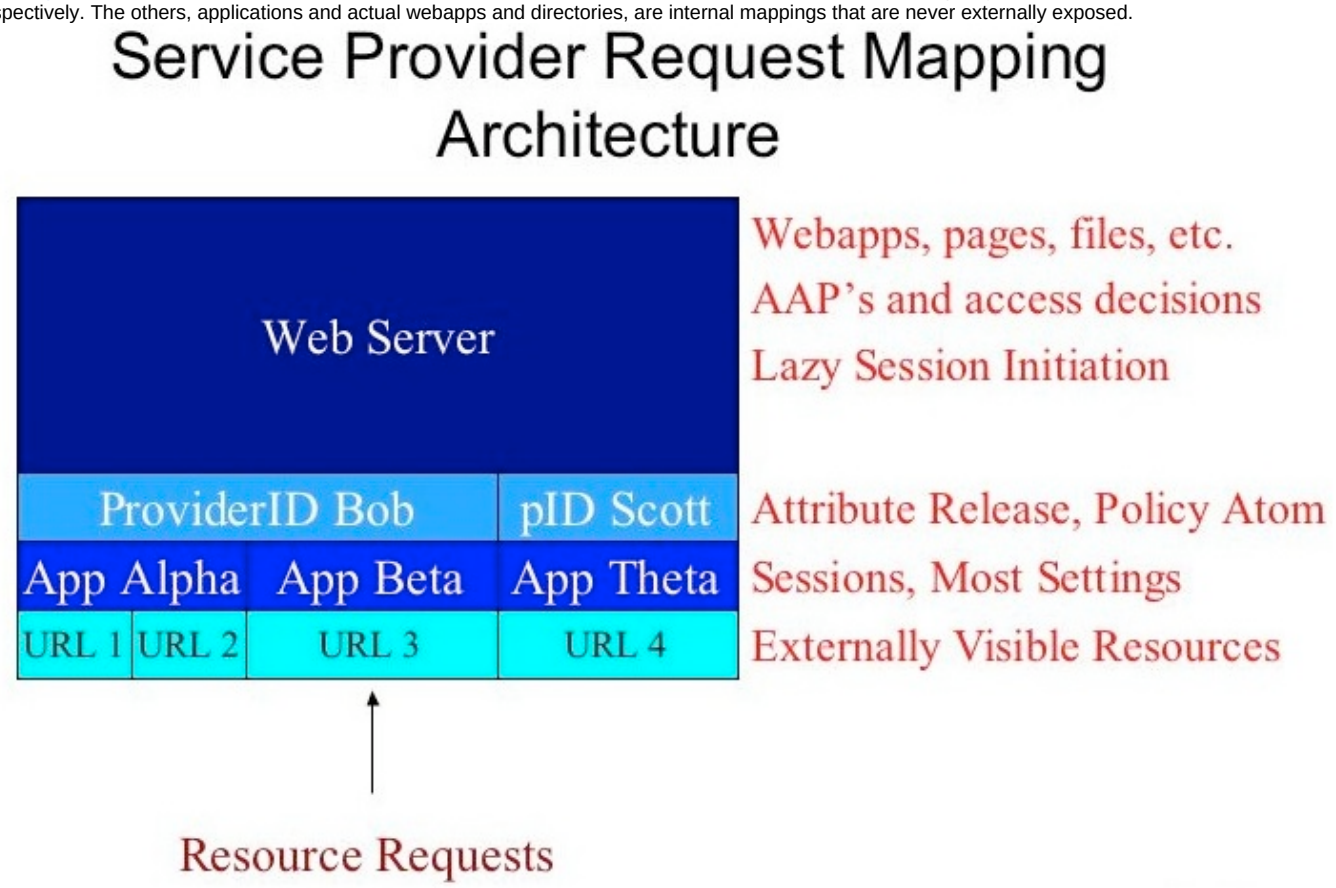
SPProtectionConfig

Request Handling

The Shibboleth SP provides several layers of indirection in its request handling to accommodate a wide variety of deployment scenarios. Each request is passed through this series of evaluations beginning at the requested URL and terminating with a web service, receiving processing tags and information on each hop. Eventually received attributes are matched to a protected resource so that an access control decision can be made and content released.

