Apache

Apache 2.4 Support

You should review this page and the htaccess page thoroughly because Apache 2.4 is much more complicated than earlier versions. In particular, if you're trying to combine Shibboleth with other authentication schemes (like Basic), you may need to enable the `ShibCompatValidUser` option, documented below.

Half of Shibboleth runs within the web server. For Apache, this half is implemented in a module called *mod_shib_13.so, mod_shib_20.so, mod_shib_22.so,* or *mod_shib_24.so* depending on the Apache version. Like all Apache modules, the initial configuration is controlled with Apache's configuration file(s), but one of the primary options there (normally implicit/defaulted) is to point the module at the overall SP configuration file *(shibboleth2.xml)* where a lot of the options not specific to Apache are controlled.

At runtime, the module has the ability to process both a variety of Apache commands and rules specified in the SP configuration and make sense of both. This allows for a choice of approaches based on the need for native integration with Apache or for portability between web servers. Native integration using Apache commands is the better choice and is more secure.

Prepping Apache

It's critical that you correctly configure the virtual hosts you will be using with the SP module by setting the `ServerName` command to the appropriate value. With Apache 2.x, you use this command to establish the proper hostname as well as the logical scheme and port the virtual host appears to run on from the client's point of view. If you fail to perform this step, the redirects generated by the module will be incorrect and various problems will ensue. Other related commands (varying by version) include `UseCanonicalName` and `UseCanonicalPhysicalPort`. Before you do anything with the SP, do the work to get these commands working for you to enable proper generation of redirects.

You do not *have* to set `UseCanonicalName On`, but you usually should for two reasons:

First, it's usually necessary to ensure that the redirects generated by the module are not affected by the client's choice of name (e.g., via use of /etc/hosts to map a custom hostname to the server). Failure to do this will often result in requests to the IdP with an unregistered response location that will be rejected there. There are ways around this but they're beyond the scope of this topic and depend on the IdP's cooperation.

Second, it's *necessary* to enable this option if you plan to use the `<RequestMapper>` feature in the SP configuration. Failure to do so will render your system vulnerable to trivial attacks. If for some reason you don't want to turn the option on, do **NOT** use the `<RequestMapper>` feature to determine how to protect content. Generally you don't have to use that feature with Apache and the coupling to the canonical name option is the main reason it's not recommended.

Finally, on non-Windows systems you should make sure Apache is configured in so-called "worker" mode, using the "worker" MPM, either via a setting in an OS-supplied file like */etc/sysconfig/httpd* or in the Apache configuration directly. Many servers come incorrectly configured in "prefork" mode, which emulates Apache 1.3's process model and causes vastly greater resource usage inside the shibd daemon.

If you think, probably incorrectly, that you can't do this, then you are going to experience huge spikes in memory usage under load. To mitigate this, you can create a `stackSize` property in the appropriate `<Listener>` element to shrink the default stack size for daemon threads to a manageable size. You may need to create that XML element, as it is normally defaulted. You can also control stack size through operating system options such as the *ulimit* com mand.

Loading the Module

Most of the primary options needed to make the module usable are compiled into the code when you build it. This allows the module to run silently in most cases simply by loading it (the exact path will vary):

Apache 2.2 Example

```
LoadModule mod_shib /opt/shibboleth-sp/lib/shibboleth/mod_shib_22.so
```

You're free to load the module anywhere in your Apache configuration layout that you prefer. The software includes a number of template files, one for each Apache version, that include the command above, a useful `Alias` setup that allows the default error templates to access an example style sheet, and a simple `<Location>` example that protects a single directory.

It's best if you put your own custom configuration rules in Apache configuration files that you control rather than relying on the template files supplied with the software, because those files are overwritten during updates.

Properly Routing Handler URLs

For certain Apache configurations, such as using the wsgi_module to run Python applications (like django), the URL paths that Shibboleth handlers rely on (e.g. /Shibboleth.sso, see here) may not get properly routed to mod_shib by Apache, because another module has already claimed that namespace. To ensure this routing, you will need to set a Location directive within Apache's configuration file specifying routing to mod_shib. For the default path used by the handler (`/Shibboleth.sso`), this directive is:

Apache SetHandler

```
<Location /Shibboleth.sso>
  SetHandler shib
</Location>
```

You would have to vary the location in this directive if you change the `handlerURL` attribute of the `<Sessions>` setting in *shibboleth2.xml*, but this isn't common.

Global Options

There are a handful of global options that apply to the module's overall configuration and are usually left out in favor of the values generated at compile time. They also correspond to a number of environment variables that can be used in place of commands. They are generally needed only when the software is run out of a different directory from the build path.

