Security with LDAP

Andrew Findlay

Skills 1st Ltd

andrew.findlay@skills-1st.co.uk

Network Information Services

When managing a large number of computers it is convenient to store configuration data in a central location rather than maintain separate files on each machine. Thus, DNS quickly replaced large `/etc/hosts` files on the Internet, and a number of more general nameservices were developed to serve more local needs. Sun's *YP/NIS* is probably the best known, but *Hesiod[1]* does a similar job and protocol suites such as *XNS* and *NetBIOS* also include nameservices of one sort or another.

The more general nameservices support a number of 'maps', each of which translates from keys to values. A NIS serving Unix machines has maps such as passwd, group, automount, services, and rpc. Often there are two or more key fields in a map: the passwd map for example has to service lookups both by username and by uid. Data is accessed with the same library calls that originally handled local files, so most application programs are not aware that the underlying storage has changed.

A network using a NIS relies on it completely for normal operation so maintaining its integrity is vital. Most NIS systems in use today have very little protection against active attackers: they depend on the IP address of the server being widely known and assume that nobody has taken over its address. This is obviously insecure in a world where TCP connections can be hijacked at a distance, so system designers are turning to cryptography to provide assurance that good data is being used.

Network Authentication Service

Authenticating users is one of the more visible aspects of computer security. People are used to providing usernames and passwords when they want to use a machine or other resource.

One approach to authentication in a network is simply to use the NIS to access conventional Unix password hashes and to do the validation locally on the desktop machine. This scheme is

commonly used with *YP/NIS* but it suffers from having to make all password hashes available to all desktop machines. There are shadow-password schemes that prevent 'ordinary' users from getting hold of the hash data, but these are fairly easy to bypass when used with a NIS. With access to a collection of password hashes, a cracker can mount a dictionary attack with a good chance of success so it would be better to keep the hashes away from client machines entirely.

Another approach is to use a special-purpose authentication protocol such as RADIUS or TACACS+. These were originally developed to control access to modem banks, but are equally usable with desktop computers. Kerberos is another way to do authentication without exposing secrets: this also has the benefit of providing a complete single-sign-on environment if it is fully implemented. Several other protocols have been pressed into service for authentication over the years: SMB allows authentication against an NT domain, and I have even heard of POP being used in this way! It comes as no surprise to find that LDAP is now used an authentication protocol as well.

Ideally, a network authentication service should provide reliable answers, be fast, and should work without exposing any information about the password. This is harder than it first looks, as the client does not want to expose the password to the server in case the server turns out to be compromised. Similarly, the server should not expose any hash data to the client. Network snoopers should not be able to gather any data that can be replayed later to gain unauthorised access. Each protocol tackles these problems in different ways, with varying degrees of success. LDAP has a number of authentication and security options which can provide very good security if used correctly, though as ever there is a trade-off between security and ease of management.

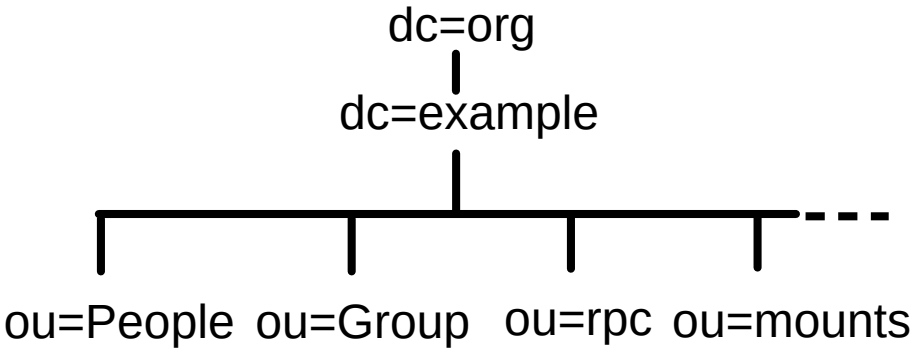
LDAP Basics

LDAP[6] is a directory-access protocol derived from X.500. It works with a tree structure where each node or object in the tree contains a set of attribute-value data. Each object belongs to one or more objectclasses, which define the mandatory and optional attributes. The original application of both X.500 and LDAP was to provide a 'white pages' directory service, where most objects in the tree represented people and the tree had a geographic or organisational structure.

