LDAP Authentication

Toby Burress <kurin@causa-sui.net >
Revision: 44962

Copyright © 2007, 2008 The FreeBSD Documentation Project

FreeBSD is a registered trademark of the FreeBSD Foundation.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the "™" or the "®" symbol.

2014-05-26 by bcr.

Abstract

This document is intended as a guide for the configuration of an LDAP server (principally an OpenLDAP server) for authentication on FreeBSD. This is useful for situations where many servers need the same user accounts, for example as a replacement for NIS.

Table of Contents

1. Preface 1
2. Configuring LDAP 2
3. Client Configuration 7
4. Security Considerations 12
A. Useful Aids 14
B. OpenSSL Certificates for LDAP 15

1. Preface

This document is intended to give the reader enough of an understanding of LDAP to configure an LDAP server. This document will attempt to provide an explanation of net/nss_ldap and security/pam_ldap for use with client machines services for use with the LDAP server.

When finished, the reader should be able to configure and deploy a FreeBSD server that can host an LDAP directory, and to configure and deploy a FreeBSD server which can authenticate against an LDAP directory.

This article is not intended to be an exhaustive account of the security, robustness, or best practice considerations for configuring LDAP or the other services discussed herein. While the author takes care to do everything correctly, he does not address security issues beyond a general scope. This article should be considered to lay the theoretical groundwork only, and any actual implementation should be accompanied by careful requirement analysis.

2. Configuring LDAP

LDAP stands for “Lightweight Directory Access Protocol” and is a subset of the X.500 Directory Access Protocol. Its most recent specifications are in RFC4510 and friends. Essentially it is a database that expects to be read from more often than it is written to.

The LDAP server OpenLDAP will be used in the examples in this document; while the principles here should be generally applicable to many different servers, most of the concrete administration is OpenLDAP-specific. There are several server versions in ports, for example net/openldap24-server. Client servers will need the corresponding net/openldap24-client libraries.

There are (basically) two areas of the LDAP service which need configuration. The first is setting up a server to receive connections properly, and the second is adding entries to the server's directory so that FreeBSD tools know how to interact with it.

2.1. Setting Up the Server for Connections

Note

This section is specific to OpenLDAP. If you are using another server, you will need to consult that server's documentation.

2.1.1. Installing OpenLDAP

First, install OpenLDAP:

Example 1. Installing OpenLDAP

```
# cd /usr/ports/net/openldap24-server
# make install clean
```

This installs the `slapd` and `slurpd` binaries, along with the required OpenLDAP libraries.

2.1.2. Configuring OpenLDAP

Next we must configure OpenLDAP.

You will want to require encryption in your connections to the LDAP server; otherwise your users' passwords will be transferred in plain text, which is considered insecure. The tools we will be using support two very similar kinds of encryption, SSL and TLS.

TLS stands for "Transportation Layer Security". Services that employ TLS tend to connect on the *same* ports as the same services without TLS; thus an SMTP server which supports TLS will listen for connections on port 25, and an LDAP server will listen on 389.

SSL stands for "Secure Sockets Layer", and services that implement SSL do *not* listen on the same ports as their non-SSL counterparts. Thus SMTPS listens on port 465 (not 25), HTTPS listens on 443, and LDAPS on 636.

The reason SSL uses a different port than TLS is because a TLS connection begins as plain text, and switches to encrypted traffic after the `STARTTLS` directive. SSL connections are encrypted from the beginning. Other than that there are no substantial differences between the two.

Note

We will adjust OpenLDAP to use TLS, as SSL is considered deprecated.

Once OpenLDAP is installed via ports, the following configuration parameters in `/usr/local/etc/openldap/slapd.conf` will enable TLS:

```
security ssf=128

TLSCertificateFile /path/to/your/cert.crt
TLSCertificateKeyFile /path/to/your/cert.key
TLSCACertificateFile /path/to/your/cacert.crt
```

Here, `ssf=128` tells OpenLDAP to require 128-bit encryption for all connections, both search and update. This parameter may be configured based on the security needs of your site, but rarely you need to weaken it, as most LDAP client libraries support strong encryption.

The `cert.crt`, `cert.key`, and `cacert.crt` files are necessary for clients to authenticate *you* as the valid LDAP server. If you simply want a server that runs, you can create a self-signed certificate with OpenSSL:

Example 2. Generating an RSA Key

```
% openssl genrsa -out cert.key 1024
Generating RSA private key, 1024 bit long modulus
.....................++++++
...++++++
e is 65537 (0x10001)
% openssl req -new -key cert.key -out cert.csr
```

At this point you should be prompted for some values. You may enter whatever values you like; however, it is important the "Common Name" value be the fully qualified domain name of the OpenLDAP server. In our case, and the examples here, the server is *server.example.org* . Incorrectly setting this value will cause clients to fail when making connections. This can the cause of great frustration, so ensure that you follow these steps closely.