`ShibPrefix`	Corresponds to SHIBSP_PREFIX variable (typically only usable on Windows)
`ShibConfig`	Corresponds to SHIBSP_CONFIG variable
`ShibSchemas`	Corresponds to SHIBSP_SCHEMAS variable

Server / VirtualHost Options

`ShibURLScheme`	Apache 1.3 only: Controls the URL scheme Apache will report to modules, should reflect the logical value seen by clients from outside your network.
`ShibCompatValidUser On|Off`	Default is Off, matching the behavior prior to this command's existence. Addresses a conflict when using Shibboleth in conjunction with other auth/auth modules by restoring "standard" Apache behavior when processing the "valid-user" and "user" Require rules. See the ht access topic for more detail.

AuthConfig Options

The rest of the options supported by the module are what Apache calls "AuthConfig" options. This means they are meant to appear inside Apache content-control sections like `<Directory>`, `<File>`, or `<Location>`, or in `.htaccess` files (if the "AuthConfig" override is enabled).

Note that Apache has complex (and unclear at times) rules for combining settings across all those different sections. The SP itself doesn't impact these rules; Apache is doing all the merging and evaluation and the SP just performs based on the resulting merge. One particular quirk seems to be that mod_rewrite can cause problems with anything but `<Location>`-based settings. It seems to ignore settings applied based on physical path when it creates subrequests, so the original request seems to have the intended settings, but the rewritten request doesn't.

A summary of the various commands follows:

`AuthType` *type*	Partially activates the module when *type* is set to "shibboleth" (or "basic", see `ShibBasicHijack` below). Must be accompanied by a `Require` command. See http://httpd.apache.org/docs/2.2/mod/core.html#authtype (or equivalent for your Apache version).
`Require` *rule operands*...	Enables an authorization rule in the module. For complete details, see the sections below on authentication and authorization. Also see http://httpd.apache.org/docs/2.2/mod/core.html#require (or equivalent for your Apache version). **Apache 2.4** Note that the supported rules have changed somewhat for Apache 2.4 because of its API differences. Direct re-use of your configuration probably won't work if you move to Apache 2.4.

`ShibRequestSetting` *setting value*	Allows any valid content setting to be set or altered for the applicable request(s). This command takes two parameters, the name of the content setting, and the value to set it to. For boolean/flag options, you can use the exact values "1", "true", and "On" or the exact values "0", "false", and "Off". For complete details, see the section below on content settings
`ShibRequestUnset` *setting*	Allows a content setting to be reverted to its default value at a particular point in the merging process that Apache carries out. There aren't a lot of cases where this has value, but in a few edge cases like `requireSessionWith` it can be useful.
`ShibDisable On|Off`	When enabled, this allows the module to short-circuit and ignore requests much faster than without the option set. This is useful for bypassing processing for high-traffic, public content.
`ShibBasicHijack On|Off`	Allows for compatibility with extensive legacy **mod_auth** configurations by activating the module when `AuthType` is set to `basic`
`AuthGroupFile` *filename*	Identifies a **mod_auth**-style file containing group membership information for simple access control needs. See http://httpd.apache.org/docs/2.2/mod/mod_authz_groupfile.html#authgroupfile. On Apache 2.4, this command is no longer handled directly by the SP module.
`ShibRequireAll On|Off`	Normally, `Require` rules are processed such that satisfying any one rule will grant access. Subject to certain constraints (see the htaccess topic), turning this option on will change the behavior such that all rules must be satisfied. ⚠ **Not supported on Apache 2.4+** This command is subsumed by Apache 2.4's own support for controlling authorization rule composition.
`AuthzShibAuthoritative On|Off`	As explained in the htaccess topic, this option controls the behavior of the module when it encounters `Require` rules it does not understand and `ShibRequireAll` is enabled. Defaults to "On". ⚠ **Not supported on Apache 2.4+** This command is subsumed by Apache 2.4's own support for controlling authorization rule composition and dispatching of rules to specific modules.
`ShibUseEnvironment On|Off`	Defaults to "On", this turns on the use of environment variables to publish attributes to applications. This is strongly preferred over the header option.
`ShibUseHeaders On|Off`	Defaults to "Off", this turns on the use of request headers to publish attributes to applications. Use of this option should be avoided. Be sure to review the topic on spoof checking if you enable it.
`ShibAccessControl` *path to an authentication plug-in configuration file*	Enables the use of XML Access Control rules for access control. This option can also be used in an .htaccess file. This allows non-root users to set complex access control rules without a restart of the web server. The plug-in is loaded on every request, which allows on-the-fly changes of access control rules (though is less efficient if large rulesets are used). ⚠ **Apache 2.4** With Apache 2.4+ the `ShibAccessControl` command is not supported anymore. Instead use `require shib-plugin path` as is described in htaccess
`ShibExpireRedirects On|Off`	Defaults to "On". Addresses issues with some browsers, notably Firefox 5+, that cause redirects generated by the SP to be cached, resulting in various errors following the login process. This usually manifests as a message replay error at the IdP, caused by the original redirect to the IdP being replayed. This option is enabled by default, but the older behavior can be restored, causing the cache-related headers on redirects to be governed by standard Apache settings.
`ShibCompatWith24 On|Off`	This option can be enabled to up-level the syntax requirements for the `Require` rules supported by the SP into the form used with Apache 2.4. You can enable this option to help migrate rules into a form that will work on Apache 2.4 before actually upgrading to it. This minimizes the compatibility issues for an upgrade.
`ShibRequestMapperAuthz On|Off`	Defaults to "On". Controls whether or not access control plugins attached using the `<RequestMapper>` in **shibboleth2.xml** are supported or not. Because this is less efficient to support in Apache 2.4, this option is provided to decrease request processing time in the event that such plugins are not in use. Disabling this does **not** prevent other features of the `<RequestMapper>` from being supported

