Del Elson
Babel Com Australia
http://www.babel.com.au/

Centralised Authentication using OpenLDAP

linux.conf.au 2002

Centralised Authentication using OpenLDAP

Table of Contents

Centralised Authentication using OpenLDAP

AUTHENTICATION ON LINUX USING OPENLDAP

Introduction

This article covers a number of issues with LDAP authentication on Linux. In this article, I have focussed on Red Hat Linux version 7.1 and later (with some comments about earlier revisions), however many of the same principles apply to Debian and other Linux distributions.

Topics

The topics that I will cover here include:

- Introduction to Authentication systems
- Introduction to LDAP
- Installing an OpenLDAP server
- Setting up an OpenLDAP client
- Configuring OpenLDAP
- Migrating to LDAP
- LDAP tools
- Securing OpenLDAP

Centralised Authentication using OpenLDAP

Introduction

Authentication and PAM

Authentication

Authentication is the process whereby a user logging on to a Linux system has their credentials checked before being allowed access. Usually, this means that a user needs to provide a login name, and a password.

Many different programs provide authentication in different ways. For example, the basic UNIX `login` program provides a simple text interface for a user to enter a user ID and password. Graphical login systems such as XDM (or gdm or kdm) provide a different interface. Programs such as `ssh` can authenticate users based on things like RSA or DSA keys, as well as passwords.

Authentication Protocols

There are many different authentication suites or protocols available on Linux today. Like all traditional UNIX systems, Linux is capable of authenticating users against entries in the `/etc/passwd` and `/etc/shadow` files, but it also supports such authentication schemes as Kerberos, RADIUS, and LDAP.

PAM

PAM is a set of libraries provided with most modern Linux distributions, and it is installed by default in Red Hat Linux. The PAM libraries provide a consistent interface to an authentication protocol. An application can use the PAM libraries to allow the use of any authentication protocol within that application, so that if the system administrator wants to change from, for example, /etc/passwd authentication to LDAP, the application does not have to be re-written or recompiled.

PAM requires a PAM module for each authentication system. There are many such PAM modules on the above web site.

User Identification and NSS

Limitations of PAM

Unfortunately, PAM only provides part of the information needed to keep track of users on a Linux system. In addition to being able to check that a user has entered the correct password, a Linux system needs other information such as the user's numeric user ID, their home directory, default shell, etc. This information, which would normally be stored in the /etc/passwd file, can be determined through a system interface known as **NSS**.

NSS

Amongst other things, the NSS system provides a richer set of information about each user than just a login name and password. Traditionally, this was done by using files (e.g., `/etc/passwd'), but other name services (like the Network Information Service (NIS) became popular for providing this information. These were usually hacked into the standard C library.

Later Linux libraries contain a cleaner solution to this problem by using dynamic libraries. This is designed after a method used by Sun Microsystems. Linux follows the Solaris naming convention and calls this service "Name Service Switch" (NSS).

Note that NSS is capable of providing information on other "databases" other than the user and group databases. This includes things like the /etc/hosts file, the /etc/services and /etc/protocol files (listing TCP/IP parameters and name mappings), etc. For the purposes of this tutorial we shall be covering all of these only briefly. For more information see the /etc/nsswitch.conf file on your system.

NSS and authentication protocols

Only some authentication schemes provide enough information to be useful to NSS. For example, Kerberos only stores user authentication information, not details such as a home directory or default shell. Therefore, there is no NSS module for Kerberos.

Other authentication schemes that do not provide information to NSS include most of the one-time password systems such as S/KEY and SecureID.

Issues in Network Authentication

Storage of passwords

Obviously, one of the big issues in network based authentication systems is solving the problem of how the passwords get stored, accessed, and passed across the network.

There are two basic options:

- Store the passwords in plain text on the server.
- Store encrypted passwords on the server.

Storing encrypted passwords

Obviously, of the above methods, storing passwords in some encrypted form on the server would seem to be a better option. There is more to this than meets the eye, however. If passwords are going to be stored in some encrypted manner, then they can either be stored using a as a one-way encryption scheme such as a hash method, or using a two-way encryption scheme such as a block cipher.

If a two way encryption scheme is used, then a theoretical cracker needs only to break into the server, recover the encryption key, and decrypt the passwords. This means that password storage is no more secure than if passwords were stored unencrypted.

If a one way encryption scheme is used, then we need to solve the problem of passing passwords across the network.

Passing passwords

Assume a one-way encryption method (e.g.: a hash) is used to encrypt the users' passwords. The server that is authenticating the users needs to be able to regenerate the one-way hash somehow after receiving the authentication request. This means that it needs the original, unencrypted password sent to it, because this is the only way to regenerate the hash and compare it to the stored hash. So, our basic problem is that if one way encrypted passwords are stored, we need to pass those passwords across the network in clear text. Unless your network is very secure, this is a bad idea.

Attempted (and failed) solutions

There are two basic solutions to the above problem:

1. Pass the hash across the network instead of the unencrypted password (i.e. Perform the hash on the client).
2. Use some kind of challenge - response method based on the hash of the password. This also requires doing the hash on the client.

Obviously, method 1 means that the client need only know the hash of the password, rather than the password itself, in order to fake a login. This means we have essentially reduced ourselves to storing unencrypted passwords, because the hashed password is useful as a "plain text" login. This is essentially what NIS does (although somewhat differently as it passes the hash from server to client).

Method 2 is no better. Because the client is trusted to use the hash of the password rather than the clear text form of the password in generating its half of the challenge/response protocol, the hashed password is essentially as useful (to a maliciously written client program) as the plain text password used to create the hash. So we are again essentially storing plain text passwords on the server. This is what Microsoft does in its NTLM authentication system.

SSL

The only real solution to the above is to allow that plain text versions of the password will be passed across the network, and secure the network somehow. A good method of securing an arbitrary TCP based network protocol (such as one might use in authentication) is SSL, or TLS as it has now become.

SSL basically creates an encrypted "channel" between client and server, and we can now use this channel to pass the clear text password between the client and server. This means we have the security of encrypted password storage combined with the security of no plain text passwords being visible on the network.