Storing NIS Data in an LDAP Directory

RFC2307[2] describes a schema for storing NIS data in an LDAP directory. Mappings are provided for all the common databases: passwd, group, hosts, shadow, services, netgroup, protocol, ethers

etc. The RFC describes objectclasses and attributes, but does not mandate a particular tree structure. A common practice is to use a structure like this:



Here, domain-component naming is used to match the domain name *example.org* and separate subtrees are provided for each type of information. Data from the passwd file goes into ou=People rather than ou=passwd because it is assumed that each person has one primary account and that other information will be added to the same entry (phone number, e-mail address etc).

Passwd data

Typical data in an account entry looks like this:

```
dn: uid=andrew,ou=People,dc=example,dc=org
uid: andrew
cn: Andrew Findlay
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
shadowLastChange: 11296
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
uidNumber: 500
gidNumber: 500
homeDirectory: /home/andrew
gecos: Andrew Findlay
userPassword: {crypt}$1$uQSw.ohy$XuiRSCq0kp...
```

The first line sets the Distinguished Name - the unique name identifying this entry in the tree. *uid* is used as the naming attribute: this refers to the User ID that we would normally call 'username' in a Unix context. There is one aspect of *uid* to be particularly aware of: it is matched in a case-insensitive way so it is not possible to represent distinct accounts 'fred' and 'FRED'.

Notice that the *uid* attribute occurs again inside the entry, along with *uidNumber* and *gidNumber*: there must be exactly one of each of these attributes and they are used when resolving calls to *getpwnam()* etc.

The full name of the account owner appears twice: first as *cn* (Common Name) which is an attribute that can take multiple values and is much used by white-pages lookups. The extra copy in the *gecos* attribute allows control over the exact string returned to *getpwnam()* calls.

The *objectClass* attributes refer to aspects of the RFC2307 schema. They set the list of mandatory and optional attributes for the entry, and are also useful in searches. A call of the form *getpwnam("andrew")* is likely to result in an LDAP search of the form `(uid=andrew) AND (objectClass=posixAccount)`

Group data

As we have seen, data from passwd and shadow files maps line-by-line into the account entry. Data from the group file is handled in a similar way. A conventional Unix group file has one group per line, with every group member listed:

```
ldapbods:x:389:tim,steve,colin,damy,andrew
```

This is convenient for seeing who is in a group, but extremely inefficient for the most common operation: listing the groups that a particular user is a member of at login time. The usual algorithm is to enumerate every entry in the group map looking for occurrences of the username.

RFC2307 defines group membership using the *memberUid* attribute in a *posixGroup* object. Because *memberUid* is a multi-valued attribute, the group map can now be searched very efficiently. The example group entry above would look like this in the directory:

```
dn: cn=ldapbods,ou=Group,dc=example,dc=org
objectClass: posixGroup
objectClass: top
cn: ldapbods
userPassword: {crypt}x
gidNumber: 389
memberUid: tim
memberUid: steve
memberUid: colin
memberUid: damy
memberUid: andrew
```

Now, to find the list of groups that a user belongs to, the search is of the form:

```
(objectClass=posixGroup) AND (memberUid=andrew)
```

The LDAP server can be configured with indexes on any attribute, so searches of this form can be made very efficient and large databases can be handled without performance penalties.

Putting it all together

So much for the theory, but how much of this actually works? To find out, I built a testbed on a portable PC running Red Hat 7.1 and supporting several VMware virtual machines with diverse operating systems. I also connected a Sun Ultra 5 running Solaris 8.