The following diagram illustrates the layers through which a request passes and the information the query collects from each along the way. Different layers of this chain are visible to relying parties in different ways. The two most important layers for transactions with IdP's are the externally visible URL's and the providerId's associated with these URL's. These control the basic unit of access via web browser and policy and attribute release control, respectively. The others, applications and actual webapps and directories, are internal mappings that are never externally exposed.



To address scenarios where the web application would like to be able to directly interact with the user agent before requiring transport of attributes and the formation of a session, lazy session handling is supported by Shibboleth. All layers between the URL and webserver then become temporarily transparent, but at the application's discretion, sessions and attribute requests may be triggered.
These container elements define the basic behavior of the entire structure.

`<RequestMap applicationId="default" requireSession="true/false" exportAssertion="true/false" />`	The `RequestMap` element must appear only once. It's a container holding `Host` and `Path` elements. Request URLs processed by Shibboleth are parsed and matched against this set of elements in order to determine how to process the request. Attributes on the `RequestMap`, `Host` and `Path` elements specify whether to require an authenticated session and how to locate the associated Application element and settings.

- `requireSession` : This attribute controls whether Shibboleth will forcibly establish an authenticated session with the user before handing off the request to the web server or application. If `true` , Shibboleth will force session establishment. If `false` (the default), web applications are responsible for ensuring that a session exists if necessary, so-called lazy session establishment. Most deployments should not specify `false` for protected content without a full understanding of the implications.
- `exportAssertion` : When `true` , the entire SAML attribute assertion received from the IdP is exported to a CGI request header called `Shib-Attributes` encoded in `base64` . This requires an application to be able to parse the raw XML. Defaults to `false` , which most deployments should use.

`<RequestMapProvider type="edu.internet2.middleware.shibboleth.SP.provider.XMLRequestMap" uri="pathname"/>`	This element must only appear once. It specifies a request mapper that defines how Shibboleth will handle sessions and other behavior for a given request. For the built-in type `edu.internet2.middleware.shibboleth.SP.provider.XMLRequestMap` there must be a `RequestMap` element within this element or the `uri` attribute must contain the local pathname of an XML file containing one.

Hosts and Paths

The tree constructed of the following configuration elements within the `RequestMap` element defines the mapping between the bottom two tiers of the above picture and partially describe the functionality of the bottom URL layer.

`<Host scheme="protocol" name="fqdn" port="integer" applicationId="id" requireSession="true/false" exportAssertion="true/false"/>`	Individual (real or virtual) hosts that this SP protects are enumerated by `Host` elements inside the `RequestMap` element. If a request is processed by Shibboleth for a URL on this host, these parameters will be applied to it. If there are `Path` elements within this element that match the URL and contain the a `pplicationId` , `requireSession` , or `exportAssertion` attributes, they will override values in this element; similarly, values within this element will override those in the containing `RequestMap` element.
`<Path name="pathname" applicationId="id" requireSession="true/false" exportAssertion="true/false" />`	This element allows for different application identifiers and session handling to be defined iteratively for subdirectories or documents within a host. Requests are processed on a best-match basis, with the innermost element taking precedence. Path elements may be contained by `Host` elements or other `Path` elements.

- `applicationId` : The closest match for the URL tree determines the `applicationId` to be used, with the innermost element always taking precedent and overriding outer elements. This is used to divide the webspace by application. The main `RequestMap` contains a fixed value of "default" to reference the default `Applications` element.
- `scheme` : This specifies the protocol on which this host responds. Valid choices are `http` , `https` , `ftp` , `ldap` , and `ldaps` . If omitted, both `http` and `https` are in effect.
- `name` : This is the fully-qualified domain name of the host. This appended to the `scheme` must match what is contained in the URL for the element's settings to apply to the request.
- `port` : This is the port the host is listening on, if not the standard port for the scheme.
- `requireSession` : This attribute controls whether Shibboleth will forcibly establish an authenticated session with the user before handing off the request to the web server or application. If `true` , Shibboleth will force session establishment. If `false` (the default), applications are responsible for ensuring that a session exists if necessary, so-called lazy session establishment. Most deployments should not specify `false` for protected content without a full understanding of the implications.
- `exportAssertion` : When `true` , the entire SAML attribute assertion received from the IdP is exported to a CGI request header called `Shib-Attributes` , encoded in `base64` . This requires an application to be able to parse the raw XML. Defaults to `false` , which most deployments should use.

