Linux Authentication: Part 3

THE LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL

In the final article in this series Bruce Richardson widens the scope. As well as acting as a password database, LDAP can also store a huge range of user and network information

What is a directory service?

A directory service is a specialised database used to store information in a freeform, flexible heirarchy. It can hold any kind of information but network/Internet directories typically hold information on users or network resources.

Directories differ from standard databases in that they are optimised for fast information retrieval rather than robust data storage or bulletproof transactional updates. You wouldn't use a directory service to store your financial records but you might well use one to store your company address book. Any information that isn't constantly updated but is frequently looked up is a possible candidate for storage in a directory.

Novell's NDS and Microsoft's Active Directory are two examples of commercial, proprietary directory services.

What is LDAP?

LDAP is a protocol for accessing directory services over a TCP/IP network. It was originally designed to be a lightweight front-end to the much grander and more complex X.500 directory access protocol. Since the X.500 protocol is complex, difficult to implement and runs over the little-used OSI network protocol stack, early adaptors of LDAP found they were better off using LDAP on its own as a front-end to simpler datastores, which is how LDAP is now most commonly used.

Why should I use LDAP?

- LDAP offers a way to centralise information on all your network resources, greatly reducing administrative overheads even if you run a mixture of operating systems.
- It's an Open standard.
- Disparate applications, which normally each have their own datastores, can share information, eliminating duplication and a potential source of error.

Getting OpenLDAP

The source for the OpenLDAP server and utilities are available from the main site at *http://www.openldap.org/software/download*. At the time of writing the latest version is 2.0.23. Be warned, though, that the 2.x version of OpenLDAP has a greatly increased set of dependencies, mostly for the secure authentication methods required for version 3 of the LDAP protocol. We recommend using the versions packaged with your distribution unless you have specialist requirements. At a bare minimum you need the package containing the slapd daemon, which stores the directory information.

The OpenLDAP suite

The OpenLDAP project maintains and develops a suite of software for maintaining and querying LDAP servers. It is the only practical Open Source implementation currently available (the Michigan University version is a reference implementation only and not actively maintained) and will be used for all the examples in this article.

This article cannot hope to cover the whole vast topic that is LDAP. After an overview of the structure of LDAP Directories it will show you how to place data into an OpenLDAP directory and make some basic use of it.

LDAP objects

There are two parts to an LDAP datastore: the schema, which defines what kind of objects may be stored in the directory, and the database, which contains a record for each object stored.

The schema defines what types of objects may be stored in the directory. For each object it defines a set of attributes – some of which are compulsory, some optional. The schema defines how many instances of each attribute an object may have and the properties for each attribute (is it case sensitive, what format may its data take, etc.) This article will not examine the details of schemas (you will normally never have to edit your own schema files) but it is useful to know of their existence because you can extend the capabilities of your LDAP directory by including new schemas. Nor will this article enumerate the LDAP objects found in the core schemas, or their various attributes. Hopefully the examples given will be enough to give you the general idea.

LDAP heirarchy

Information in an LDAP directory is organised into a heirarchical tree structure in much the same way as

your computer's filesystem is organised. It starts with a root node (or "suffix"), to which a number of nodes are appended. These nodes in turn may contain sub-nodes and so on. Each node is represented by an object in the datastore.

The root node, for example, might be represented by an "organization" object ("o"), while the subnodes are most often organizationalUnit objects (ou). A node is named for its position in the heirarchy, starting with the least significant name. So the Returns department within the Sales department within the Example company would be named "ou=Returns,ou=Sales,o=Example".