Enabling the Module for Authentication

⚠ **Need AuthType and Require**

You **MUST** supply the `AuthType` and `Require` commands at or above the "level" of the content you want to protect in the document tree, or the module won't run. You **CANNOT** rely solely on the `<RequestMapper>` because of Apache's internal design.

One of the "quirks" (I would say "bugs") in Apache is that it requires a complicated set of inter-related general commands to be in place in order for an "auth" module to actually "see" a request. Just because you load the module doesn't mean Apache will ever call on it to do any work. This can make things confusing; if you see Apache just serving up content and the SP seems to be ignoring the requests, the lack of these commands in place is is usually the problem.

Specifically, you **MUST** create a command pair of `AuthType` and `Require` for the content you want to protect, or the module will not run. There are normally a couple of different strategies for this.

If you're using the module to protect specific content on the server, and you expect the module to step in and force the user to login automatically, you can place the following commands inside any appropriate content block or `.htaccess` file:

Enabling the Module for Specific Content

```
AuthType shibboleth
ShibRequestSetting requireSession 1
Require valid-user
```

The two generic commands around the middle one are Apache's way of signaling that you want the module to run, and that any authenticated user for which an identifier is set is acceptable. The middle setting tells the SP to perform authentication any time a session isn't already in place, which ensures that the authorization rule can be met.

Note that if you use a different `AuthType`, or leave out either one of those general commands, various errors will result.

To exclude a specific path from the directory above add a Location override like this:

Exclude a directory from authentication

```
<Location /public>
 AuthType Shibboleth
 ShibRequestSetting requireSession false
 Require shibboleth
 </Location>
```

Another common trick is to enable the module across an entire server or at least virtual host, but leave specific rules for authentication and access to commands in other places. This introduces a bit of inefficiency, but does simplify the rest of your configuration:

Enabling the Module Globally (Overrides other Authentication Rules)

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

The difference here is that unless some other option is introduced, the module won't actually do anything, but it's always going to be minimally active. The special keyword "shibboleth" on the `Require` command is a way of telling the module that it should be active, but not actually block any access by default.

Note that using a global rule as above will override and circumvent rules applied in `<Directory>` blocks or in local htaccess files. This includes both Shibboleth rules or rules for other authentication methods that might be in use. The above should **only** be used when the entire server is dedicated to hosting a single Shibboleth-enabled application that performs its own authorization.

Authorization

The SP contains a plugin interface for Access Control and includes a pair of implementations, one that's portable and one that enables "familiar" use of the Apache `Require` command in the usual places, including `.htaccess` files.

With Apache 2.4, the `ShibRequestMapperAuthz` command allows the portable option to be bypassed for efficiency if not used.

For details on the portable option, see the NativeSPXMLAccessControl topic.

For details on the `.htaccess` option, see the htaccess topic.

Content Settings

The SP supports an extensible set of content settings, properties that control how it interacts with requests and enforces various requirements.

On Apache, these settings can be controlled using either the `ShibRequestSetting` command (really more of a meta-command), or by attaching properties using the `<RequestMapper>` mechanism in the SP configuration. By default, the SP ships with the "Native" RequestMapper plugin enabled. This plugin is what permits settings to be attached using either the native Apache command, or the XML syntax.

To use the `ShibRequestSetting` command, you simply add a pair of parameters with the name of the content setting and the value you want.

For example, to set `applicationId` to `foo`, the command would be:

```
ShibRequestSetting applicationId foo
```

Apache Takes Precedence

The Apache command will always take precedence over anything you put into the `<RequestMapper>`, so be careful if you combine the two (and be careful if you allow "AuthConfig" overrides, since that will allow control over SP behavior by anybody with the ability to control `.htaccess` files. To prevent this outright, you can alter the plugin type to "XML" to prevent the `ShibRequestSetting` command from functioning.

The general use of Apache commands should be self-explanatory (or at least should be explained by reading/learning about Apache configuration). For more information about using the `<RequestMapper>` feature, refer to the How to do Request Mapping in the SP topic.

Secure Use of the RequestMapper on Apache

Significant caution should be exercised when using the `<RequestMapper>` with Apache. If you do so, you **MUST** ensure that the `UseCanonicalName` command is `On` (it's usually not enabled by default). This is discussed above in the "Prepping Apache" section. While the mechanism is safe if the directions are followed, unless portability is a significant requirement it's a better idea to stick to Apache commands instead.