The LDAP server was *slapd* from OpenLDAP 2.0.18 (a security advisory has since been issued requiring upgrade to 2.0.21). The config file is shown in Appendix 1. slapd sends logging data to syslog, and it is a good idea to separate this into a file of its own with a line like this in `/etc/syslog.conf`:

```
local4.*                /var/log/slapd.log
```

I find it convenient to keep a wide window on screen running '`tail -f`' on the logfile while testing.

To create the initial data I used the migration tools from PADL software. These are often included with Linux distributions, but it is worth fetching the latest version from www.padl.com The tools need very little configuration, and will load data from existing files in `/etc` into the LDAP directory. Note that the tools create the container entries *ou=People* etc but they do not create the top-level entry (*dc=example,dc=org* in this case) so it must be created by hand before the tools will run. To do this, create a file of the form:

```
dn: dc=example,dc=org
objectClass: dcObject
objectClass: organization
o: The Example Organisation
dc: example
```

and load it with a command like this:

```
ldapmodify -a -H ldap://localhost/ -r -c -x \
    -D 'cn=Manager,dc=example,dc=org' -W \
    -f <filename>
```

At this stage, the LDAP server can be tested with command-line tools such as *ldapsearch*. Once it is working it is possible to configure systems to use it as a NIS.

The Name Service Switch and nss_ldap

NSS is a facility found in Solaris and Linux which allows new name-resolution code to be installed as shared objects without having to rebuild existing libraries. Some versions of FreeBSD and AIX have a similar concept called the Information Retrieval System, apparently derived from BIND 8 code.

In principle you can install a new name-resolution system just by placing a shared library in the right directory and editing

/etc/nsswitch.conf. Solaris and many Linux distributions come with LDAP modules for NSS, though they are often derived from fairly old code and you may need to fetch the latest version from www.padl.com if you want all the security features.

I tested nss_ldap on a virtual machine with a small Red Hat 7.2 installation. The first thing to do is to configure the LDAP parameters in `/etc/ldap.conf`:

```
# ldap.conf
#
# Location of LDAP server.
# Must be resolvable without using LDAP!
#
host brick.skills-1st.co.uk
#
# The distinguished name of the search base.
#
base dc=example,dc=org
#
```

This simple config is enough to get things going, though we will return to add more security later.

To make the machine use LDAP for passwd and group lookups, edit `/etc/nsswitch.conf` and change lines as shown:

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

The effect of this is that lookups will use local files first and progress to LDAP for anything not found locally. This allows for system accounts to be kept in /etc files (necessary during bootup) while user accounts are held centrally.

If the name service cache daemon (nscd) is running, it may be necessary to restart it before LDAP lookups work. Similarly, nsswitch.conf is normally only read once per process so don't expect any existing processes to notice until they are restarted (and that includes the shell that you used to set all this up from!)

An easy test is to find the home-directory of a user that is listed in the LDAP store but not in the local password file:

```
echo ~fred
```

The Solaris version of nss_ldap is based on the iPlanet LDAP libraries and it does not use /etc/ldap.conf, preferring instead to use a program called *ldapclient* to set up the parameters. The problem with *ldapclient* is that it depends on a feature that is implemented differently in different LDAP servers so it does not easily work with OpenLDAP. There are a number of mailing-list archives containing discussion of the problem and suggesting various workarounds, though I have not yet made it work myself.

FreeBSD is supposed to be able to use the PADL nss_ldap code, but it requires a rebuild of at least the main C library to make it work. Again, I have not had time to test this, but in principle it should work with the same data that supports Linux.

Authentication with pam_ldap

The usual way of using LDAP for user authentication is to locate the account entry in the directory and then try to 'bind' to the directory as that account, presenting the password or other credentials as received from the user. In the simplest case, the password is presented to the directory server in clear, but there are a number of more secure schemes.

Most Unix-like systems now implement Pluggable Authentication Modules (PAMs) of some form. This allows authentication methods to be changed easily without having to re-compile every program that might need to check a password. PAMs can also implement policy such as time-of-day restrictions and other security requirements. A particularly powerful feature is that PAMs can be stacked, allowing for a choice of authentication methods and perhaps several different policy modules to be active as well. As with NSS, PAM is implemented with shared libraries so re-configuration is quite easy.

