Authentication and Authorization

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone is allowed to be where they want to go, or to have information that they want to have.

For general access control, see the Access Control How-To.

Related Modules and Directives

There are three types of modules involved in the authentication and authorization process. You will usually need to choose at least one module from each group.

- Authentication type (see the `AuthType` directive)
  - mod_auth_basic
  - mod_auth_digest
- Authentication provider (see the `AuthBasicProvider` and `AuthDigestProvider` directives)
  - mod_authn_anon
  - mod_authn_dbd
  - mod_authn_dbm
  - mod_authn_file
  - mod_authnz_ldap
  - mod_authn_socache
- Authorization (see the `Require` directive)
  - mod_authnz_ldap
  - mod_authz_dbd
  - mod_authz_dbm
  - mod_authz_groupfile
  - mod_authz_host
  - mod_authz_owner
  - mod_authz_user

In addition to these modules, there are also mod_authn_core and mod_authz_core. These modules implement core directives that are core to all auth modules.

The module mod_authnz_ldap is both an authentication and authorization provider. The module mod_authz_host provides authorization and access control based on hostname, IP address or characteristics of the request, but is not part of the authentication provider system. For backwards compatibility with the mod_access, there is a new module mod_access_compat.

You probably also want to take a look at the Access Control howto, which discusses the various ways to control access to your server.

Introduction

If you have information on your web site that is sensitive or intended for only a small group of people, the techniques in this article will help you make sure that the people that see those pages are the people that you wanted to see them.

This article covers the "standard" way of protecting parts of your web site that most of you are going to use.

Note:

If your data really needs to be secure, consider using mod_ssl in addition to any authentication.

The Prerequisites

The directives discussed in this article will need to go either in your main server configuration file (typically in a <Directory> section), or in per-directory configuration files (.htaccess files).

If you plan to use .htaccess files, you will need to have a server configuration that permits putting authentication directives in these files. This is done with the AllowOverride directive, which specifies which directives, if any, may be put in per-directory configuration files.

Since we're talking here about authentication, you will need an AllowOverride directive like the following:

```
AllowOverride AuthConfig
```

Or, if you are just going to put the directives directly in your main server configuration file, you will of course need to have write permission to that file.

And you'll need to know a little bit about the directory structure of your server, in order to know where some files are kept. This should not be terribly difficult, and I'll try to make this clear when we come to that point.

You will also need to make sure that the modules mod_authn_core and mod_authz_core have either been built into the httpd binary or loaded by the httpd.conf configuration file. Both of these modules provide core directives and functionality that are critical to the configuration and use of authentication and authorization in the web server.

Getting it working

Here's the basics of password protecting a directory on your server.

First, you need to create a password file. Exactly how you do this will vary depending on what authentication provider you have chosen. More on that later. To start with, we'll use a text password file.

This file should be placed somewhere not accessible from the web. This is so that folks cannot download the password file. For example, if your documents are served out of /usr/local/apache/htdocs, you might want to put the password file(s) in /usr/local/apache/passwd.

To create the file, use the htpasswd utility that came with Apache. This will be located in the bin directory of wherever you installed Apache. If you have installed Apache from a third-party package, it may be in your execution path.

To create the file, type:

```
htpasswd -c /usr/local/apache/passwd/passwords rbowen
```

htpasswd will ask you for the password, and then ask you to type it again to confirm it:

```
# htpasswd -c /usr/local/apache/passwd/passwords rbowen
New password: mypassword
Re-type new password: mypassword
Adding password for user rbowen
```

If htpasswd is not in your path, of course you'll have to type the full path to the file to get it to run. With a default installation, it's located at /usr/local/apache2/bin/htpasswd

Next, you'll need to configure the server to request a password and tell the server which users are allowed access. You can do this either by editing the httpd.conf file or using an .htaccess file. For example, if you wish to protect the directory /usr/local/apache/htdocs/secret, you can use the following directives, either placed in the file /usr/local/apache/htdocs/secret/.htaccess, or placed in httpd.conf inside a <Directory "/usr/local/apache/htdocs/secret"> section.

```
AuthType Basic
AuthName "Restricted Files"
# (Following line optional)
AuthBasicProvider file
AuthUserFile "/usr/local/apache/passwd/passwords"
Require user rbowen
```

Let's examine each of those directives individually. The AuthType directive selects the method that is used to authenticate the user. The most common method is Basic, and this is the method implemented by mod_auth_basic. It is important to be aware, however, that Basic authentication sends the password from the client to the server unencrypted. This method should therefore not be used for highly sensitive data, unless accompanied by mod_ssl. Apache supports one other authentication method: AuthType Digest. This method is implemented by mod_auth_digest and was intended to be more secure. This is no longer the case and the connection should be encrypted with mod_ssl instead.

