Apache Module mod_auth_ldap

Description:	Allows an LDAP directory to be used to store the database for HTTP Basic authentication.
Status:	Experimental
Module Identifier:	auth_ldap_module
Source File:	mod_auth_ldap.c
Compatibility:	Available in version 2.0.41 and later

Summary

`mod_auth_ldap` supports the following features:

- Known to support the OpenLDAP SDK[1] (both 1.x and 2.x), and the iPlanet (Netscape)[2] SDK.
- Complex authorization policies can be implemented by representing the policy with LDAP filters.
- Support for Microsoft FrontPage allows FrontPage users to control access to their webs, while retaining LDAP for user authentication.
- Uses extensive caching of LDAP operations via mod_ldap[3].
- Support for LDAP over SSL (requires the Netscape SDK) or TLS (requires the OpenLDAP 2.x SDK).

Topics

Directives

See also

- `mod_ldap`

Contents

Operation

There are two phases in granting access to a user. The first phase is authentication, in which mod_auth_ldap verifies that the user's credentials are valid. This also called the *search/bind* phase. The second phase is authorization, in which mod_auth_ldap determines if the authenticated user is allowed access to the resource in question. This is also known as the *compare* phase.

The Authentication Phase

During the authentication phase, mod_auth_ldap searches for an entry in the directory that matches the username that the HTTP client passes. If a single unique match is found, then mod_auth_ldap attempts to bind to the directory server using the DN of the entry plus the password provided by the HTTP client. Because it does a search, then a bind, it is often referred to as the search/bind phase. Here are the steps taken during the search/bind phase.

1. Generate a search filter by combining the attribute and filter provided in the AuthLDAPURL directive with the username passed by the HTTP client.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny or decline access.
3. Fetch the distinguished name of the entry retrieved from the search and attempt to bind to the LDAP server using the DN and the password passed by the HTTP client. If the bind is unsuccessful, deny or decline access.

The following directives are used during the search/bind phase

AuthLDAPURL	Specifies the LDAP server, the base DN, the attribute to use in the search, as well as the extra search filter to use.
AuthLDAPBindDN	An optional DN to bind with during the search phase.
AuthLDAPBindPassword	An optional password to bind with during the search phase.

The Authorization Phase

During the authorization phase, mod_auth_ldap attempts to determine if the user is authorized to access the resource. Many of these checks require mod_auth_ldap to do a compare operation on the LDAP server. This is why this phase is often referred to as the compare phase. mod_auth_ldap accepts the following Require directives to determine if the credentials are acceptable:

- Grant access if there is a `require valid-user` directive.
- Grant access if there is a `require user` directive, and the username in the directive matches

the username passed by the client.

- Grant access if there is a `require dn` directive, and the DN in the directive matches the DN fetched from the LDAP directory.
- Grant access if there is a `require group` directive, and the DN fetched from the LDAP directory (or the username passed by the client) occurs in the LDAP group.
- otherwise, deny or decline access

`mod_auth_ldap` uses the following directives during the compare phase:

`AuthLDAPURL`	The attribute specified in the URL is used in compare operations for the `require user` operation.
`AuthLDAPCompareDNOnServer`	Determines the behavior of the `require dn` directive.
`AuthLDAPGroupAttribute`	Determines the attribute to use for comparisons in the `require group` directive.
`AuthLDAPGroupAttributeIsDN`	Specifies whether to use the user DN or the username when doing comparisons for the `require group` directive.