Red Hat, SuSE, Mandrake, and FreeBSD all provide pam_ldap modules derived from the PADL code. Solaris includes a module based on iPlanet code, though it is quite possible to replace this with the PADL version if required. In all cases, configuration is shared with the equivalent NSS module so little extra work is required.

To use LDAP for authentication is a matter of installing the pam_ldap module and editing the appropriate PAM config: either `/etc/pam.conf` or one or more files in `/etc/pam.d`. Look for the line describing pam_unix and add pam_ldap at that point. On Red Hat Linux the file is `/etc/pam.d/system-auth` and it looks like this:

```
auth      required    /lib/security/pam_env.so
auth  sufficient /lib/security/pam_unix.so likeauth nullok
auth   sufficient  /lib/security/pam_ldap.so use_first_pass
auth      required    /lib/security/pam_deny.so
account   required    /lib/security/pam_unix.so
account   required    /lib/security/pam_ldap.so

password  required    /lib/security/pam_cracklib.so \
                                retry=3 type=
password  sufficient  /lib/security/pam_unix.so nullok \
                                use_authtok md5 shadow
password sufficient /lib/security/pam_ldap.so use_authtok
password  required    /lib/security/pam_deny.so

session   required    /lib/security/pam_limits.so
session   required    /lib/security/pam_unix.so
session   optional    /lib/security/pam_ldap.so
```

Note that you should not edit the file directly on Red Hat Linux: use the *authconfig* program to maintain it.

Note also that the example config files supplied with the *pam_ldap* source code might give LDAP more control than you want, including the ability to supply an alternate password for *root*.

Solaris uses /etc/pam.conf so the format is slightly different. Here is a snippet describing the authentication requirements for *rlogin*:

```
rlogin auth sufficient \
             /usr/lib/security/$ISA/pam_ldap.so.1
rlogin auth sufficient \
            /usr/lib/security/$ISA/pam_rhosts_auth.so.1
rlogin auth required \
            /usr/lib/security/$ISA/pam_unix.so.1
```

Note the $ISA variable: this is the Instruction Set Architecture, which allows for 32-bit and 64-bit programs to co-exist. Strictly this means that if you replace any PAM modules you should provide both 32-bit and 64-bit versions but so far I have not found any 64-bit programs using authentication This will probably change...

FreeBSD also uses /etc/pam.conf but has a slightly different layout. Here is part of the section for login (and thus rlogin too):

```
login auth    sufficient  pam_skey.so
login auth    requisite   pam_cleartext_pass_ok.so
login auth    sufficient  /usr/local/lib/pam_ldap.so \
                                    try_first_pass
login auth    required    pam_unix.so  try_first_pass
```

Where a PAM password clause has been defined (as in the Red Hat example above), pam_ldap supports changing passwords using LDAP so the user need never know that LDAP is being used rather than local files.

With both nss_ldap and pam_ldap in place on client machines, all user-management can be done through the directory.

Now add Windows: SAMBA PDC

Samba has had the ability to act as an NT-style Primary Domain Controller for some time. Recent releases have added a facility to store the PDC SAM data in an LDAP directory. This is clearly marked as experimental code in Samba 2.2.2 though the Samba-TNG project claims to be more advanced in this area. The two versions have different approaches: mainstream Samba aims to provide Windows-Unix integration, where Samba-TNG aims for a complete PDC implementation that just happens to run on Unix rather than Windows.

If Samba is installed on a machine using *nss_ldap* and *pam_ldap* it will of course use them, so networks running SMB with cleartext passwords may not need to do more than that. There are

advantages to the PDC model though, so Samba's move to closer integration with LDAP is particularly welcome.

To enable LDAP storage for the PDC, Samba must be configured with the ––with–ldapsam option. This completely overrides local storage of SAM data so you cannot use the same binary in both modes.