Finally, the certificate signing request needs to be signed:

Example 3. Self-signing the Certificate

```
% openssl x509 -req -in cert.csr -days 365 -signkey cert.key -
out cert.crt
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd
Getting Private key
```

This will create a self-signed certificate that can be used for the directives in `slapd.conf` , where `cert.crt` and `cacert.crt` are the same file. If you are going to use many OpenLDAP servers (for replication via `slurpd`) you will want to see Appendix B, *OpenSSL Certificates for LDAP* to generate a CA key and use it to sign individual server certificates.

Once this is done, put the following in `/etc/rc.conf` :

```
slapd_enable="YES"
```

Then run **`/usr/local/etc/rc.d/slapd start`** . This should start OpenLDAP. Confirm that it is listening on 389 with

```
% sockstat -4 -p 389
ldap     slapd      3261  7  tcp4   *:389                 *:*
```

2.1.3. Configuring the Client

Install the net/openldap24-client port for the OpenLDAP libraries. The client machines will always have OpenLDAP libraries since that is all security/pam_ldap and net/nss_ldap support, at least for the moment.

The configuration file for the OpenLDAP libraries is `/usr/local/etc/openldap/ldap.conf` . Edit this file to contain the following values:

```
base dc=example,dc=org
uri ldap://server.example.org/
ssl start_tls
tls_cacert /path/to/your/cacert.crt
```

Note

It is important that your clients have access to `cacert.crt` , otherwise they will not be able to connect.

Note

There are two files called `ldap.conf` . The first is this file, which is for the OpenLDAP libraries and defines how to talk to the server. The second is `/usr/local/etc/ldap.conf` , and is for pam_ldap.

At this point you should be able to run **`ldapsearch -Z`** on the client machine; `-Z` means "use TLS". If you encounter an error, then something is configured wrong; most likely it is your certificates. Use openssl(1)'s `s_client` and `s_server` to ensure you have them configured and signed properly.

2.2. Entries in the Database

Authentication against an LDAP directory is generally accomplished by attempting to bind to the directory as the connecting user. This is done by establishing a "simple" bind on the directory with the user name supplied. If there is an entry with the `uid` equal to the user name and that entry's `userPassword` attribute matches the password supplied, then the bind is successful.

The first thing we have to do is figure out is where in the directory our users will live.

The base entry for our database is `dc=example,dc=org` . The default location for users that most clients seem to expect is something like `ou=people,` *`base`*, so that is what will be used here. However keep in mind that this is configurable.

So the ldif entry for the `people` organizational unit will look like:

```
dn: ou=people,dc=example,dc=org
objectClass: top
objectClass: organizationalUnit
ou: people
```

All users will be created as subentries of this organizational unit.

Some thought might be given to the object class your users will belong to. Most tools by default will use `people`, which is fine if you simply want to provide entries against which to authenticate. However, if you are going to store user information in the LDAP database as well, you will probably want to use `inetOrgPerson`, which has many useful attributes. In either case, the relevant schemas need to be loaded in `slapd.conf` .

For this example we will use the `person` object class. If you are using `inetOrgPerson`, the steps are basically identical, except that the `sn` attribute is required.

To add a user `testuser`, the ldif would be:

```
dn: uid=tuser,ou=people,dc=example,dc=org
objectClass: person
objectClass: posixAccount
objectClass: shadowAccount
objectClass: top
uidNumber: 10000
gidNumber: 10000
homeDirectory: /home/tuser
loginShell: /bin/csh
uid: tuser
cn: tuser
```

I start my LDAP users' UIDs at 10000 to avoid collisions with system accounts; you can configure whatever number you wish here, as long as it is less than 65536.

We also need group entries. They are as configurable as user entries, but we will use the defaults below:

```
dn: ou=groups,dc=example,dc=org
objectClass: top
objectClass: organizationalUnit
ou: groups

dn: cn=tuser,ou=groups,dc=example,dc=org
objectClass: posixGroup
objectClass: top
gidNumber: 10000
cn: tuser
```

To enter these into your database, you can use `slapadd` or `ldapadd` on a file containing these entries. Alternatively, you can use sysutils/ldapvi.

The `ldapsearch` utility on the client machine should now return these entries. If it does, your database is properly configured to be used as an LDAP authentication server.