The require Directives

Apache's `Require` directives are used during the authorization phase to ensure that a user is allowed to access a resource.

require valid-user

If this directive exists, `mod_auth_ldap` grants access to any user that has successfully authenticated during the search/bind phase.

require user

The `require user` directive specifies what usernames can access the resource. Once `mod_auth_ldap` has retrieved a unique DN from the directory, it does an LDAP compare operation using the username specified in the `require user` to see if that username is part of the just-fetched LDAP entry. Multiple users can be granted access by putting multiple usernames on the line, separated with spaces. If a username has a space in it, then it must be the only user on the line. In this case, multiple users can be granted access by using multiple `require user` directives, with one user per line. For example, with a `AuthLDAPURL` of `ldap://ldap/o=Airius?cn` (i.e., `cn` is used for searches), the following require directives could be used to restrict access:

```
require user Barbara Jenson
require user Fred User
require user Joe Manager
```

Because of the way that `mod_auth_ldap` handles this directive, Barbara Jenson could sign on as *Barbara Jenson*, *Babs Jenson* or any other `cn` that she has in her LDAP entry. Only the single `require user` line is needed to support all values of the attribute in the user's entry.

If the `uid` attribute was used instead of the `cn` attribute in the URL above, the above three lines could be condensed to

```
require user bjenson fuser jmanager
```

require group

This directive specifies an LDAP group whose members are allowed access. It takes the distinguished name of the LDAP group. For example, assume that the following entry existed in the LDAP directory:

```
dn: cn=Administrators, o=Airius
objectClass: groupOfUniqueNames
uniqueMember: cn=Barbara Jenson, o=Airius
uniqueMember: cn=Fred User, o=Airius
```

The following directive would grant access to both Fred and Barbara:

```
require group cn=Administrators, o=Airius
```

Behavior of this directive is modified by the `AuthLDAPGroupAttribute` and `AuthLDAPGroupAttributeIsDN` directives.

require dn

The `require dn` directive allows the administrator to grant access based on distinguished names. It specifies a DN that must match for access to be granted. If the distinguished name that was retrieved from the directory server matches the distinguished name in the `require dn`, then authorization is granted.

The following directive would grant access to a specific DN:

```
require dn cn=Barbara Jenson, o=Airius
```

Behavior of this directive is modified by the `AuthLDAPCompareDNOnServer` directive.

Examples

- Grant access to anyone who exists in the LDAP directory, using their UID for searches.

  ```
  AuthLDAPURL ldap://ldap1.airius.com:389/ou=People,
  o=Airius?uid?sub?(objectClass=*)
  require valid-user
  ```

- The next example is the same as above; but with the fields that have useful defaults omitted. Also, note the use of a redundant LDAP server.

  ```
  AuthLDAPURL ldap://ldap1.airius.com ldap2.airius.com/ou=People,
  o=Airius
  require valid-user
  ```

- The next example is similar to the previous one, but is uses the common name instead of the UID. Note that this could be problematical if multiple people in the directory share the same `cn`, because a search on `cn` **must** return exactly one entry. That's why this approach is not recommended: it's a better idea to choose an attribute that is guaranteed unique in your directory, such as `uid`.

  ```
  AuthLDAPURL ldap://ldap.airius.com/ou=People, o=Airius?cn
  require valid-user
  ```

- Grant access to anybody in the Administrators group. The users must authenticate using their UID.

```
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid
require group cn=Administrators, o=Airius
```

- The next example assumes that everyone at Airius who carries an alphanumeric pager will have an LDAP attribute of `qpagePagerID`. The example will grant access only to people (authenticated via their UID) who have alphanumeric pagers:

```
AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid??(qpagePagerID=*)
require valid-user
```

- The next example demonstrates the power of using filters to accomplish complicated administrative requirements. Without filters, it would have been necessary to create a new LDAP group and ensure that the group's members remain synchronized with the pager users. This becomes trivial with filters. The goal is to grant access to anyone who has a filter, plus grant access to Joe Manager, who doesn't have a pager, but does need to access the same resource:

```
AuthLDAPURL
ldap://ldap.airius.com/o=Airius?uid??(|(qpagePagerID=*)(uid=jmanager))
require valid-user
```

  This last may look confusing at first, so it helps to evaluate what the search filter will look like based on who connects, as shown below. The text in blue is the part that is filled in using the attribute specified in the URL. The text in red is the part that is filled in using the filter specified in the URL. The text in green is filled in using the information that is retrieved from the HTTP client. If Fred User connects as `fuser`, the filter would look like

```
(&(|(qpagePagerID=*)(uid=jmanager))(uid=fuser))
```

  The above search will only succeed if *fuser* has a pager. When Joe Manager connects as *jmanager*, the filter looks like

```
(&(|(qpagePagerID=*)(uid=jmanager))(uid=jmanager))
```

  The above search will succeed whether *jmanager* has a pager or not.