Performance Issues

Obviously, in a system that only has a few hundred users, performance of the authentication system is rarely an issue. Where a centralised authentication system is being used on a network with thousands, tens of thousands, or even millions of users, the speed at which an authentication attempt can be done can become an issue.

LDAP servers, because they are reasonably light-weight in comparison to databases, and also because they tend to be highly optimised for read access (which is what is required for an authentication attempt), can provide good performance in these situations. Certainly better performance than NIS, or flat text files.

LDAP

What is LDAP?

LDAP

LDAP (an acronym for *Lightweight Directory Access Protocol*) is a network **protocol** used for accessing information in an object oriented database. LDAP includes features that make it useful to both PAM and NSS, in that it can authenticate users, as well as provide user information such as home directory names and default shells, to NSS.

Note that LDAP does not define a storage method, it simply defines the protocol used to access the data!

LDAP Server

An *LDAP Server* or *Directory Server* (sometimes called a *DS* for short) is a server that can send and receive information in the LDAP protocol. Typically, an LDAP server will be a piece of software that listens on the standard LDAP ports (389 and sometimes 636) for connections, and responds to LDAP queries and requests. To draw an analogy with databases, LDAP is the equivalent of SQL, and an LDAP server is like a database server such as Oracle or MySQL.

What can be stored in an LDAP server?

LDAP servers are particularly useful for storing information about people. This is because of the object oriented nature of LDAP. Unlike a relational database, an object in an LDAP directory can contain an arbitrary number of attributes, and each attribute can have an arbitrary number of values. This is useful for many reasons. For example, a database row containing a column for a phone number would allow a single entry in that phone number column for each row in the database table. A person, however, may have more than one phone number, and so LDAP allows multiple phone numbers to be stored in the same person object.

Object Oriented Terminology

Note the slightly different terminology here: We say an LDAP *directory* as opposed to a *database*, we call entries in the directory *objects* instead of *rows* and we call field values of an object *attributes* instead of columns.

Centralised Authentication using OpenLDAP

Why use LDAP?

LDAP as a storage mechanism

There are a number of reasons why we might use LDAP:

- LDAP allows us to centralise the information about users, passwords, home directories, etc, in a single place on a network. If we were using /etc/passwd files, for example, we would have to make sure that all passwd files were kept in sync across the network, which would be an absolute nightmare on a large network with users changing passwords regularly.
- LDAP offers encrypted transactions. Most LDAP servers offer encrypted connections using SSL (either using Start TLS on port 389 or LDAPS on port 636), which is more secure than some mechanisms whereby plain text passwords are sent over the network.
- An LDAP directory is useful for other purposes. For example, it can quickly and easily be used as a company's staff e-mail and contacts directory.
- It is possible to use LDAP in a tree structured manner, unlike the /etc/passwd or NIS tables which basically store users in a flat structure. With a large number of users it makes sense to divide them into organisational units so that they can be found and managed more easily. In the long term, this makes managing an LDAP directory less onerous than managing /etc/passwd files or an NIS/NIS+ database.

How is LDAP different to a database?

An LDAP server is essentially an object oriented database that has been designed different objectives:

- Because of its object oriented nature, LDAP can store many different types of data in a single "table". This is good for storing system information (as would be required by any NSS backend), because the types of information I need to store about a user might be different to the types of information I need to store about a network service. With a database, storing different types of information would require multiple tables.
- Because LDAP is used for storing data that is **mostly** static, it can be heavily optimised for read access. The design goals of a database are such that a large amount of read/write access occurs, and so databases are usually designed to have good read and write performance. In the design of an LDAP data store, read performance can happily be optimised at the expense of write performance. This usually means that read access to an LDAP server is faster than read access to an RDBMS, although write access will be faster in an RDBMS.

Authentication and File Sharing

Limitations of any authentication system in an NFS environment

One of the reasons we may want to authenticate users around a network in a central manner is to allow them access to a central file store, such as an NFS server.

Solving the authentication problem does not solve any of the issues in NFS security, however.

Unix user to ID mapping

The basic function of UNIX authentication systems are to map a user name (e.g.: "del") to a user number, or numerical user ID, such as "501".

Once this user ID is obtained, the NFS system "trusts" this user. NFS contains no authentication protocol as part of it (although NIS and LDAP are commonly used authentication protocols in an NFS environment).

One issue with this is that if I trust an entire network to make NFS mounts from an NFS server, then a malicious person can plug a machine that they have set up themselves into the network, give themselves a user ID that matches one on my file server, and access files as if they were the user that owned that user ID.

LDAP provides an authentication method which I can use to map user names to numbers on a network, but NFS does not mandate the use of LDAP before making an NFS connection or reading a file on an NFS mount.

Alternatives -- CIFS?

CIFS (or SMB) is Microsoft's file sharing protocol, and it has its own authentication system (NTLM) that is mandated as part of making a CIFS connection to a server.

This means that no matter what client I plug into a CIFS network, I must authenticate to the server before I can access files.

There are several problems with this:

- NTLM, as an authentication protocol, basically sucks. Essentially its challenge/response system based on MD2 hashes reduces the server's password store to one that holds plain text passwords. A cracker that breaks an NT server can obtain the MD2 hash of every user's password, and therefore impersonate any user on the network at any point in time.
- CIFS, as a file sharing protocol, also sucks. I'm not going to go into a detailed explanation of the many reasons for this (ask Andrew Tridgell for that if you have a day or so to spare), but my unfavouritest point about it is that it is TCP socket based, unlike NFS which is UDP based. This means that (for example) SAMBA has to spawn a new process and/or create a new TCP socket end-point to handle every connection. On a large network this becomes impractical.

AFS?

AFS is a distributed file sharing protocol, which has Kerberos as an authentication method within it.

I don't like this very much either, because I don't like Kerberos. It's reasonably secure as a network authentication protocol goes, but the Kerberos server essentially stores a token that is useful as a plain text password for each user, much in the same way that NT does.

A good solution?

There aren't any.