3. Client Configuration

The client should already have OpenLDAP libraries from Section 2.1.3, "Configuring the Client", but if you are installing several client machines you will need to install net/openldap24-client on each of them.

FreeBSD requires two ports to be installed to authenticate against an LDAP server, security/pam_ldap and net/nss_ldap.

3.1. Authentication

security/pam_ldap is configured via `/usr/local/etc/ldap.conf` .

Note

This is a *different file* than the OpenLDAP library functions' configuration file, `/usr/local/etc/openldap/ldap.conf` ; however, it takes many of the same options; in fact it is a superset of that file. For the rest of this section, references to `ldap.conf` will mean `/usr/local/etc/ldap.conf` .

Thus, we will want to copy all of our original configuration parameters from `openldap/ldap.conf` to the new `ldap.conf` . Once this is done, we want to tell security/pam_ldap what to look for on the directory server.

We are identifying our users with the `uid` attribute. To configure this (though it is the default), set the `pam_login_attribute` directive in `ldap.conf` :

Example 4. Setting **`pam_login_attribute`**

```
pam_login_attribute uid
```

With this set, security/pam_ldap will search the entire LDAP directory under `base` for the value `uid=`*`username`*. If it finds one and only one entry, it will attempt to bind as that user with the password it was given. If it binds correctly, then it will allow access. Otherwise it will fail.

3.1.1. PAM

PAM, which stands for "Pluggable Authentication Modules", is the method by which FreeBSD authenticates most of its sessions. To tell FreeBSD we wish to use an LDAP server, we will have to add a line to the appropriate PAM file.

Most of the time the appropriate PAM file is `/etc/pam.d/sshd`, if you want to use SSH (remember to set the relevant options in `/etc/ssh/sshd_config`, otherwise SSH will not use PAM).

To use PAM for authentication, add the line

```
auth  sufficient  /usr/local/lib/pam_ldap.so  no_warn
```

Exactly where this line shows up in the file and which options appear in the fourth column determine the exact behavior of the authentication mechanism; see pam.d(5)

With this configuration you should be able to authenticate a user against an LDAP directory. PAM will perform a bind with your credentials, and if successful will tell SSH to allow access.

However it is not a good idea to allow *every* user in the directory into *every* client machine. With the current configuration, all that a user needs to log into a machine is an LDAP entry. Fortunately there are a few ways to restrict user access.

`ldap.conf` supports a `pam_groupdn` directive; every account that connects to this machine needs to be a member of the group specified here. For example, if you have

```
pam_groupdn cn=servername,ou=accessgroups,dc=example,dc=org
```

in `ldap.conf`, then only members of that group will be able to log in. There are a few things to bear in mind, however.

Members of this group are specified in one or more `memberUid` attributes, and each attribute must have the full distinguished name of the member. So `memberUid: someuser` will not work; it must be:

```
memberUid: uid=someuser,ou=people,dc=example,dc=org
```

Additionally, this directive is not checked in PAM during authentication, it is checked during account management, so you will need a second line in your PAM files under `account`. This will require, in turn, *every* user to be listed in the group, which is not necessarily what we want. To avoid blocking users that are not in LDAP, you should enable the

`ignore_unknown_user` attribute. Finally, you should set the `ignore_authinfo_unavail` option so that you are not locked out of every computer when the LDAP server is unavailable.

Your `pam.d/sshd` might then end up looking like this:

Example 5. Sample **`pam.d/sshd`**

```
auth            required        pam_nologin.so          no_warn
auth            sufficient      pam_opie.so             ↺
no_warn no_fake_prompts
auth            requisite       pam_opieaccess.so       ↺
no_warn allow_local
auth            sufficient      /usr/local/lib/pam_ldap.so
  no_warn
auth            required        pam_unix.so             ↺
no_warn try_first_pass

account         required        pam_login_access.so
account         required        /usr/local/lib/pam_ldap.so
  no_warn ignore_authinfo_unavail ignore_unknown_user
```

Note

Since we are adding these lines specifically to `pam.d/sshd` , this will only have an effect on SSH sessions. LDAP users will be unable to log in at the console. To change this behavior, examine the other files in `/etc/pam.d` and modify them accordingly.

3.2. Name Service Switch

NSS is the service that maps attributes to names. So, for example, if a file is owned by user `1001`, an application will query NSS for the name of `1001`, and it might get `bob` or `ted` or whatever the user's name is.

Now that our user information is kept in LDAP, we need to tell NSS to look there when queried.