ISAPI Configuration

If ISAPI is the webserver used, an additional mapping structure must be defined.

`<Implementation />`	A container element placed inside the `Global` element, the contents of this element will vary depending on the web server or environment that this Shibboleth deployment serves. Multiple configurations may be specified, but only one per implementation type. Currently, this element may contain the `ISAPI` element.

<ISAPI normalizeRequest="*true/false*" />	The configuration information for Shibboleth SPs deployed on Microsoft IIS is stored inside this container element. This element must contain one or more Site elements, each of which maps an INSTANCE ID value to a default hostname. If normalizeRequest is true (the default), all redirects and computed request URLs generated by Shibboleth will be created using the hostname assigned to the site instance handling the request. If false , the browser's supplied URL is sometimes used to compute the information, which can cause errors particularly with protocol handlers. Placed inside the Implementation element.
<Site id="*INSTANCE_ID*" name="*fqdn*" scheme="*http/https*" port="*integer*" />	This element is placed in the ISAPI element to specify a mapping from individual instance ID's to a corresponding hostname. The port and scheme can also be specified, but should normally be left out, enabling them to be determined from the browser request. While IIS permits multiple hostnames to be assigned to a web site, only one can be specified here. If you really need to allow for multiple names (unusual), you should set the normalizeRequest attribute to false.

Applications

The information in this section of the configuration affects Shibboleth behavior at the application layer, above URL's, and also controls the mapping of this layer onto the providerId layer above it. The application-level settings control most SAML-related configuration, session management, and define protocol handlers. URL's map directly to these, and these map further to specific protocol handlers.

<Applications id="*default* providerId="*identifier*" signRequest="*true/false*" signedResponse="*true/false*" signedAssertions="*true/false*">	The Applications element must appear once and contains default settings for requests handled by the SP. It must contain at least one each of the Sessions , Errors elements, and may contain CredentialUse , saml:AttributeDesignator , saml:Audience for backward compatibility, FederationProvider , TrustProvider , RevocationProvider , and Application elements.

- id : This attribute has a fixed value of "default" and should not be changed.
- providerId : Distinct from the internal identifier, the providerId is the unique identifier by which this provider is known to the world. This value is referenced by IdP's when creating rules for the release of attributes to SP and will often be provided to federations to facilitate IdP configuration.
- signRequest : If true , the SP will sign attribute requests that it sends to IdPs by default. This is usually unnecessary, as the TLS/SSL transport can provide authentication more efficiently.
- signedResponse : If true , the SP will require that all SAML attribute responses it receives are signed by default.
- signedAssertions : If true , the SP will require that individual SAML assertions it receives are signed by default. This may be particularly useful if the application is forwarding the assertion, but requires a liberal (or no) AAP to avoid corrupting the signature.

The settings defined in the <Applications> element can be overridden by using the various elements within the RequestMap to assign a non-default applicationId to particular content in Host and Path elements. An Application element is then inserted containing a matching id attribute, and finally specific elements that override the defaults are placed within it. A fully specified Sessions element is always required for any new application created, because each application needs a distinct handlerURL so that new sessions can be unambiguously mapped to a particular application.

<Application id="*identifier*" providerId="*identifier*" signRequest="*true/false*" signedResponse="*true/false*" signedAssertions="*true/false*">	Individual applications that require different attributes, session settings, metadata, etc. can be differentiated from the default configuration as specified in the Applications element. It must contain a Sessions element, but overriding other elements is optional.