What I'd like is a light-weight, distributed, encrypted, secure, file sharing protocol, that incorporated LDAP as an authentication and user information back end scheme. It would be UDP based (except for the LDAP parts of course which are TCP), incorporate flexible and opportunistic encryption methods (preferably including Rijndael / AES), portable (with clients available for at least Windows NT, 2000, 95, 98, DOS, and UNIX), and of course it would be open source.

Are there a couple of hundred C programmers in the audience that might be prepared to donate me a year or so of their time?

Setting Up OpenLDAP

Installing the OpenLDAP Server

OpenLDAP

OpenLDAP is an open source implementation of an LDAP directory server. OpenLDAP is installed by default with Red Hat 7.1 or later, and is available on Red Hat versions from 6.2 onwards.

Note that earlier releases of Red Hat used release 1 of the OpenLDAP product. Although this is still considered a stable release of OpenLDAP by the OpenLDAP team, I would advise against using it for a number of security reasons (it does not support SSL or schema checking). Your OpenLDAP version should be 2.0.7-3 or later.

Client or Server installation?

Note that OpenLDAP, like most network services, comes in two parts, a client and a server. You usually only need one LDAP server on your network, however a second can be helpful. You will need to install the LDAP client libraries on every machine on your network.

Note that every LDAP server will usually also need to be an LDAP client.

Installing the OpenLDAP Client and Server RPMS

As usual, you can install OpenLDAP from the source code by obtaining the source files from the OpenLDAP web site (http://www.openldap.org/) and following the compilation instructions.

My preference, however, is to install the OpenLDAP packages from the RPM files as follows. Note that you will need to install both the server and client packages if you want to set up an OpenLDAP server. First, put your Red Hat CD-ROM into your CD-ROM drive and use the following commands:

```
  mount /dev/cdrom /mnt/cdrom
  cd /mnt/cdrom/RedHat/RPMS
  rpm -Uhv openldap-2*.rpm openldap-servers-*.rpm
openldap-clients-*.rpm
  umount /mnt/cdrom
```

It's possible that the packages are already installed on your system (to verify this you can do *rpm -q openldap*), and also possible you may hit one or more dependencies in installing the above RPMs. In particular, the openldap packages require the openssl package, and at least the krb5-libs package.

Installing an OpenLDAP client

Installing the OpenLDAP Client RPMS

There are in fact two parts to being an LDAP client. The first part is the OpenLDAP client libraries mentioned in the previous section, which can be installed as follows:

```
mount /dev/cdrom /mnt/cdrom
cd /mnt/cdrom/RedHat/RPMS
rpm -Uhv openldap-2*.rpm openldap-clients-*.rpm
umount /mnt/cdrom
```

LDAP PAM and NSS libraries

Using LDAP will almost certainly require you to install the PAM libraries for LDAP. In Red Hat 6.2 and later, these are packaged in with the nss_ldap package (since the pam_ldap libraries are not much use without the nss_ldap libraries, and vice-versa). These are normally installed by default -- to test this you can do *rpm -q nss_ldap*.

If the nss_ldap package is not installed, you can install it using RPM as follows:

```
mount /dev/cdrom /mnt/cdrom
cd /mnt/cdrom/RedHat/RPMS
rpm -Uhv nss_ldap*.rpm
umount /mnt/cdrom
```

If you need to obtain the source code for the pam_ldap and nss_ldap libraries, they are available from PADL.com at the following locations:

- http://www.padl.com/pam_ldap.html for pam_ldap
- http://www.padl.com/nss_ldap.html for nss_ldap

Configuring OpenLDAP

/etc/openldap/slapd.conf

Configuration of openldap is done through the /etc/openldap/slapd.conf file. There is a manual page describing the contents of the slapd.conf file (see man slapd.conf) as well as an excellent administration guide (http://www.openldap.org/doc/admin/) on the OpenLDAP web site. As a starting point, you might like to use the following simple configuration file:

Sample conf file

```
#
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include
/etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema
include         /etc/openldap/schema/rfc822-
MailMember.schema
include         /etc/openldap/schema/autofs.schema
include
/etc/openldap/schema/kerberosobject.schema

#####################################################
# ldbm database definitions
#####################################################

database        ldbm
suffix          "o=MyCompany,c=AU"
rootdn          "uid=root,ou=People,o=MyCompany,c=AU"
rootpw          secret
directory       /var/lib/ldap
# Indices to maintain
index   objectClass,uid,uidNumber,gidNumber     eq
index   cn,mail,surname,givenname
eq,subinitial

#
# ACLs
#

access to dn=".*,ou=People,o=MyCompany,c=AU"
   attr=userPassword
 by self write
 by dn="uid=root,ou=People,o=MyCompany,c=AU" write
 by * auth

access to dn=".*,o=MyCompany,c=AU"
 by self write
 by dn="uid=root,ou=People,o=MyCompany,c=AU" write
 by * read

access to dn=".*,o=MyCompany,c=AU"
 by * read

defaultaccess read
```

Running OpenLDAP

More on slapd.conf

There are a couple of things that will need noting in the configuration file above:

- Replace "o=MyCompany,c=AU" throughout the file with a *Base DN* which represents your organisation. Note that I prefer to use the X.500 style specification above, but you could use the DNS specification which is "dc=mycompany,dc=com,dc=au" or similar. For example, if your company was called "farnarkle.com" you could use "dc=farnarke,dc=com", or you could use "o=farnarke,c=US". Remember this Base DN, it will be important later.
- I have elected to include some elementary Access Control in the file. The standard slapd.conf file included with Red Hat Linux does not include ACLs, but they are mandatory for real use. You may want to expand on the above ACLs -- see the slapd.conf manual or the administrator's guide.
- I have included a default root password -- *secret* This is a bad idea once you have data in your LDAP directory. We will deal with this later.
- I have not included TLS certificates, keys, or other information. I would consider this to be a security issue on a network, because without these the server will operate entirely in plain text mode. This will be covered later.

Starting OpenLDAP

Once you have a working slapd.conf file, you should be able to start your server. This is easy enough to do, you can just run the following command:

```
/etc/rc.d/init.d/ldap start
```

Provided that the slapd.conf file is correct, you should be able to use *pstree* to see a running slapd process. If the slapd.conf file is incorrect, look for error messages (running *slapd -d* here might help), fix up any problems you see, and try again.