The net/nss_ldap port does this. It uses the same configuration file as security/pam_ldap, and should not need any extra parameters once it is installed. Instead, what is left is simply to edit `/etc/nsswitch.conf` to take advantage of the directory. Simply replace the following lines:

```
group: compat
passwd: compat
```

with

```
group: files ldap
passwd: files ldap
```

This will allow you to map usernames to UIDs and UIDs to usernames.

Congratulations! You should now have working LDAP authentication.

3.3. Caveats

Unfortunately, as of the time this was written FreeBSD did not support changing user passwords with passwd(1). Because of this, most administrators are left to implement a solution themselves. I provide some examples here. Note that if you write your own password change script, there are some security issues you should be made aware of; see Section 4.3, “Password Storage”

Example 6. Shell Script for Changing Passwords

```
#!/bin/sh

stty -echo
read -p "Old Password: " oldp; echo
read -p "New Password: " np1; echo
read -p "Retype New Password: " np2; echo
stty echo

if [ "$np1" != "$np2" -]; then
  echo "Passwords do not match."
  exit 1
fi

ldappasswd -D uid="$USER",ou=people,dc=example,dc=org \
  -w "$oldp" \
  -a "$oldp" \
  -s "$np1"
```

Caution

This script does hardly any error checking, but more important it is very cavalier about how it stores your passwords. If you do anything

like this, at least adjust the `security.bsd.see_other_uids` sysctl value:

```
# sysctl security.bsd.see_other_uids=0  .
```

A more flexible (and probably more secure) approach can be used by writing a custom program, or even a web interface. The following is part of a Ruby library that can change LDAP passwords. It sees use both on the command line, and on the web.

Example 7. Ruby Script for Changing Passwords

```
require 'ldap'
require 'base64'
require 'digest'
require 'password' # ruby-password

ldap_server = "ldap.example.org"
luser = "uid=#{ENV['USER']},ou=people,dc=example,dc=org"

# get the new password, check it, and create a salted hash ↺
from it
def get_password
  pwd1 = Password.get("New Password: ")
  pwd2 = Password.get("Retype New Password: ")

  raise if pwd1 != pwd2
  pwd1.check # check password strength

  salt = rand.to_s.gsub(/0\./, '')
  pass = pwd1.to_s
  hash = "{SSHA}"+Base64.encode64(Digest::SHA1.digest↺
("#{pass}#{salt}")+salt).chomp!
  return hash
end

oldp = Password.get("Old Password: ")
newp = get_password

# We'll just replace it.  That we can bind proves that we ↺
either know
# the old password or are an admin.

replace = LDAP::Mod.new(LDAP::LDAP_MOD_REPLACE | ↺
LDAP::LDAP_MOD_BVALUES,
                        "userPassword",
                        [newp])

conn = LDAP::SSLConn.new(ldap_server, 389, true)
```

```
conn.set_option(LDAP::LDAP_OPT_PROTOCOL_VERSION, 3)
conn.bind(luser, oldp)
conn.modify(luser, [replace])
```

Although not guaranteed to be free of security holes (the password is kept in memory, for example) this is cleaner and more flexible than a simple `sh` script.

4. Security Considerations

Now that your machines (and possibly other services) are authenticating against your LDAP server, this server needs to be protected at least as well as `/etc/master.passwd` would be on a regular server, and possibly even more so since a broken or cracked LDAP server would break every client service.

Remember, this section is not exhaustive. You should continually review your configuration and procedures for improvements.

4.1. Setting Attributes Read-only

Several attributes in LDAP should be read-only. If left writable by the user, for example, a user could change his `uidNumber` attribute to `0` and get `root` access!

To begin with, the `userPassword` attribute should not be world-readable. By default, anyone who can connect to the LDAP server can read this attribute. To disable this, put the following in `slapd.conf` :

Example 8. Hide Passwords

```
access to dn.subtree="ou=people,dc=example,dc=org"
  attrs=userPassword
  by self write
  by anonymous auth
  by * none

access to *
  by self write
  by * read
```

This will disallow reading of the `userPassword` attribute, while still allowing users to change their own passwords.

Additionally, you'll want to keep users from changing some of their own attributes. By default, users can change any attribute (except for those which the LDAP schemas themselves deny changes), such as `uidNumber`. To close this hole, modify the above to

Example 9. Read-only Attributes

```
access to dn.subtree="ou=people,dc=example,dc=org"
  attrs=userPassword
  by self write
  by anonymous auth
  by * none

access to attrs=homeDirectory,uidNumber,gidNumber
  by * read

access to *
  by self write
  by * read
```

This will stop users from being able to masquerade as other users.