The AuthName directive sets the *Realm* to be used in the authentication. The realm serves two major functions. First, the client often presents this information to the user as part of the password dialog box. Second, it is used by the client to determine what password to send for a given authenticated area.

So, for example, once a client has authenticated in the "Restricted Files" area, it will automatically retry the same password for any area on the same server that is marked with the "Restricted Files" Realm. Therefore, you can prevent a user from being prompted more than once for a password by letting multiple restricted areas share the same realm. Of course, for security reasons, the client will always need to ask again for the password whenever the hostname of the server changes.

The AuthBasicProvider is, in this case, optional, since file is the default value for this directive. You'll need to use this directive if you are choosing a different source for authentication, such as mod_authn_dbm or mod_authn_dbd.

The AuthUserFile directive sets the path to the password file that we just created with htpasswd. If you have a large number of users, it can be quite slow to search through a plain text file to authenticate the user on each request. Apache also has the ability to store user information in fast database files. The mod_authn_dbm module provides the AuthDBMUserFile directive. These files can be created and manipulated with the dbmmanage and htdbm programs. Many other types of authentication options are available from third party modules in the Apache Modules Database.

Finally, the Require directive provides the authorization part of the process by setting the user that is allowed to access this region of the server. In the next section, we discuss various ways to use the Require directive.

Letting more than one person in

The directives above only let one person (specifically someone with a username of `rbowen`) into the directory. In most cases, you'll want to let more than one person in. This is where the `AuthGroupFile` comes in.

If you want to let more than one person in, you'll need to create a group file that associates group names with a list of users in that group. The format of this file is pretty simple, and you can create it with your favorite editor. The contents of the file will look like this:

```
GroupName: rbowen dpitts sungo rshersey
```

That's just a list of the members of the group in a long line separated by spaces.

To add a user to your already existing password file, type:

```
htpasswd /usr/local/apache/passwd/passwords dpitts
```

You'll get the same response as before, but it will be appended to the existing file, rather than creating a new file. (It's the `-c` that makes it create a new password file).

Now, you need to modify your `.htaccess` file to look like the following:

```
AuthType Basic
AuthName "By Invitation Only"
# Optional line:
AuthBasicProvider file
AuthUserFile "/usr/local/apache/passwd/passwords"
AuthGroupFile "/usr/local/apache/passwd/groups"
Require group GroupName
```

Now, anyone that is listed in the group `GroupName`, and has an entry in the `password` file, will be let in, if they type the correct password.

There's another way to let multiple users in that is less specific. Rather than creating a group file, you can just use the following directive:

```
Require valid-user
```

Using that rather than the `Require user rbowen` line will allow anyone in that is listed in the password file, and who correctly enters their password. You can even emulate the group behavior here, by just keeping a separate password file for each group. The advantage of this approach is that Apache only has to check one file, rather than two. The disadvantage is that you have to maintain a bunch of password files, and remember to reference the right one in the `AuthUserFile` directive.

Possible problems

Because of the way that Basic authentication is specified, your username and password must be verified every time you request a document from the server. This is even if you're reloading the same page, and for every image on the page (if they come from a protected directory). As you can imagine, this slows things down a little. The amount that it slows things down is proportional to the size of the password file, because it has to open up that file, and go down the list of users until it gets to your name. And it has to do this every time a page is loaded.

A consequence of this is that there's a practical limit to how many users you can put in one password file. This limit will vary depending on the performance of your particular server machine, but you can expect to see slowdowns once you get above a few hundred entries, and may wish to consider a different authentication method at that time.

Alternate password storage

Because storing passwords in plain text files has the above problems, you may wish to store your passwords somewhere else, such as in a database.

mod_authn_dbm and mod_authn_dbd are two modules which make this possible. Rather than selecting AuthBasicProvider file, instead you can choose dbm or dbd as your storage format.

To select a dbm file rather than a text file, for example:

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider dbm
    AuthDBMUserFile "/www/passwords/passwd.dbm"
    Require valid-user
</Directory>
```

Other options are available. Consult the mod_authn_dbm documentation for more details.

Using multiple providers

With the introduction of the new provider based authentication and authorization architecture, you are no longer locked into a single authentication or authorization method. In fact any number of the providers can be mixed and matched to provide you with exactly the scheme that meets your needs. In the following example, both the file and LDAP based authentication providers are being used.

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider file ldap
    AuthUserFile "/usr/local/apache/passwd/passwords"
    AuthLDAPURL ldap://ldaphost/o=yourorg
    Require valid-user
</Directory>
```

In this example the file provider will attempt to authenticate the user first. If it is unable to authenticate the user, the LDAP provider will be called. This allows the scope of authentication to be broadened if your organization implements more than one type of authentication store. Other authentication and authorization scenarios may include mixing one type of authentication with a different type of authorization. For example, authenticating against a password file yet authorizing against an LDAP directory.