Migrating to OpenLDAP

Introduction

Until now, we have been going through a fairly standard install/configure/start process, the sort of installation and configuration that you might expect with any network service, such as MySQL, Oracle, or PostgreSQL.

What we have now, however, is a working, but empty, LDAP directory. This will only be of use once we have data inside it.

The first thing you need to do is to populate it with data. Data from your existing authentication database should suffice, and OpenLDAP provides an excellent set of migration scripts which can be used to populate the directory.

Using the supplied LDAP tools

OpenLDAP provides a suite of tools to migrate data from your existing NIS or /etc/passwd database into LDAP. If you currently run another authentication scheme such as Kerberos or S/Key, and you are migrating to LDAP, then I'm afraid you are on your own.

In Red Hat Linux 7.1, the migration tools are in /usr/share/openldap/migration/. In Red Hat 6.2 and earlier they were in /usr/lib/openldap/migration/. In either case, open a shell window, change to that directory, and get to work.

First, edit the migrate_common.ph file. Around line 72 you will see a couple of lines like this:

```
$DEFAULT_MAIL_DOMAIN = "babel.com.au";
$DEFAULT_BASE = "o=Babel,c=AU";
```

You will need to edit these two lines, providing your default mail domain, and the *Base DN* that you defined earlier in the slapd.conf file.

Next, it's a simple matter of running the migration tools. This can be done using a simple command, assuming that you are migrating from /etc/passwd files to LDAP:

```
migrate_all_online.sh
```

(make sure that your LDAP server is running before using the above command).

This will ask you for the root DN and password (enter the password *secret* that we defined in the slapd.conf file), and will start to populate your LDAP directory.

Basic LDAP security

Secure the root account

The slapd.conf file that was set up earlier contains an entry for the root DN, as well as a password. This password is used as a fallback password for the root DN in case the entry is not found in the directory. Of course, when you first set up LDAP with an empty database, this entry was required. Now it is not, so you should comment it out or remove it.

It should be the case that your root password was obtained from your /etc/passwd or /etc/shadow file and inserted into LDAP. To test this, repeat the above search, using the *-D* flag of *ldapsearch* to attempt a logon to your LDAP directory, as follows:

```
  ldapsearch -x -D 'uid=root,ou=People,o=MyCompany,c=AU'
-W 'uid=root'
```

ldapsearch will ask you for a password -- enter your root password here, and if everything goes to plan this should work correctly.

Denying password read access

The slapd.conf file I gave earlier disallows read access to the password attribute, by anyone other than the owner or the root user. This makes sense, although it is not a default option of OpenLDAP (I think it should be).

Note that "auth" access is allowed to the password access for all users ("*"). This is required to allow people to authenticate to the LDAP server.

"auth" access allows us to test our password against the password stored in the directory, without being able to read that password. This is handled internally by the LDAP server.

Other ACLs

The ACL system within LDAP is quite complex, and can allow/disallow access to individual attributes within an LDAP object. This is useful, for example, if you have an LDAP client that is an HR system, and you want to allow write access to the "salary" attribute for the payroll officer only, and for users to be able to read but not change their own salary. A sample ACL to do this might be:

```
access to dn=".*,ou=People,o=Babel,c=AU"
   attr=salary
 by self read
 by dn="uid=payroll,ou=People,o=Babel,c=AU" write
 by * none
```

The ACL system is described in more detail in the OpenLDAP administrator's guide.

Looking Inside OpenLDAP

Setting up LDAP queries

Setting up LDAP queries

Having data in your LDAP directory is all very well and good, but at some stage you are going to want to query that data. There are a standard set of command line based LDAP query and management tools provided with OpenLDAP. These include *ldapadd*, *ldapmodify*, and *ldapsearch*. Each of these tools has a *man* page, and you would do well to read these man pages in detail.

The standard configuration file for these tools is */etc/openldap/ldap.conf*. The file format of this file is fairly simple, on a single system it need only contain the following two lines:

```
BASE o=MyCompany,c=AU
HOST 127.0.0.1
```

Remember to substitute your *Base DN* that you defined in the slapd.conf file instead of the *o=MyCompany,c=AU* entry shown above.

On a network, you may have to substitute the IP address of your LDAP server instead of *127.0.0.1* shown above. For those of you who understand LDAP concepts a little better: OpenLDAP doesn't (yet) support SLP or DNS RR based location, so you have to be fairly precise about the location of the server -- either an IP address, a host name from /etc/hosts, or something that can be found in DNS.

Performing a first LDAP search

Once you have done that, you should be able to perform a simple search. You could start by looking for your root user by using the following simple command:

```
ldapsearch –x ’uid=root’
```

You should see an entry fairly similar to this one:

```
version: 2

#
# filter: uid=root
# requesting: ALL
#

# root,People,MyCompany,AU
dn: uid=root,ou=People,o=MyCompany,c=AU
uid: root ... (etc)
```

Now that you have come this far, stop and smile a lot. You have managed to get LDAP working, which is sometimes not an easy task!

Setting up PAM and NSS for LDAP using authconfig

Note for users of old systems

First, a warning: This only works with authconfig on Red Hat versions 7.0 or later. If you are using Red Hat 6.2 or earlier then you will need to read the man pages regarding the configuration files, and edit them all by hand. Note also that there is no /etc/pam.d/system-auth file on Red Hat 6.2.

authconfig

Setting up PAM and NSS for LDAP requires editing a configuration file in /etc/pam.d (*/etc/pam.d/system-auth*), modifying the NSS configuration file which is */etc/nsswitch.conf*, and editing the pam_nss and pam_ldap configuration file which is */etc/ldap.conf*

Fortunately, Red Hat provides a utility which will do this automatically for you, called authconfig. To use this, run *authconfig* from the command line.