Using TLS

To use TLS, simply set the `AuthLDAPStartTLS` to on. Nothing else needs to be done (other than ensure that your LDAP server is configured for TLS).

Using SSL

If `mod_auth_ldap` is linked against the Netscape/iPlanet LDAP SDK, it will not talk to any SSL server unless that server has a certificate signed by a known Certificate Authority. As part of the configuration `mod_auth_ldap` needs to be told where it can find a database containing the known CAs. This database is in the same format as Netscape Communicator's `cert7.db` database. The easiest way to get this file is to start up a fresh copy of Netscape, and grab the resulting `$HOME/.netscape/cert7.db` file.

To specify a secure LDAP server, use *ldaps://* in the `AuthLDAPURL` directive, instead of *ldap://*.

Using Microsoft FrontPage with mod_auth_ldap

Normally, FrontPage uses FrontPage-web-specific user/group files (i.e., the `mod_authn_file` and `mod_authz_groupfile` modules) to handle all authentication. Unfortunately, it is not possible to just change to LDAP authentication by adding the proper directives, because it will break the *Permissions* forms in the FrontPage client, which attempt to modify the standard text-based authorization files.

Once a FrontPage web has been created, adding LDAP authentication to it is a matter of adding the following directives to *every* `.htaccess` file that gets created in the web

```
AuthLDAPURL             the url
AuthLDAPAuthoritative   off
AuthLDAPFrontPageHack   on
```

`AuthLDAPAuthoritative` must be off to allow `mod_auth_ldap` to decline group authentication so that Apache will fall back to file authentication for checking group membership. This allows the FrontPage-managed group file to be used.

How It Works

FrontPage restricts access to a web by adding the `require valid-user` directive to the `.htaccess` files. If `AuthLDAPFrontPageHack` is not on, the `require valid-user` directive will succeed for any user who is valid *as far as LDAP is concerned*. This means that anybody who has an entry in the LDAP directory is considered a valid user, whereas FrontPage considers only those people in the local user file to be valid. The purpose of the hack is to force Apache to consult the local user file (which is managed by FrontPage) - instead of LDAP - when handling the `require valid-user` directive.

Once directives have been added as specified above, FrontPage users will be able to perform all management operations from the FrontPage client.

Caveats

- When choosing the LDAP URL, the attribute to use for authentication should be something that will also be valid for putting into a `mod_authn_file` user file. The user ID is ideal for this.
- When adding users via FrontPage, FrontPage administrators should choose usernames that already exist in the LDAP directory (for obvious reasons). Also, the password that the administrator enters into the form is ignored, since Apache will actually be authenticating against the password in the LDAP database, and not against the password in the local user file. This could cause confusion for web administrators.
- Apache must be compiled with `mod_auth_basic`, `mod_authn_file` and `mod_authz_groupfile` in order to use FrontPage support. This is because Apache will still use the `mod_authz_groupfile` group file for determine the extent of a user's access to the FrontPage web.
- The directives must be put in the `.htaccess` files. Attempting to put them inside `<Location>` or `<Directory>` directives won't work. This is because `mod_auth_ldap` has to be able to grab the `AuthUserFile` directive that is found in FrontPage `.htaccess` files so that it knows where to look for the valid user list. If the `mod_auth_ldap` directives aren't in the same `.htaccess` file as the FrontPage directives, then the hack won't work, because `mod_auth_ldap` will never

get a chance to process the `.htaccess` file, and won't be able to find the FrontPage-managed user file.