- id : This attribute defines an internal identifier allowing individual applicationId attributes as part of Host and Path elements to point to this Application to handle requests.
- providerId : Distinct from the internal identifier, this is the unique identifier that will be used when communicating with IdP sites to request authentication or attributes. This value is referenced by IdP's when creating rules for the release of attributes to SP's and will often be provided to federations to facilitate !IdP configuration. If none is specified, the default Applications element's providerId applies.
- signRequest : If true , the SP will sign attribute requests that it sends to IdP's on behalf of this application. This is usually unnecessary, as the TLS/SSL transport can provide authentication.
- signedResponse : If true , the SP will require that all SAML attribute responses it receives for this application be signed.
- signedAssertions : If true , the SP will require that individual SAML assertions it receives for this application be signed. This may be particularly useful if the application is forwarding the assertion, but requires a liberal (or no) AAP to avoid corrupting the signature.

<Sessions wayfURL="*URL*" handlerURL="*URL*" handlerSSL="*true/false*" lifetime="*seconds*" timeout="*seconds*" checkAddress="*true/false*" cookieName="*URL*" cookieProps="*URL*"/>	Configuration parameters that affect the way Shibboleth handles sessions for an individual application are bundled in this element, which must be included in each `Application` and the default `Applications` element. Note that these parameters only apply to Shibboleth sessions, and not any sessions applications manage on their own behalf.

- `wayfURL` : The URL of the WAYF service responsible for redirecting users accessing this application to their IdP. For applications with a single user community, this can point directly to the IdP's SSO protocol handler.
- `handlerURL` : Specifies the URL for the protocol handler/assertion consumer service, at which new sessions are initiated or lazy sessions are triggered. This can be an absolute URL, or a relative path to be prefixed by the base URL of the virtual host. Using an absolute URL allows a virtual server to funnel requests to a fixed location, to force use of SSL, for example. This URL issues the session cookie set on behalf of the application, and this cookie must be returned in subsequent requests, so the virtual host's domain name and port must be consistent with this domain name and port for some browsers to properly return the cookie. If default ports are used (and thus left unspecified), browsers will generally return cookies set via SSL to a non-SSL port. If non-default ports are used, it is recommended that this be a relative URL so that each virtual host handles its own cookie operations. For Shibboleth to function properly in IIS, the file extension at the end of this URL must match the value configured into IIS and mapped to the ISAPI extension. This causes the request to be serviced properly, even though no file by that name actually exists.
- `handlerSSL` : If `true` , requests for the session initiators associated with this application will force sessions over SSL. This is should be set `true` when applications are hosted over `http://` , and is also useful in situations where the web server is providing improper schema information to Shibboleth.
- `cookieName` : Optionally specifies the name given to in-memory session cookies that are associated with this application. If omitted, Shibboleth will generate a cookie name for you of the form *`shibsession`*`<Application ID>` .
- `cookieProps` : A string of additional Set-Cookie properties can be specified using this element which give the browser further instructions about cookie processing and use. This is mostly useful to force secure cookies and constrain their paths. Always begin with a semicolon to delineate from the session ID value.
- `lifetime` : Duration in seconds of the Shibboleth session; this does not affect the lifetime of application sessions initiated independently of Shibboleth. Defaults to 3600. If 0 is specified, sessions are infinite, subject to purging by the cache.
- `timeout` : If the value in seconds elapses following the last request in a session, the session will be expired for inactivity and a new session must be initiated upon the next request. Defaults to `1800` . If `0` is specified, there is no inactivity timeout.
- `checkAddress` : If `true` (the default), Shibboleth will check the browser's client address to ensure that session cookies are issued and used by a consistent client address. In most circumstances, this should be enabled to help prevent attacks using stolen cookies, but this can cause problems for users behind proxies or NAT devices.

Protecting Content

Protection of web pages is primarily achieved through mapping attributes provided by an !IdP to a localized vocabulary for authorization rules. This is defined in the AAP.xml file. This applies to both Apache and IIS. Both webservers also support the XML based authorization rules in ShibbolethXml.