Within authconfig, it is a fairly straight forwards exercise. Mark the box labelled "Use LDAP". In the next boxes you should enter a value for the Server (this will be the same IP address you used in */etc/openldap/ldap.conf*), and a Base DN (this will be the same as the Base DN that you specified in */etc/openldap/slapd.conf*)

Test your configuration

Once you have finished that, then everything should be set up. I find that the best way to test this is as follows:

- Firstly, find an account in your LDAP directory, using a command like "ldapsearch -x 'uid=someaccount'. This account should be one that has been copied from your /etc/passwd file that you had on the system before setting up LDAP.
- Check that the account exists, using *finger someaccount*. You should get a response showing the user id, name, and other details.
- Using vi or another text editor, edit your */etc/passwd* file and remove the account.
- Using finger again, check to see if the account still exists. If everything is going correctly, it will -- the account details are now being fetched from LDAP instead of from /etc/passwd

Command Line LDAP Tools

Introduction

OpenLDAP comes with a range of UNIX command-line tools for LDAP directory management. These tools are useful as a reference, and since similar tools are provided with many other commercial LDAP systems, it is a good idea to learn these.

The tools come with comprehensive man pages, however it may also be best to read some LDAP information on the internet and/or an LDAP administration guide in order to learn some of the terminology.

Command line tools

The tools are:

- **ldapsearch** -- for searching for entries in LDAP.
- **ldapadd** -- for adding entries to LDAP.
- **ldapmodify** -- for editing entries in LDAP.
- **ldapdelete** -- for deleting entries from LDAP.
- **ldappasswd** -- for changing an LDAP entry's password.
- **ldapmodrdn** -- for renaming LDAP entries.

Common parameters

These tools mostly take a common set of command line parameters, which include options such as:

- **-v** -- verbose mode.
- **-k** -- use Kerberos authentication
- **-x** -- use simple authentication
- **-c** -- continuous operation (don't abort on error)
- **-f** -- read the information from a file
- **-D** -- specifies the LDAP DN to bind with
- **-W** -- ask for a password for simple authentication
- **-H <URI>** -- use a particular LDAP server, eg: -H ldap://ldap.mycompany.com

Other LDAP tools

Graphical Tools

There is a wide variety of LDAP tools available for free or commercially on the internet. A quick search on freshmeat for LDAP will reveal several. Here are a couple that I quite like:

GQ

GQ is a powerful tool that allows you to look inside your LDAP directory and administer any part of it. GQ allows full control over your LDAP server, including:

- LDAP object browser
- Tools to add/delete/update LDAP entries
- LDAP schema browser
- Tools to build entries from other entries, or set up templates.

GQ allows you to get inside your LDAP directory and see the internal workings of LDAP. GQ is available in source code form from the web site, and the source code includes a .spec file so that you can build an RPM file for it if you prefer.

Directory Administrator

Directory administrator is an application for managing user and group entries in an LDAP directory servers. It provides a friendly interface to manage users' personal details, address book information, and mail routing (for sendmail versions which support mail routing information stored in LDAP.

I find Directory Administrator's interface simple and intuitive, although it is not as powerful as GQ and doesn't go beyond simple user and group management.

The author of Directory Administrator has been promising a new version for some time, and I have a few items on my wish list for it, the main issue being that its main window provides an unsorted and unstructured view of all users in your directory tree. Where there are a large number of users in your LDAP directory, this is unsatisfactory (I have around 1500 in one LDAP tree I manage, and it would not be uncommon for some organisations to have tens of thousands).

As a basic tool for managing user information on a small Linux network with an LDAP directory, however, Directory Administrator is a great tool.

LDAP clients

What is an LDAP client?

An LDAP client is any program that can use LDAP for accessing information.

We have covered the use of an LDAP server as an authentication service, but other common types of access include looking up e-mail addresses and other types of personal information in an LDAP server.

Some companies prefer to call LDAP clients "Directory Enabled Applications". There is a technical term to describe the act of making up complicated sounding names and acronyms to describe a simple concept. The term is "wanking".

Netscape Communicator

Netscape Communicator makes a great read-only LDAP client. Simply type this URI into the location bar:

ldap://myldapserver/uid=root,ou=People,o=Mycompany,c=AU

You will see a result!

Netscape Mail and Mozilla

Both Netscape Mail (from Netscape Communicator) and Mozilla, as well as many other e-mail clients, have the ability to look up e-mail addresses in one or many LDAP servers. Within these applications, the directory servers are set up in the "Preferences" section, or within the address book.

There are few instructions on how to do this but the task is reasonably straight forwards. Simply locate your directory server, inform Netscape or Mozilla of its address, base DN, and any other search criteria, and voila, you have a "directory enabled application".

For other Linux mail clients I'm not sure how lucky you're going to be. The text based ones don't seem to be in a big hurry to support LDAP, although it is starting to appear in more of the graphical ones such as Evolution, etc.

Write your own LDAP client

Language support

Of course there are C libraries for LDAP, but there are also Perl modules (go look in CPAN), PHP support, Python support, etc. There are several web-based graphical LDAP clients either written in Perl or PHP available via freshmeat. Most of the web based mail systems (eg: HORDE/IMP, phpGroupWare, etc) have at least rudimentary LDAP support.

Writing an LDAP client

Of course there is no reason that you cannot extend the LDAP schema for your organisation and start developing your own LDAP clients, either in PHP, Perl, C, or whatever.

LDAP is really useful for storing "arbitrary" information about people. It would make a great data store for an human resources system, or payroll system, for example.

Remember that write access to an LDAP server is generally relatively slow. Information that changes regularly may be best stored outside of LDAP, whereas reasonably static information could be stored within it (although I also subscribe to the principle of orthogonality of information, which means you should really only have one data store for your application).

Portability

One good thing about LDAP is that the protocol is portable between LDAP servers. If you begin developing your application using OpenLDAP, and later decide to run it on NDS, then it should port relatively easily.

For example, GQ (mentioned earlier) makes a great graphical LDAP browser for OpenLDAP, NDS and Active Directory as well.

Advanced Topics

Red Hat Kickstart and OpenLDAP

Building systems with kickstart

Occasionally I like to build systems using Red Hat's Kickstart tool. The latest version of the Red Hat installation system includes some kickstart options which can be used to connect a system to an LDAP based network as a client during system installation time.