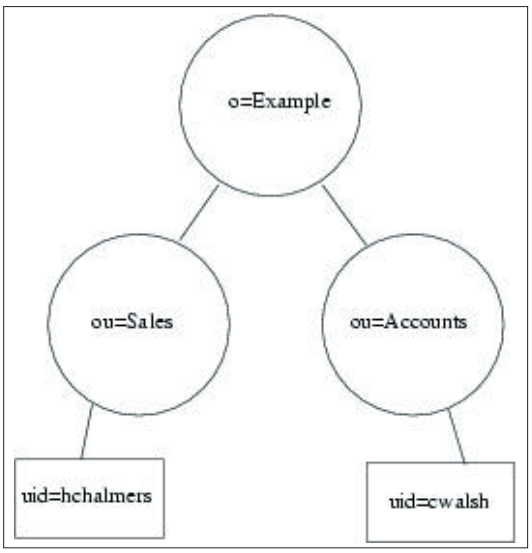

Figure 1: Our example heirarchy

The distinguished name

Any database needs a way of uniquely identifying each record. LDAP objects use their dn attribute, where dn stands for *D*istinguished *N*ame. The dn is constituted from the path to the tree node where the object is located and an attribute that uniquely distinguishes the object from all other objects in that node. This attribute is referred to as the rdn (*R*elative *D*istinguished *N*ame) and is often the cn (*C*anonical *N*ame) or uid (login id) of the person in question. So the dn of Harry Chalmers, who works in the Sales department of the Example organisation, might be "cn=Harry Chalmers,ou=Sales,o=Example" or "uid=hchalmers,ou=Sales,o=Example". The dn of the Sales department itself is "ou=Sales,o=Example".

The LDAP protocol

The LDAP standard defines a communications protocol. It's not at all concerned with how a directory service actually stores its information. Current LDAP implementations use a wide range of datastores, ranging from flatfile text databases to fully-fledged SQL database servers.

The slapd.conf file

```
#################
# Global settings

# Schema and objectClass definitions
include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/cosine.schema
include         /etc/ldap/schema/nis.schema
include         /etc/ldap/schema/inetorgperson.schema

# Schema check allows for forcing entries to
# match schemas for their objectClasses's
schemacheck     on

pidfile         /var/run/slapd.pid
argsfile        /var/run/slapd.args
replogfile      /var/lib/ldap/replog
loglevel        0

##########################
# Database backend settings

# The backend type, ldbm, is the default standard
database        ldbm

# The base of your directory
suffix          "o=Example"

# Where the database file are physically stored
directory       "/var/lib/ldap"

# Indexing options
index objectClass eq

# Save the time that the entry gets modified
lastmod on

rootdn = "uid=sysadmin,ou=People,o=Example"
rootpw = "notverysecure"

###################
# Access permissions

# The userPassword by default can be changed
# by the entry owning it if they are authenticated.
Others should not be able to see it, except the
# admin entry below
access to attribute=userPassword
        by dn="" write
        by anonymous auth
        by self write
        by * none

# The admin dn has full write access
access to *
        by dn="uid=sysadmin,ou=people,o=example" write
        by * read
```

The people unit

What happens though if Harry moves to the Accounts section? LDAP objects can't be renamed, so his old entry would have to be deleted and a new one with the new dn would have to be created. This might of course have unwanted side effects. To avoid these it's usual to create a notional "People" or "Staff" organisational unit, put all the staff in it and use that ou in the dn. LDAP objects can be in more than one ou, so you can still reflect your organisational structure in the directory. With this scheme, Harry's dn is always "uid=hchalmers,ou=People,o=Example" no matter how many times he moves within the company.

Multiple back-end databases

```
##########################
# Database back-end settings

database        password
suffix          "ou=people,o=example"
file            "/etc/passwd"

database        ldbm
suffix          "o=example"
directory       "/var/lib/ldap"
```

Example structure

This article will show the creation of an LDAP directory for the Example organisation, whose structure is shown in Figure 1. As you can see, it's a very simple organisation with only two departments (though we will add the notional "People" unit) and two members of staff.

Configuring the server

First of all you'll need to get the OpenLDAP server – see the "Getting OpenLDAP" boxout. Once installed you should edit the slapd.conf config file. This will usually be located somewhere like */etc/ldap/*. A simple example can be seen in the sidebar. Simple it may be, but most OpenLDAP installations will not need anything more complex than this.