IIS:

The IIS filter module supports the mapping of attributes into HTTP headers via AAP files. Rule-based access control can be configured using the `Request Map` in ShibbolethXml. In addition, all of the configuration settings, such as control over whether to prompt for new sessions automatically, are managed via the `RequestMap` element, so there are no additional commands to document at this time.

Apache:

The Apache module provided can also interpret AAP settings to map attributes to HTTP request headers and to `Require` rules, permitting protection of both static and dynamic content. Any of the typical ways of protecting content may be used (`.htaccess` , or `httpd.conf` blocks including `<Directory>` , `<Location>` , `<Files>` , etc.). They define what content is to be protected and access control rules to apply against it.

There are two ways to require Shibboleth authentication, but both also require enabling the module to activate by specifying an `AuthType` of `shibboleth` and supplying at least one `Require` rule in `httpd.conf` or `.htaccess` files. This is an Apache requirement. Without `AuthType` and `Require` in effect for a request, the Shibboleth filter will not be invoked by Apache and the request just passes through.

The `Require` rule can enforce a specific access control policy based on attributes, can specify `valid-user` to require any authenticated session, or you can use a placeholder rule name of `Shibboleth` . This can be used when the actual protection rules are placed in the RequestMap, or to support LazySessions. In such cases, the module is activated, but in a passive mode that does not automatically force a session, but will process and validate a session if one exists, leaving the authorization decision to the application. Using a static access control rule that will fail in the absence of a session is only sensible if one of the two approaches below that force a session are used.

To require a session, either the Apache command, `ShibRequireSession On` , or the `requireSession` boolean XML attribute on the closest `Request Map` , `Host` , or `Path` elements in ShibbolethXml can be used. Both approaches are equivalent, and using either one to require a session will supersede a false or absent setting of the other type. If your requests are still passing through, you may have forgotten to apply the Apache settings above.

As an example, the following commands will simply require Shibboleth authentication for a resource:

```
AuthType shibboleth
         ShibRequireSession On
         Require valid-user
```

When you want to defer all decisions to the RequestMap, you can apply the filter passively to all content like so:

```
<Location />
         AuthType shibboleth
         Require shibboleth
</Location>
```

A complete list of Shibboleth Apache directives and their values is below:

`AuthType <string>`	Use `shibboleth` for direct invocation, or `Basic` plus the `ShibBasicHijack` option described below.
`ShibBasicHijack <on/off>`	Controls whether Shibboleth should or should not ignore requests with `AuthType Basic` . Defaults to `off`.
`AuthGroupFile <pathname>`	Same as `mod_auth` ; collects values found in `REMOTE_USER` into a named group for access control. An attribute must be mapped to `REMOTE_USER` for this to work (defaults to `eduPersonPrincipalName`). `mod_auth` will not support group files when the Shibboleth module is loaded, since they share the same command.
ShibRequireSession <on/off>	Controls whether to require an authenticated session before passing control to the authorization phase or the actual resource. Defaults to `off` .
`ShibRequireSessionWith <SessionInitiator_id>`	Initiate a session using a specific SessionInitiator if no session exists.
`ShibApplicationId <ApplicationId>`	Set ApplicationId for this content.
`ShibRequireAll <on/off>`	Controls whether all `Require` rules specified must be satisfied before access to the resource is granted. Defaults to `off` , which means any single rule can be satisfied, the usual Apache behavior.
`ShibDisable <on/off>`	Disable all Shibboleth module activity here to save processing effort. Defaults to `off` . note: if a require statement is set, there needs to be another AuthType defined to handle it.
`ShibRedirectToSSL <portnumber>`	When this directive is set all non-ssl http GET or HEAD request are automatically redirected to https before processing by Shibboleth. Other methods (like POST) are presented an error page to block access. This is highly recommended in combination with setting your cookies to secure as it will prevent looping and enhance security. Of course you'll set it as `ShibRedirectToSSL 443` most of the time. (available from v1.3d)
`ShibURLScheme <http/https>`	Used in advanced virtual hosting environments which need to generate SSL redirects from virtual servers that use only HTTP (eg when offloading SSL to another machine). With this directive you can force Shibboleth to send out a redirect to the HANDLER (eg Shibboleth.sso) on https, even though it runs on http when looking at the local machine. Supplements the Apache `ServerName` and `Port` commands with this missing option. Defaults to a null value in which the scheme for redirects is based on the physical connection to the server. This is a server-level command, while the rest of the commands listed are content commands that can appear anywhere.
`ShibExportAssertion <on/off>`	Controls whether the SAML attribute assertion provided by the AA is exported in a base64-encoded HTTP header, `HTTP_SHIB_ATTRIBUTES` . Defaults to off .