Kickstart Options

The options are described in detail in the Red Hat Linux Customization Guide but the main one you need is the "auth" parameter which should read:

```
  auth --enableldap --enableldapauth --ldapserver=<your
ldap server> --basedn=<your base DN>
```

Red Hat also include some information on OpenLDAP in their Red Hat Linux Reference Guide, especially chapter 4. If you are going to use Kerberos authentication with OpenLDAP then you will also need to read chapter 9.

Making OpenLDAP more secure

SSL and TLS

The main way in which you can improve the security of OpenLDAP is to include secure sockets layer (SSL) and transport layer security (TLS) mode in your client/server connection. This encrypts all of the LDAP traffic using the SSL protocol. OpenLDAP version 2.0 and later have the capability to run in SSL and TLS mode using the OpenSSL libraries, although you should really be using a version later than 2.0.7 to get this capability working properly.

Configuration steps

The main steps you need to follow in getting OpenLDAP to work with SSL are:

- Make sure that OpenLDAP is compiled with the OpenSSL libraries.
- Generate SSL keys for OpenLDAP
- Configure OpenLDAP to use the SSL keys
- Test!

SSL and StartTLS

Note that there are two different modes of operating OpenLDAP with SSL. These are:

- TLS, otherwise known as "Start TLS" mode. This is the more modern approach to secure communications, as it uses the same TCP port number to connect to the OpenLDAP server (389) but switches to secure communications using a "Start TLS" command before any data is transferred.
- SSL mode, which operates on a different TCP port number (636) from the standard LDAP port, and begins the connection in secure mode.

TLS mode is more flexible than SSL mode (since clients or servers that do not understand SSL can continue communicating by ignoring the Start TLS command), but unfortunately many older LDAP servers and clients do not implement this mode, and only use SSL mode. Therefore it is preferable, in my opinion, to get your LDAP servers and clients working in both TLS and SSL mode.

Compiling OpenLDAP with OpenSSL

Use the RPMs

Hopefully, this is the easiest part of the job. If you are using a packaged installation of OpenLDAP provided with, for example, Red Hat or Debian Linux, you will find that this has already been done for you. Once again, make sure that you have a recent version of OpenLDAP (from Red Hat, any version that is 2.0.7-3 or later should be fine).

Compiling from Source

If you are compiling OpenLDAP from source, you will simply need to give the following additional flag when running the OpenLDAP configure script:

```
--with-tls
```

You also need to make sure that the OpenSSL libraries are present on your system.

Generating SSL keys for OpenLDAP

Creating PEM format keys

To use OpenLDAP in TLS or SSL mode you will need to generate a PEM format SSL key. Assuming that you have the OpenSSL package installed (otherwise you would not be doing this), you will find a Makefile in the /usr/share/ssl/certs directory that contains the appropriate commands to create this key. You can run these with the following simple command:

```
  cd /usr/share/ssl/certs
  make slapd.pem
```

Key Building

During the key building process, you will be asked for a whole raft of details about your server. This includes the country code, state, organisation name, e-mail address, server name, etc. Answer these questions as best as you are able -- they are encoded in the PEM file.

No make?

If you do not have OpenSSL's Makefile, or you cannot run the "make" program for some reason, then you will need to build the PEM file manually. Here is a rough outline of the commands needed to do this:

```
  /usr/bin/openssl req -newkey rsa:1024 -keyout
tempfile1 -nodes -x509 -days 365 -out tempfile2
  cat tempfile1 > ldap.pem
  echo "" >> ldap.pem
  cat tempfile2 >> ldap.pem
  rm -f tempfile1 tempfile2
```

Self signed keys

Note that this command creates an RSA key, and then self-signs that key. Self-signed keys are not usual in (for example) HTTPS communications, as the keys are generally signed by a third party known as a *Certification Authority*. In this case, however, a self-signed key is perfectly OK as most LDAP clients do not check the signature of the key.

Of course, for more advanced users, it would be entirely possible to generate a key and have it signed by an external Certification Authority. It's up to you.

Important note

Once you have generated the keys, it is important that they are made readable (only) by the user that is running the LDAP server. For Red Hat Linux, OpenLDAP's server (slapd) runs as an unprivileged user called "ldap" in a group called "ldap". To make this user and group the owner of the key that you have just generated, run the following command:

```
  chown ldap.ldap slapd.pem
```

Configuring OpenLDAP to use SSL keys

Modify slapd.conf

To configure OpenLDAP to use the SSL key you have just generated, you need to modify the /etc/openldap/slapd.conf file. The following lines will need to be added to the file. They can be anywhere in the file, but I suggest putting them in close to the top, after all of the "include" statements:

```
TLSCipherSuite HIGH:MEDIUM:+SSLv2
TLSCertificateFile /usr/share/ssl/certs/slapd.pem
TLSCertificateKeyFile /usr/share/ssl/certs/slapd.pem
```

Note that there are different allowable values for the TLSCipherSuite line, but the above line is the one that I recommend.

Modify your startup script

You may also need to modify your LDAP start up script. If you are using Red Hat Linux version 7.0 or later then you don't need to do this. Otherwise, you should locate the line in your startup script (probably /etc/rc.d/init.d/ldap or /etc/init.d/ldap) that contains the line to start slapd, and modify it to look like this:

```
slapd -h '"ldap:/// ldaps:///"'
```

It is the options after "-h" that we are most interested in at this point. You will notice that we are telling the LDAP server to work in both ldap and ldaps (secure) mode. Theoretically, once this is working, you could turn ldap mode off and only use the secure mode, but this is not recommended as many LDAP clients don't support this secure mode. Note that there may be other options on the line that runs slapd in your startup script, and you should leave those intact.

Restart LDAP

You will now need to restart your LDAP server in order for it to re-read the slapd.conf file:

```
/etc/rc.d/init.d/ldap stop
/etc/rc.d/init.d/ldap start
```

Testing OpenLDAP and SSL

netstat

The first step in testing that your LDAP server is listening in SSL mode is to run the following command:

```
netstat -a | grep LISTEN
```

Looking through the output of that command, you should see that your slapd server is listening on port 389 (ldap) as well as port 636 (ldaps). If SSL mode is not working, then it will be listening on port 389 (ldap) only.