The slapd.conf file is divided into two parts. The first contains global settings for the server and the second contains settings for each of the various databases amongst which the administrator chooses to divide the directory information. The access control settings, which look ostensibly like a third block, can be global or back-end-specific.

Global settings

The first block of settings import schema definitions needed for a typical range of storage tasks. The

third block are administrative settings, which you will normally never need to worry about or alter. The "schemacheck on" setting makes the server reject records that don't match the defined schemas. This may be turned off for a small performance gain but if you subsequently enter bogus records this can cause indexing problems and a dramatic slowdown.

Database section

The example database configuration is very simple. It specifies one back-end, using a traditional *nix dbm hash-database system. This backend contains the whole directory. It is possible, however, to split the directory information across multiple back-ends of differing types. All that is needed is to add entries for each back-end, specifying the tree node from which each back-end starts. In the config shown in the "Multiple back-end databases" sidebar, the "people" organisation unit information is retrieved from the */etc/passwd* file, while other information is stored in the dbm database.

The order in which back-ends are allocated is significant. When a back-end is assigned a suffix it is assumed to include that node and all subnodes which have not already been assigned. In other words, when doing a lookup the server goes through the list of back-ends in the order they are defined until it finds one that includes the part of the directory tree it is looking for, at which point it looks no further. So if the order in which the two back-ends in the sidebar are defined were reversed, the password database would never be used.

The rootdn and rootpw settings together define the name and password of a user who may administer the database remotely, even if there is no actual entry for that user in the directory. This is a quick hack to do some of the initial setup. These settings should be deleted as soon as a proper entry for the sysadmin user, complete with password, has been placed in the directory.

In addition to the password and dbm datatabases shown already, OpenLDAP can retrieve information from SQL database servers or arbitrary shell scripts.

Access control settings

Access control settings can be part of the global or back-end-specific settings. Back-end settings override global ones for their specific section of the directory hierarchy.

There isn't space here to go into the structure of OpenLDAP access settings. The first rule in the example allows any user to log in or to change their own password, while the second sets default access, giving full rights to an admin user and read-only rights to anyone else.

example.ldif

```
dn: o=Example
o: Example
objectclass: top
objectclass: organization

dn: ou=People,o=Example
ou: People
objectclass: top
objectclass: organizationalUnit

dn: ou=Sales,o=Example
ou: Sales
objectclass: top
objectclass: organizationalUnit

dn: ou=Accounts,o=Example
ou: Accounts
objectclass: top
objectclass: organizationalUnit

dn:
uid=hchalmers,ou=People,o=Example
cn: Harry Chalmers
givenname: Harry
sn: Chalmers
mail: hchalmers@example.com
uid: hchalmers
userPassword: default
ou: People
ou: Sales
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson

dn:
uid=cwalsh,ou=People,o=Example
cn: Carrie Walsh
givenname: Carrie
sn: Walsh
mail: cwalsh@example.com
uid: cwalsh
userPassword: default
ou: People
ou: Accounts
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
```

Extending the server

LDAP directories are intended to be easily extensible and the OpenLDAP server makes this simple. If, for example, you wanted to use this server to store Netscape Roaming Profiles then all you need to do is include the Netscape schema and add an appropriate access configuration line.

Enabling your changes

Once you have configured your installation the way you want it, restart the server. From the command line this is as simple as:

```
# /etc/init.d/slapd restart
```

Adding records to the directory

LDIF (the *D*ata *I*nterchange *F*ormat) is a standard format for representing LDAP data, which is guaranteed to work no matter what the actual datastore back-end. An LDIF representation of the Example organisation is shown in the sidebar example.ldif. The records must be entered in sequence so that no object is inserted before an object that it relies upon. If you look at the records you can see the Distinguished Name attributes that uniquely identify each object, the attributes that are used to store personal and system information about the two staff members and the objectclass properties, which identify the object type, according to the directory schema.