There are many ways that authorization can be performed once Shibboleth has successfully transported attributes; any module that performs a standard call can be used. The below example uses the standard pre-installed mod_auth by placing a `.htaccess` file that references a group file stored at `/pathname` :

```
AuthGroupFile /pathname
         require group workgroup
```

An
AuthGroupFile used by Shibboleth might resemble:

```
workgroup: joe@example.edu, jane@demo.edu, jim@sample.edu
```

`Require <string>`	Enforce authorization using one of the following methods:

- `valid-user` : Any Shibboleth user from a trusted !IdP site is accepted, even if no actual attributes are received. This is a very minimal kind of policy, but is useful for testing or for deferring real policy to an application.
- `user` : A space-delimited list of values compared against `REMOTE_USER` , which is populated by default from `urn:mace:dir:attribute-def:eduPersonPrincipalName` . Any attribute can be mapped to `REMOTE_USER` as defined by `AAP.xml` .
- `group` : A space-delimited list of group names defined within `AuthGroupFile` files, again provided that a mapping to `REMOTE_USER` exists.
- *`alias`* : An arbitrary rule name that matches an `Alias` defined in an `AAP.xml` . The rule value is a space-delimited list of attribute values, whose format depends on the attribute in question (e.g. an affiliation rule might look like:

  ```
  require affiliation staff@osu.edu faculty@mit.edu=
  ```

- `shibboleth` : If a session cookie of the expected name exists, the corresponding session will be validated and any cached attributes exported as otherwise specified. Authorization will be controlled by the resource, unless additional rules are specified. If however a session does not already exist, or if the current session expires or times out, no session will be requested and control will pass to the resource. This is known as "lazy session" (see below).

For `user` and *`alias`* -based rules, if a tilde character (~) is placed immediately following `user` or *`alias`* , the expressions that follow are treated as regular expressions. The syntax supported is generally based on the one defined by XML Schema. This specification borders on unreadable, but the syntax is generally Perl-like. Expressions should generally be "anchored" with the ^ and $ symbols to ensure mid-string matches don't cause false positives.

For example, the rule:

```
require affiliation ~ ^member@.+\.edu$
```

would evaluate to allowing anyone with an `eduPersonAffiliation` of `member` from a `.edu` domain.

If the regular expression to be used contains a space (e.g. ^[^].*$ to enforce a non-space character at the start of the attribute value), then the expression as a whole needs to be surrounded by double quotes in order to be interpreted correctly:

```
require affiliation ~ "^[^ ].*"
```

Using Attributes and Session Data in Applications

Apart from the simple RM functionality provided, attribute information may be made available directly to applications via the standard practice of creating custom HTTP request headers before passing control to the resource. Web applications should make no assumption about the presence of specific attributes for their use unless they have intimate knowledge of the attribute release policies in place.

The `AAP.xml` rules control this interface, and map Shibboleth attributes to header names, such as mapping `urn:mace:dir:attribute-def:eduPersonAffiliation` to `Shib-EP-Affiliation` . Using that example, any values of the mapped attribute will be placed in that header, delimited by semicolons. An application that uses a CGI-like syntax to access the header will find the values in the `HTTP_SHIB_EP_AFFILIATION` variable. Any attribute can be placed in any header, to drive legacy applications that expect information in a particular location.