Testing the raw connection using openssl

Fortunately, OpenSSL comes with not only a handy key generator, but also an "SSL line tester". It works by running the following command:

```
openssl s_client -connect localhost:636 -showcerts
```

Note that I have elected to use the direct connection to LDAP in SSL mode rather than going through TLS ... the latter is possible, but somewhat trickier.

Output from openssl s_client

If all goes to plan, you should see some output from your LDAP server in the command window, showing information about the certificate, as well as the certificate, session IDs, and master keys, that are in use by the server.

LDAP clients

You could now find an LDAP client that supports SSL or TLS mode, such as GQ, turn on the option in the client that enables TLS mode, and see if it works.

Enabling SSL support in the nss_ldap and pam_ldap modules is relatively simple, just re-run authconfig to state that you want SSL mode. Either that, or modify the /etc/ldap.conf file to include the line:

```
ssl start_tls
```

There are several other tls options in this file, you should ignore those unless you have a specific need to perform certificate verification (in which case you really should know what you are doing and why).

Voila, you now have a secure LDAP server!

Setting up an LDAP replica

Replicas and reliability

Setting up an LDAP replica is an important part of reliability on a network. LDAP replicas allow for load balancing, however the current NSS and PAM clients for Linux don't currently support load balancing and server failover in any intelligent way.

Slapd and slurpd

OpenLDAP only supports a master/slave replica method, not a multi-master method as supported by some LDAP servers. There are two parts to a master LDAP server, these are "slapd", which is the standard LDAP server, and "slurpd", which is the replicator.

Slurpd only runs on a master LDAP server that is forwarding its updates to a slave server.

Creating a "replica" user

Before you begin any replication, use an LDAP client (eg: GQ) to create a user account in LDAP that can be used for replication. Note that it's possible to use the root account for this but generally I would advise against it. I normally call this account "ldapreplica".

Enabling the master

A few extra parameters in the slapd.conf file are required to enable a master server.

```
#
# Replicas
#
replica         host=192.168.1.12:389
  binddn="uid=ldapreplica,ou=People,o=Babel,c=AU"
  bindmethod=simple credentials=somepassword
```

Enabling the slave

A slave server requires a few extra parameters in the slapd.conf file:

```
updatedn "uid=ldapreplica,ou=People,o=Babel,c=AU"
updateref ldap://babel.babel.home
```

The "updatedn" parameter restricts writes to the LDAP database to this user only. This is because we don't want any other user other than our replica user being able to write to the slave directory.

The "updateref" parameter gives a URI to pass back to LDAP clients, essentially telling them to go elsewhere if they do attempt a write.

Slave slapd and ACLs

The ACLs in a slave slapd will need modifying to allow the "ldapreplica" user write access to any part of the directory that needs modification. For example:

```
access to dn=".*,o=Babel,c=AU"
 by dn="uid=ldapreplica,ou=People,o=Babel,c=AU" write
 by * read
```

Using the Replica

ldap.conf

Now that you have a replica on your network, you essentially have two places where you can perform LDAP queries.

The first place to specify these locations is in /etc/openldap/ldap.conf, which is the default configuration file for LDAP clients on your system.

The HOST line in this file can contain multiple entries, like this:

```
HOST 192.168.1.12 192.168.1.2
```

These entries specify a list of servers that can be tried for LDAP queries, in priority order.

NSS and PAM

NSS and PAM use a separate ldap.conf file, located in /etc.

The format of this file currently only allows a single host line to specify the address of the LDAP server.

For load balancing purposes on a large network, you could have servers spread over a number of subnets, and use the /etc/ldap.conf file to point NSS and PAM at the nearest server.

Don't do this

There is a really sneaky way of enabling failover in the nss_ldap. It basically involves compiling up two separate nss_ldap libraries and making them use different ldap.conf files. Of course you also have to give them different names, and fiddle with your /etc/nsswitch.conf file to make it all work.

I am really not going to describe how to do this, it's horrible.

Schema Extensions

LDAP Schema

Earlier, I stated that "an object in an LDAP directory can contain an arbitrary number of attributes, and each attribute can have an arbitrary number of values".

This is essentially correct, except for the fact that an LDAP server allows us to define a schema, which defines the type of attributes for each type of object, and what sort of values those attributes can contain.

Here be dragons

This section is reasonably complex, but fundamental to understanding the way that LDAP works. You can avoid all of this mess by enabling the following directive in your slapd.conf file:

```
schemacheck off
```

However, do not, under any circumstances, do that.

ObjectClasses, and Attributes

Each LDAP object has one or more "objectClass" attributes. Each of these attributes names an object class that is defined in the LDAP schema.

Each object class can define one or more attributes. These attributes can be mandatory, optional, multi-valued, single-valued, and can have a specific "syntax", which basically defines whether the values can be strings, numeric, binary, etc.

The total list of attributes that an object must have is the sum of all of the mandatory attributes of its object classes. Similarly, the list of attributes that an object may have is the sum of all of the optional attributes. Actually, by "sum" here, I mean "union of the set" because an attribute may be defined as mandatory or optional by more than one object class.

ASN.1

Some LDAP servers store the LDAP schema, that is the list of object classes, attributes, syntaxes, etc, in the directory. Some store it in a separate database. OpenLDAP is fairly ugly, and stores it in ASN.1 format in plain text files. It does, however, allow read-only access to this using a special syntax defined within LDAP, and GQ contains a fairly useful schema browser for examining this schema in a more friendly manner.

ASN.1 syntax is very ugly. If you absolutely need to extend the OpenLDAP schema then you will need to learn a bit about it, however that's beyond the scope of this paper.

You can create ASN.1 entries by copying and modifying (carefully) existing entries in the /etc/openldap/schema directory.

Organisational Numbers

Obtaining an organisation number

Each LDAP object class, attribute, syntax, and entry of any kind, must have a unique distinguished number. This is a number of the format "x.x.x.x.x.x.x ..." and so on.

These numbers are assigned, controlled, and issued by the IETF, essentially the same body that makes up numbers in the /etc/services and /etc/protocol files, as well as issues IP addresses, etc.