The LDAP-specific parts of the Samba config file look like this:

```
ldap admin dn = cn=manager,dc=example,dc=org
ldap server = brick.skills-1st.co.uk
ldap suffix = dc=example,dc=org
```

A minimal Samba PDC config is shown in Appendix 2.

Note that the LDAP suffix given here matches the search base used by nss_ldap and pam_ldap. This allows Samba to share the entries that already exist to describe people/accounts. Note also that the admin DN is given: in this case I have used the name of the DSA manager (equivalent to root in directory terms), but it should be possible to use a less powerful ID if ACLs are set up carefully. Samba needs to bind to the directory as a user with write permission so that it can insert new attributes in entries. The password is stored in the secrets database using the command:

```
smbpasswd -w <password>
```

Unfortunately, the password has to be typed on the command line so make sure you clear out any command-history files afterwards!

Samba introduces several new attributes, so it's schema file must be copied from `samba-2.2.2/examples/LDAP/samba.schema` to the server schema directory and an appropriate include statement has to be added to *slapd.conf*. The schema file supplied tries to re-define *displayName* so this clause must be commented out before restarting the server.

Samba needs to set up two new password hashes and several other attributes for each user. The password hashes cannot be derived from the Unix password data, so be prepared to supply new passwords for each user:

```
smbpasswd -a <username>
```

Unfortunately, this is where the experimental nature of the code starts to show through. Some of the attributes get bad values: in particular, *rid* gets set to zero, and both *smbHome* and *profilePath* get values including %-expansions that are not properly handled when the data is used. These need to be fixed by hand: I used *ldapsearch* to extract the complete attribute set into a file for editing, and *ldapmodify* to put the updated values back.

Before a Windows client can join a domain, it needs a 'trust account' to be set up on the PDC. With the appropriate scripts configured on Samba, this can be largely automated but for test purposes I created Unix-style accounts by hand. These have usernames formed by appending '$' to the Windows 'computer-name', and are made ready for use with the command:

```
smbpasswd -a -m <computername>
```

With these preparations done, I was able to join an NT client to my domain and log in using the same username and password used with the Unix clients.

There is obviously some work still needed on the LDAP PDC code, but once stable there are some interesting possibilities. One option would be to do away with backup domain controllers altogether, and support distribution and resilience by having replicated LDAP servers each serving the same data to one or more Samba PDCs. I have yet to work out the exact effect of moving a Windows client from one such domain controller to another.

Looking at the problem the other way round, it should be possible to run *nss_ldap* and *pam_ldap* against an Active Directory system. The sample /etc/ldap.conf shipped with the PADL code includes an example configuration for this purpose, though I have not tried it.

Security

With such a concentration of data in the directory, security becomes very important[3]. Anyone who could modify the data could give themselves access to vast numbers of machines at a stroke. Some data needs to be protected from unauthorised viewing: although all passwords are hashed, anyone who can read the hashes can mount a dictionary attack. More subtly, anyone who can hijack a client-server connection can feed bogus data to an individual client, or use the client's privileges to modify server data. All these things can be protected against, and LDAP now has most of the tools needed to do it.

Access Control

Control over who may read what and who may change what is exercised with Access Control Lists (ACLs). This is one of the non-standardised areas of LDAP, and it varies a lot from one server to another. However, all modern servers provide enough control to protect passwords. Here is a simple ACL section from OpenLDAP:

```
access to attrs=userPassword
        by self write
        by anonymous auth
        by * none

access to *
        by * read
```

This allows authenticated users to change their own passwords, allows un-authenticated users to authenticate, and prevents all other access to the *userPassword*. It permits read access to everything else. Obviously if Samba PDC data is being stored then similar protection must be extended to *lmPassword* and *ntPassword*.

Client Authentication

The simplest form of client authentication is to bind to the server using a cleartext password. This is the method normally used by *pam_ldap* for checking login passwords. For security, this method should only be used with encrypted connections.

A more secure method is to use one of the SASL authentication mechanisms, such as DIGEST-MD5[4]. This is based on a secret known to both the client and the server, allowing for a simple challenge-response scheme. SASL is also capable of negotiating data encryption to protect subsequent operations.