4.2. `root` Account Definition

Often the `root` or manager account for the LDAP service will be defined in the configuration file. OpenLDAP supports this, for example, and it works, but it can lead to trouble if `slapd.conf` is compromised. It may be better to use this only to bootstrap yourself into LDAP, and then define a `root` account there.

Even better is to define accounts that have limited permissions, and omit a `root` account entirely. For example, users that can add or remove user accounts are added to one group, but they cannot themselves change the membership of this group. Such a security policy would help mitigate the effects of a leaked password.

4.2.1. Creating a Management Group

Say you want your IT department to be able to change home directories for users, but you do not want all of them to be able to add or remove users. The way to do this is to add a group for these admins:

Example 10. Creating a Management Group

```
dn: cn=homemanagement,dc=example,dc=org
```

```
objectClass: top
objectClass: posixGroup
cn: homemanagement
gidNumber: 121 # required for posixGroup
memberUid: uid=tuser,ou=people,dc=example,dc=org
memberUid: uid=user2,ou=people,dc=example,dc=org
```

And then change the permissions attributes in `slapd.conf` :

Example 11. ACLs for a Home Directory Management Group

```
access to dn.subtree="ou=people,dc=example,dc=org"
  attr=homeDirectory
  by dn="cn=homemanagement,dc=example,dc=org"
  dnattr=memberUid write
```

Now `tuser` and `user2` can change other users' home directories.

In this example we have given a subset of administrative power to certain users without giving them power in other domains. The idea is that soon no single user account has the power of a `root` account, but every power root had is had by at least one user. The `root` account then becomes unnecessary and can be removed.

4.3. Password Storage

By default OpenLDAP will store the value of the `userPassword` attribute as it stores any other data: in the clear. Most of the time it is base 64 encoded, which provides enough protection to keep an honest administrator from knowing your password, but little else.

It is a good idea, then, to store passwords in a more secure format, such as SSHA (salted SHA). This is done by whatever program you use to change users' passwords.

A. Useful Aids

There are a few other programs that might be useful, particularly if you have many users and do not want to configure everything manually.

security/pam_mkhomedir is a PAM module that always succeeds; its purpose is to create home directories for users which do not have them. If you have dozens of client servers

and hundreds of users, it is much easier to use this and set up skeleton directories than to prepare every home directory.

sysutils/cpu is a pw(8)-like utility that can be used to manage users in the LDAP directory. You can call it directly, or wrap scripts around it. It can handle both TLS (with the `-x` flag) and SSL (directly).

sysutils/ldapvi is a great utility for editing LDAP values in an LDIF-like syntax. The directory (or subsection of the directory) is presented in the editor chosen by the `EDITOR` environment variable. This makes it easy to enable large-scale changes in the directory without having to write a custom tool.

security/openssh-portable has the ability to contact an LDAP server to verify SSH keys. This is extremely nice if you have many servers and do not want to copy your public keys across all of them.

B. OpenSSL Certificates for LDAP

If you are hosting two or more LDAP servers, you will probably not want to use self-signed certificates, since each client will have to be configured to work with each certificate. While this is possible, it is not nearly as simple as creating your own certificate authority, and signing your servers' certificates with that.

The steps here are presented as they are with very little attempt at explaining what is going on—further explanation can be found in openssl(1) and its friends.

To create a certificate authority, we simply need a self-signed certificate and key. The steps for this again are

Example B.1. Creating a Certificate

```
% openssl genrsa -out root.key 1024
% openssl req -new -key root.key -out root.csr
% openssl x509 -req -days 1024 -in root.csr -signkey root.key -
out root.crt
```

These will be your root CA key and certificate. You will probably want to encrypt the key and store it in a cool, dry place; anyone with access to it can masquerade as one of your LDAP servers.

Next, using the first two steps above create a key `ldap-server-one.key` and certificate signing request `ldap-server-one.csr` . Once you sign the signing request with `root.key` , you will be able to use `ldap-server-one.*` on your LDAP servers.

Note

Do not forget to use the fully qualified domain name for the "common name" attribute when generating the certificate signing request; otherwise clients will reject a connection with you, and it can be very tricky to diagnose.

To sign the key, use `-CA` and `-CAkey` instead of `-signkey`:

Example B.2. Signing as a Certificate Authority

```
% openssl x509 -req -days 1024 \
-in ldap-server-one.csr -CA root.crt -CAkey root.key \
-out ldap-server-one.crt
```

The resulting file will be the certificate that you can use on your LDAP servers.

Finally, for clients to trust all your servers, distribute `root.crt` (the *certificate*, not the key!) to each client, and specify it in the `TLSCACertificateFile` directive in `ldap.conf`.