Each organisation has a unique integer with which to prefix all of their ASN.1 extensions of any protocol. The two most common protocols that use these extensions are LDAP, and SNMP.

I have elected to obtain a single number for my company, and I use this number for all of my clients. The number that I was issued with is 9080.

All schema extensions begin with "1.3.6.1.4.1" and then this number. So, all of my schema extensions begin with "1.3.6.1.4.1.9080".

Extending the organisation number

Within the organisational prefix, you are allowed to do what you want. I have elected to use the next digit as an "extension type" prefix, 1 for SNMP, and 2 for LDAP. This is a fairly common convention.

Within the LDAP extensions, I have decided to use ".1" for attributes, and ".2" for object classes.

So, all of my personally defined LDAP attributes are numbered beginning with "1.3.6.1.4.1.9080.2.1.1" and then "1.3.6.1.4.1.9080.2.1.2" and so on.

I could have elected to break these extensions down by division (eg: product manufacturing, electronics, marketing, etc). I could have elected to break them down by customer, or by country or geographical location. Entirely my own choice.

Extending the Schema

Defining a simple object class and attribute

In order to keep my schema extensions separate, I have put them in a separate file in the /etc/opendlap/schema directory. I call this file "babel.schema".

Here are a couple of entries from it:

```
attributetype
   ( 1.3.6.1.4.1.9080.2.1.1
      NAME 'beverage'
      DESC 'Favourite Drink'
      EQUALITY caseIgnoreIA5Match
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
   )

objectclass ( 1.3.6.1.4.1.9080.2.2.1
        NAME 'drinker'
        DESC 'Person who drinks beverages'
        AUXILIARY
        MAY beverage )
```

This very simply defines a string attribute known as "beverage" which I can optionally attach to any object that contains the objectClass "drinker".

The attribute syntaxes are copied from an existing definition in another schema file, although the full list of these can be obtained on line.

Single vs Multi value attributes

All attributes are multi-valued by default. This means I can include multiple entries in the "beverage" attribute defined above. This is good, because sometimes I drink beer and sometimes I drink cider, so I would prefer to include both.

An attribute can be defined as single valued by including the words "SINGLE-VALUE" after the syntax. Have a look through the existing schema files for some examples of this. For example, the nis.schema file restricts the uidNumber attribute to be single valued -- this makes sense as we don't want Unix users to have multiple UIDs.

Including your schema extensions

Slapd.conf file

The slapd.conf file defines all parameters for the LDAP server.

This file includes a number of schema files via "include" directives at the top of the file. Simply add your schema extension to the slapd.conf file as follows:

```
include /etc/openldap/schema/babel.schema
```

Stop, start, test, restart

In order to get the LDAP server to re-read the slapd.conf file and include your extensions, you must stop and re-start the server:

```
/etc/rc.d/init.d/ldap stop
/etc/rc.d/init.d/ldap start
```

Check that slapd is running again after you attempt this. If not, there may be some syntax errors in your schema extension file. Look for error messages in the system log files, try running slapd with the -d parameter (for debugging), fix the problem, and restart.

Using your schema extensions

Remember that to add an attribute of type "beverage" to an object we must first define the object as being of class "drinker".

To do this, open an object in GQ (for example), and examine the list of values of the objectclass attribute. Add a new value to this list, being "drinker". Save the object, and re-load it. You will now see an entry for the "beverage" attribute, in which you can enter one or more values.

This is possible because GQ is clever enough to understand the LDAP schema for each directory, and knows to put entry fields in for any attribute that is allowed based on the list of objectclasses.

Other applications

Your next task is to write an LDAP client (or "directory enabled application" if you prefer) to use this attribute.

Search for all objects in the directory within the current location that have the objectclass "drinker". Obtain the list of "beverages" and compare this to a list obtained by performing a MySQL database call to the refrigerator (your fridge does have an IP address these days, does it not?). If a match is found in the directory that is not in the refrigerator, then an XML/RPC call to the on-line ordering system of the nearest provider of comestibles should suffice. If the transaction fails, then you will have to interpolate the HTML code issued by your bank's on-line banking system (you may have to crank up your SOAP libraries instead and obtain this via XML over HTTP), transfer extra funds into your credit card account , and try again.

Or whatever.

Issues with OpenLDAP

OpenLDAP in comparison to other LDAP servers

As much as I like free software, I have found a number of issues in the time that I have been using OpenLDAP:

- OpenLDAP, in comparison to some LDAP servers such as iPlanet DS and Novell's NDS, is quite slow. This is particularly noticeable when both read and write operations are happening at the same time.
- I have discovered at various points a number of bugs in the OpenLDAP servers and libraries. These cause several issues, including things like having NSS lookups fail on occasion under load, having SSL transactions terminate unexpectedly, and having directory recovery problems after a system crash. To their credit, the OpenLDAP team maintain an active and publicly accessible bug tracking system where the status of such issues can be tracked and monitored. On occasion I have managed to repair certain bugs by reverting to previous versions of the NSS LDAP libraries.

Alternatives to OpenLDAP

As alternatives to OpenLDAP, both Novell and iPlanet have (non-free, non-open source) directory servers available on Linux. Both of these work fine with the standard LDAP NSS and PAM libraries from PADL, although Novell provide their own PAM and NSS libraries (for a price).

Novell provide some tools to load appropriate schema extensions into the NDS product to allow its use as an NSS server. These come with the "NDS Enterprise Edition", whereas the basic NDS (the one that now costs $US2 per user) comes only with sufficient schema details for use as an authentication but not NSS server.

"Other" OS alternatives

Windows 2000 server contains an LDAP server implementation. You've probably heard of it, it's called "Active Directory".

Active Directory contains some weird and non-standard (i.e. Non-IETF approved) schema extensions, although with an appropriately recent nss_ldap module and some fiddling with the /etc/ldap.conf file, Active Directory can be used in Linux as an authentication and NSS service. To use Active Directory as an NSS service you will need to load some schema extensions, and a few DLLs into the Active Directory server to manage these extensions.

How to do that is well beyond the scope of this tutorial.