AuthLDAPAuthoritative Directive

Description:	Prevent other authentication modules from authenticating the user if this one fails
Syntax:	`AuthLDAPAuthoritative on|off`
Default:	`AuthLDAPAuthoritative on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

Set to `off` if this module should let other authentication modules attempt to authenticate the user, should authentication with this module fail. Control is only passed on to lower modules if there is no DN or rule that matches the supplied user name (as passed by the client).

AuthLDAPBindDN Directive

Description:	Optional DN to use in binding to the LDAP server
Syntax:	`AuthLDAPBindDN` *`distinguished-name`*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

An optional DN used to bind to the server when searching for entries. If not provided, `mod_auth_ldap` will use an anonymous bind.

AuthLDAPBindPassword Directive

Description:	Password used in conjuction with the bind DN
Syntax:	`AuthLDAPBindPassword` *`password`*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

A bind password to use in conjunction with the bind DN. Note that the bind password is probably sensitive data, and should be properly protected. You should only use the `AuthLDAPBindDN` and `AuthLDAPBindPassword` if you absolutely need them to search the directory.

AuthLDAPCharsetConfig Directive

Description:	Language to charset conversion configuration file
Syntax:	`AuthLDAPCharsetConfig` *`file-path`*

Context:	server config
Status:	Experimental
Module:	mod_auth_ldap

The `AuthLDAPCharsetConfig` directive sets the location of the language to charset conversion configuration file. *File-path* is relative to the `ServerRoot`. This file specifies the list of language extensions to character sets. Most administrators use the provided `charset.conv` file, which associates common language extensions to character sets.

The file contains lines in the following format:

```
Language-Extension charset [Language-String] ...
```

The case of the extension does not matter. Blank lines, and lines beginning with a hash character (#) are ignored.

AuthLDAPCompareDNOnServer Directive

Description:	Use the LDAP server to compare the DNs
Syntax:	`AuthLDAPCompareDNOnServer on|off`
Default:	`AuthLDAPCompareDNOnServer on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

When set, `mod_auth_ldap` will use the LDAP server to compare the DNs. This is the only foolproof way to compare DNs. `mod_auth_ldap` will search the directory for the DN specified with the `require dn` directive, then, retrieve the DN and compare it with the DN retrieved from the user entry. If this directive is not set, `mod_auth_ldap` simply does a string comparison. It is possible to get false negatives with this approach, but it is much faster. Note the `mod_ldap` cache can speed up DN comparison in most situations.

AuthLDAPDereferenceAliases Directive

Description:	When will the module de-reference aliases
Syntax:	`AuthLDAPDereferenceAliases never|searching|finding|always`
Default:	`AuthLDAPDereferenceAliases Always`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

This directive specifies when `mod_auth_ldap` will de-reference aliases during LDAP operations. The default is `always`.

AuthLDAPEnabled Directive

Description:	Turn on or off LDAP authentication
Syntax:	`AuthLDAPEnabled on|off`
Default:	`AuthLDAPEnabled on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

Set to `off` to disable `mod_auth_ldap` in certain directories. This is useful if you have `mod_auth_ldap` enabled at or near the top of your tree, but want to disable it completely in certain locations.

AuthLDAPFrontPageHack Directive

Description:	Allow LDAP authentication to work with MS FrontPage
Syntax:	`AuthLDAPFrontPageHack on|off`
Default:	`AuthLDAPFronPageHack off`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

See the section on using Microsoft FrontPage with `mod_auth_ldap`.

AuthLDAPGroupAttribute Directive

Description:	LDAP attributes used to check for group membership
Syntax:	`AuthLDAPGroupAttribute` *`attribute`*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

This directive specifies which LDAP attributes are used to check for group membership. Multiple attributes can be used by specifying this directive multiple times. If not specified, then `mod_auth_ldap` uses the `member` and `uniquemember` attributes.