LDAP also supports encryption and authentication using Transport Layer Security[5]. TLS is closely related to the older SSL scheme, and uses the same certificate-based methods. In its simplest form, TLS provides proof of server identity and protection of data in transit so it is useful where plaintext passwords might be passed across the net. The same mechanism can also be used to prove the identity of the client to the server, where the client has been issued with a suitable X.509 certificate.

It is often more appropriate (and cheaper) to use a private Certification Authority for this purpose rather than getting certificates from a public supplier. This is easy to set up using OpenSSL:

```
openssl genrsa -des3 -out ca.key 2048
openssl req -new -x509 -days 365 -key ca.key -out ca.cert
```

You will be prompted to supply some information. The result of the commands above will be two files: a key file and a certificate file. These are the basis of the Certification Authority.

```
openssl genrsa  -out ldap.key 1024
openssl req -new -key ldap.key -out ldap.csr
```

You will again be asked for information to fill in the Certificate Signing Request. The critical item is to put the LDAP server name

in when prompted for 'Common Name'. This should be the *exact* name that clients will use when contacting the server. It is best to use a fully-qualified domain name. Note that the *ldap.key* file is not encrypted as it must be read by the LDAP server: protect this file very carefully! Use the CA to sign the certificate:

```
openssl x509 -req -in ldap.csr -out ldap.cert -CA ca.cert \
                  -CAkey ca.key -CAcreateserial -days 365
```

The result is a file called *ldap.cert* containing the certificate for the LDAP server. You can examine certificates with the command:

```
openssl x509 -in ldap.cert -text -noout
```

The LDAP server needs copies of its own certificate and key files, and also a copy of the CA certificate. The key file must only be readable by the server process, so make it mode 400 and owned by the owner of the server (OpenLDAP can be run as root, or can be told to set UID to another user after binding the TCP ports). Add lines to *slapd.conf* to tell the server about the new files:

```
TLSCertificateFile      /etc/openldap/keys/server.cert
TLSCertificateKeyFile   /etc/openldap/keys/server.key
TLSCACertificateFile    /etc/openldap/keys/ca.cert
```

When the server is next restarted it will be able to use TLS. It will also support SSL if told to bind to an `ldaps:///` URL. To make *nss_ldap* and *pam_ldap* use TLS, add a line to /etc/ldap.conf on the client machines:

```
ssl start_tls
```

This provides transport-level encryption of LDAP sessions, thus giving some protection to passwords etc. However, it does not guard against bogus servers: to get this extra protection it is necessary to give the client machines a copy of the CA certificate that was used to sign the server certificate. Recent versions of *nss_ldap* and *pam_ldap* will verify the server identity if lines of this form are added to /etc/ldap.conf:

```
tls_checkpeer yes
tls_cacertfile /etc/ssl/ca.cert
```

Now, if a bogus server is encountered the client will immediately close the LDAP connection and deny the attempted login. Note that the reason for the denied login can be hard to find in the client logfiles: it may be necessary to turn on a high level of debugging to see the relevant message! Note also that certificate verification is a fairly recent addition to *nss_ldap* and *pam_ldap* so the versions shipped with most OS distributions do not yet have it.

Samba-2.2.2 has the option to use TLS when accessing LDAP, though it does not yet have code to check server certificates. To enable TLS add this to the global section of the config file:

```
ldap ssl = start tls
```

Other nameservice clients like Automount and AMD do not yet have the ability to use TLS at all, so unfortunately there are still chinks in the armour.

Management Tools

This area is still rather lacking. The PADL migration tools are good for converting and loading the initial data, but once LDAP is in use as the nameservice you will need other tools to create and manage accounts etc. Scripts can be built around *ldapsearch* and *ldapmodify* from the OpenLDAP distribution, or around one of the Perl modules that support LDAP. Another useful tool is GQ - a general-purpose LDAP browser/modifier.

There is at least one project under way to develop specific management tools for LDAP NIS systems. *ldaputils* is based at SourceForge and eventually plans to have a web-based management interface.