Info

OpenLDAP homepage
http://www.openldap.org/
Linux LDAP HOWTO
http://www.linuxdoc.org/HOWTO/LDAP-HOWTO.html
LDAP/PostgreSQL HOWTO
http://www.samse.fr/GPL/ldap_pg/HOWTO
LDAP to DNS Gateway
http://ldap2dns.tiscover.com/

Experienced data mungers will note that this is highly structured data, which can be seen in *example.ldif*, can easily be generated through scripts.

Entering the data

If logged in as root at the LDAP server, you can use the *slapadd* tool to insert the data directly. In this case you should shut down the server and run:

```
# slapadd -l example.ldif
```

From a remote computer you can use the *ldapadd* tool:

```
# ldapadd -D
“uid=sysadmin,ou=People,o=Example” -h ldaphost
-f example.ldif
```

Finally, you can use the *ldappasswd* tool to give them some proper passwords.

Using the data

Now that you have the data in the directory, you can make use of it. The *ldapsearch* tool can be used to query the server and extract data from a specific set of records. Running a command line like this:

```
$ ldapsearch -x -H ldap://ldaphost:389/ -b
“o=Example” “ou=Accounts” givenname sn mail
```

should return the names and email addresses of everybody in the Accounts department.

All that effort for that?

That's just for starters and even then it's pretty useful. Pipe the output of that command through an awk or perl filter and you can feed the result to Mutt's query_command address lookup function. Or you can simply point Netscape (or even Outlook Express!) at the ldap server to have a ready-made internal address book.

Pam_ldap config

```
# Your LDAP server. Must be
resolvable without using LDAP.
host ldaphost

# The distinguished name of the
search base.
base o=Example

# The distinguished name to bind to
the server with
# if the effective user ID is
root. Password is
stored in /etc/ldap.secret (mode
600)
rootbinddn
uid=sysadmin,ou=People,o=Example

# Do not hash the password at all;
presume
the directory server will do it, if
# necessary. This is the default.
pam_password exop
```

Further uses for the directory

- Password database – The pam_ldap config sidebar shows a sample config file for the pam_ldap module. Properly configured, this module can be used to add ldap-based password authentication to any pam-enabled application, which you will be an expert at, having read the first article in this series.
- Network Resources – The LDAP Name Service Switch module allows your systems to look up a range of traditional *nix networking information including group membership, hostnames and mail aliases.
- Mail delivery – Most of the popular Open Source Mail Transport Agents, such as Exim, Postfix and Qmail, can be configured to do LDAP lookups to make mail delivery or routing decisions.
- Netscape Roaming Profiles – With only the smallest modifications to a standard OpenLDAP installation you can use an LDAP server to store users' Netscape browser preferences (bookmarks and so on). Instructions for this can be found in the Linux LDAP HOWTO (see the Info boxout at the end of the article).
- DNS Back-end – The ldap2dns utility creates DNS records directly from an LDAP Directory. It can be used with both Bind and djbdns to eliminate the admin tasks of flat-file editing, zone-file editing and all the clunkiness of maintaining a distributed DNS set-up.
- Gateway to traditional services – One of the database back-ends that OpenLDAP recognises is “shell”, in which a shell script is run, returning data from an external process in a format that slapd can serve up as LDAP information. There are sample scripts on the OpenLDAP site that can be used to act as gateways to standard *nix daemons like fingerd but you can go further than that. Once you've learned the format for returning data then anything you can pull out of a script can be served up as LDAP information.

Don't stop there

You now have your directory running, a powerful set of command line tools and a simple yet powerful data description language with which to manipulate and maintain it. The only limit to the uses LDAP can serve within your network is your imagination.

Summary

LDAP can be used to centralise the administration of a huge variety of network tasks. With careful planning all your network resources can be described and configured in one place, and because it's a popular Open standard it can be used to link all your network operating systems. If you aren't already using it it's time to ask yourself why.