Just as multiple authentication providers can be implemented, multiple authorization methods can also be used. In this example both file group authorization as well as LDAP group authorization is being used.

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider file
    AuthUserFile "/usr/local/apache/passwd/passwords"
    AuthLDAPURL ldap://ldaphost/o=yourorg
    AuthGroupFile "/usr/local/apache/passwd/groups"
    Require group GroupName
    Require ldap-group cn=mygroup,o=yourorg
</Directory>
```

To take authorization a little further, authorization container directives such as <RequireAll> and <RequireAny> allow logic to be applied so that the order in

which authorization is handled can be completely controlled through the configuration. See Authorization Containers for an example of how they may be applied.

Beyond just authorization

The way that authorization can be applied is now much more flexible than just a single check against a single data store. Ordering, logic and choosing how authorization will be done is now possible.

Applying logic and ordering

Controlling how and in what order authorization will be applied has been a bit of a mystery in the past. In Apache 2.2 a provider-based authentication mechanism was introduced to decouple the actual authentication process from authorization and supporting functionality. One of the side benefits was that authentication providers could be configured and called in a specific order which didn't depend on the load order of the auth module itself. This same provider based mechanism has been brought forward into authorization as well. What this means is that the `Require` directive not only specifies which authorization methods should be used, it also specifies the order in which they are called. Multiple authorization methods are called in the same order in which the `Require` directives appear in the configuration.

With the introduction of authorization container directives such as `<RequireAll>` and `<RequireAny>`, the configuration also has control over when the authorization methods are called and what criteria determines when access is granted. See Authorization Containers for an example of how they may be used to express complex authorization logic.

By default all `Require` directives are handled as though contained within a `<RequireAny>` container directive. In other words, if any of the specified authorization methods succeed, then authorization is granted.

Using authorization providers for access control

Authentication by username and password is only part of the story. Frequently you want to let people in based on something other than who they are. Something such as where they are coming from.

The authorization providers `all`, `env`, `host` and `ip` let you allow or deny access based on other host based criteria such as host name or ip address of the machine requesting a document.

The usage of these providers is specified through the `Require` directive. This directive registers the authorization providers that will be called during the authorization stage of the request processing. For example:

```
Require ip address
```

where *address* is an IP address (or a partial IP address) or:

```
Require host domain_name
```

where *domain_name* is a fully qualified domain name (or a partial domain name); you may provide multiple addresses or domain names, if desired.

For example, if you have someone spamming your message board, and you want to keep them out, you could do the following:

```
<RequireAll>
    Require all granted
    Require not ip 10.252.46.165
</RequireAll>
```

Visitors coming from that address will not be able to see the content covered by this directive. If, instead, you have a machine name, rather than an IP address, you can use that.

```
<RequireAll>
    Require all granted
    Require not host host.example.com
</RequireAll>
```

And, if you'd like to block access from an entire domain, you can specify just part of an address or domain name:

```
<RequireAll>
    Require all granted
    Require not ip 192.168.205
    Require not host phishers.example.com moreidiots.ex
    Require not host ke
</RequireAll>
```

Using `<RequireAll>` with multiple `<Require>` directives, each negated with `not`, will only allow access, if all of negated conditions are true. In other words, access will be blocked, if any of the negated conditions fails.

Access Control backwards compatibility

One of the side effects of adopting a provider based mechanism for authentication is that the previous access control directives `Order`, `Allow`, `Deny` and `Satisfy` are no longer needed. However to provide backwards compatibility for older configurations, these directives have been moved to the `mod_access_compat` module.

> **Note**
>
> The directives provided by `mod_access_compat` have been deprecated by `mod_authz_host`. Mixing old directives like `Order`, `Allow` or `Deny` with new ones like `Require` is technically possible but discouraged. The `mod_access_compat` module was created to support configurations containing only old directives to facilitate the 2.4 upgrade. Please check the upgrading guide for more information.

Authentication Caching

There may be times when authentication puts an unacceptable load on a provider or on your network. This is most likely to affect users of `mod_authn_dbd` (or third-party/custom providers). To deal with this, HTTPD 2.3/2.4 introduces a new caching provider `mod_authn_socache` to cache credentials and reduce the load on the origin provider(s).

This may offer a substantial performance boost to some users.

More information

You should also read the documentation for `mod_auth_basic` and `mod_authz_host` which contain some more information about how this all works. The directive `<AuthnProviderAlias>` can also help in simplifying certain authentication configurations.

The various ciphers supported by Apache for authentication data are explained in Password Encryptions.

And you may want to look at the Access Control howto, which discusses a number of related topics.