The `REMOTE_USER` variable is a special case that is generally populated automatically by the web server based on internal data that represents the current `username` . Unlike many authentication modules, Shibboleth does not guarantee that `REMOTE_USER` will have any value, because users may remain anonymous in many cases. If it does have a value, it is set solely because of an AAP file that maps an attribute to that header name. As a default and in most scenarios, the `urn:mace:dir:attribute-def:eduPersonPrincipalName` attribute should be mapped to `REMOTE_USER` . Even so, EPPN may not be provided by the AA, and `REMOTE_USER` might still be empty.

In addition to general attribute information, the following special
HTTP headers are created for any authenticated request:

`HTTP_SHIB_IDENTITY_PROVIDER`	Contains the unique identifier (=providerId=) of the !IdP site of the user. Some applications may use this to lookup additional policy or application data. It normally takes the form of a URI but could be any string in some deployments.
`HTTP_SHIB-AUTHENTICATION-METHOD`	Contains the SAML `AuthenticationMethod` URI that documents some aspect of the user's authentication to the !IdP's web authentication service.

HTTP_SHIB_APPLICATION_ID	Contains the XML applicationId attribute in shibboleth.xml that corresponds to the request based on the RequestMap and associated elements.
HTTP_SHIB_ATTRIBUTES	Contains the assertion in XML containing the SAML attribute information from the AA in base64-encoded format. This is a raw interface that provides an application with the entire assertion in, but is still a filtered view based on any attribute acceptance rules.

Finally, special support exists to obtain the value of the SAML <NameIdentifier> element, which identifies the subject of the session, the user. Most Shibboleth deployments use opaque handles that have no application value, however, some deployments may choose to use alternative identifiers with more meaning. SP's can obtain the primary subject name directly from the assertion by using a special AAP <AttributeRule> with a Name corresponding to the SAML Format identifier that describes the kind of identifier used to represent the subject. The rule specifies in what header to export the identifier value (such as REMOTE_USER=), while the =Format identifier will be placed in the HTTP_SHIB_NAMEIDENTIFIER_FORMAT header.

Lazy Sessions

This section describes how an application can trigger the establishment of a Shibboleth session and optionally receive attributes once its internal logic decides this is necessary. Shibboleth decides the application desires lazy sessions when the RequireSession attribute of the Path or Host element protecting it is set to false (or Apache directives ShibRequireSession Off -the default- and require Shibboleth are set). This application must be aware of three pieces of information:

1. **handlerURL** attribute of a Sessions element: The URL of the handler associated with this Application .
2. **location** attribute of a SessionInitiator element: The session initiator you want to use.
3. **target**: The application URL that should be accessed after the session is established; usually, this will be the application's own URL.

These three pieces of information must be combined by the application to an appropriately formed URL to trigger session initiation as follows. To request a session, the application returns an HTTP redirect that sends the browser to the handler URL with a parameter, target , containing the URL of the resource to return to with a session. This will often be the URL that's triggering the redirect. You can also make this available as a standard "login" link on a webpage. The Shibboleth module will generate the redirect to the WAYF (or directly to the !IdP, defined by the SessionInitiator) and the rest proceeds as a standard Shibboleth flow. This combined URL takes the form:

```
https://"Sessions_handlerURL" "SessionInitiator_location" "target=applicationURL"
```

For example, if an application located at https://sp.example.org/portal presents a page with an option to login, it could respond to the login button by redirecting the browser to:

```
https://"sp.example.org/Shibboleth.sso" "/WAYF/idp.example.org" "?target=https%3A%2F%2Fsp.example.org%2Fportal"
```

Security Note

When using "lazy sessions", the Shibboleth module will automatically make the principal's attributes available to the webapplication when a session is established. If no session is established, the module will clear the possible attribute-HTTP-headers from the request. This prevents header spoofing.
If you are using these HTTP headers in your webapplication (highly likely), make sure the request always has to pass through the Shibboleth module. If it is possible to bypass the Shibboleth module, somebody might be able to manipulate your webapp severely by spoofing a header! (note that this problem isn't only subject to "lazy sessions", but you probably won't notice that problem easily when using the passive "lazy sessions" mode)