AuthLDAPGroupAttributeIsDN Directive

Description:	Use the DN of the client username when checking for group membership
Syntax:	`AuthLDAPGroupAttributeIsDN on|off`
Default:	`AuthLDAPGroupAttributeIsDN on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental

Module:	mod_auth_ldap

When set `on`, this directive says to use the distinguished name of the client username when checking for group membership. Otherwise, the username will be used. For example, assume that the client sent the username `bjenson`, which corresponds to the LDAP DN `cn=Babs Jenson, o=Airius`. If this directive is set, `mod_auth_ldap` will check if the group has `cn=Babs Jenson, o=Airius` as a member. If this directive is not set, then `mod_auth_ldap` will check if the group has `bjenson` as a member.

AuthLDAPRemoteUserIsDN Directive

Description:	Use the DN of the client username to set the REMOTE_USER environment variable
Syntax:	`AuthLDAPRemoteUserIsDN on|off`
Default:	`AuthLDAPUserIsDN off`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

If this directive is set to on, the value of the `REMOTE_USER` environment variable will be set to the full distinguished name of the authenticated user, rather than just the username that was passed by the client. It is turned off by default.

AuthLDAPStartTLS Directive

Description:	Use a secure TLS connection to the LDAP server
Syntax:	`AuthLDAPStartTLS on|off`
Default:	`AuthLDAPStartTLS off`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

If this directive is set to `on`, `mod_auth_ldap` will start a secure TLS session after connecting to the LDAP server. This requires your LDAP server to support TLS.

AuthLDAPUrl Directive

Description:	URL specifying the LDAP search parameters
Syntax:	`AuthLDAPUrl` *`url`*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_ldap

An RFC 2255 URL which specifies the LDAP search parameters to use. The syntax of the URL is

```
ldap://host:port/basedn?attribute?scope?filter
```

ldap
: For regular ldap, use the string `ldap`. For secure LDAP, use `ldaps` instead. Secure LDAP is only available if Apache was linked to an LDAP library with SSL support.

host:port
: The name/port of the ldap server (defaults to `localhost:389` for `ldap`, and `localhost:636` for `ldaps`). To specify multiple, redundant LDAP servers, just list all servers, separated by spaces. `mod_auth_ldap` will try connecting to each server in turn, until it makes a successful connection.

  Once a connection has been made to a server, that connection remains active for the life of the `httpd` process, or until the LDAP server goes down.

  If the LDAP server goes down and breaks an existing connection, `mod_auth_ldap` will attempt to re-connect, starting with the primary server, and trying each redundant server in turn. Note that this is different than a true round-robin search.

basedn
: The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but could also specify a subtree in the directory.

attribute
: The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use `uid`. It's a good idea to choose an attribute that will be unique across all entries in the subtree you will be using.

scope
: The scope of the search. Can be either `one` or `sub`. Note that a scope of `base` is also supported by RFC 2255, but is not supported by this module. If the scope is not provided, or if `base` scope is specified, the default is to use a scope of `sub`.

filter
: A valid LDAP search filter. If not provided, defaults to `(objectClass=*)`, which will search for all objects in the tree. Filters are limited to approximately 8000 characters (the definition of `MAX_STRING_LEN` in the Apache source code). This should be than sufficient for any application.

When doing searches, the attribute, filter and username passed by the HTTP client are combined to create a search filter that looks like `(&(`*`filter`*`)(`*`attribute=username`*`))`.

For example, consider an URL of `ldap://ldap.airius.com/o=Airius?cn?sub?(posixid=*)`. When a client attempts to connect using a username of `Babs Jenson`, the resulting search filter will be `(&(posixid=*)(cn=Babs Jenson))`.

See above for examples of `AuthLDAPURL` URLs.

URI References

[1] http://www.openldap.org/
[2] http://www.iplanet.com/downloads/developer/
[3] http://httpd.apache.org/docs-2.1/mod/mod_ldap.html