Performance

An LDAP server providing NIS service is likely to handle much more traffic than one just servicing 'white pages' requests so performance is an important consideration in a large network. When sizing servers it is necessary to know how much traffic to expect: Table 1 summarises the results of some simple tests with both *pam_ldap* and *nss_ldap* configured.

More recent versions of *nss_ldap* make much more efficient use of the server than the older ones. The use of a cache daemon also helps, though it does result in one more long-lived LDAP connection per client machine.

An LDAP server with logging turned off should not have any difficulty in servicing 500 searches per second, but if TLS is in use the cost of connection setup and binding is likely to far outweigh the search load. A large pool of clients will also result in many hundreds of connections being held open, so the server and its supporting operating system must be configured to support a sufficient number of open files.

LDAP servers can be configured to replicate data from a master server, so the read-only query load can be spread across as many machines as necessary. Operations (like password changing) that result in changes to the database get referred back to the master server. The format of /etc/ldap.conf supports multiple servers in

the form of a space-separated list, and the servers are tried in turn until one answers or the list is exhausted.

Client & Operation	Connect	Bind	Search	Connections Held Open Per session
Red Hat 7.1 login	8	8	13*	
Red Hat 7.1 login with nscd	5	5	3	1¤
Samba connect (not PDC)	5	5	19	1
Samba connect with nscd (not PDC)	6	6	8	1¤
Samba connect with PDC and nscd	2	2	5	0¤
Red Hat 7.2 login with nscd	2	4	6	1¤
* Two of these try to return *every* entry from the Group table ¤ Plus one connection per client machine for *nscd*				
Table 1: LDAP load imposed by clients				

Conclusions

It is quite possible to use LDAP as a Network Information Service, though at the moment it is not quite an 'out of the box and into production' product. There is potential for much greater security than most other forms of NIS, and with a bit of planning it should provide good performance too.

References

1. *The Hesiod Name Server*, Stephen P Dyer, Project Athena, Proceedings of the USENIX Winter 1988 Technical Conference www.mit.edu/afs/athena.mit.edu/astaff/project/hesioddev-/hesiod.dist/doc/hesiod.PS
2. RFC2307 *An Approach for Using LDAP as a Network Information Service*, L Howard. Work is under way to update the RFC: http://www.padl.com/~lukeh/rfc2307bis.txt
3. RFC2829 Authentication Methods for LDAP
4. RFC2222 Simple Authentication and Security Layer.
5. RFC2830 Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security
6. RFC2251 Lightweight Directory Access Protocol (v3)

Sources

OpenLDAP	`www.openldap.org`
OpenSSL	`www.openssl.org`
Cyrus SASL	`asg.web.cmu.edu/cyrus/`
Berkeley DB	`www.sleepycat.com`
NSS LDAP PAM LDAP Migration tools	`www.padl.com/software.html`
GQ	`biot.com/gq/`
LDAPutils	sourceforge.net/projects/ldaputils/
This paper and others by Andrew Findlay	www.skills–1st.co.uk/papers/afindlay.html

Appendix 1: slapd.conf

```
# slapd.conf
#

# Log level is a bitfield. 768 provides
# reasonable activity logging
# Logging goes to syslog with facility code
# LOCAL4

loglevel 768

include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
include /usr/local/etc/openldap/schema/java.schema
include /usr/local/etc/openldap/schema/krb5-kdc.schema
include /usr/local/etc/openldap/schema/misc.schema
include /usr/local/etc/openldap/schema/nadf.schema
include /usr/local/etc/openldap/schema/nis.schema
include /usr/local/etc/openldap/schema/openldap.schema
include /usr/local/etc/openldap/schema/corba.schema
include /usr/local/etc/openldap/schema/samba.schema

pidfile         /usr/local/var/slapd.pid
argsfile        /usr/local/var/slapd.args

# How we want passwords stored
password-hash {SSHA}

# Define SSL and TLS properties (optional)
TLSCertificateFile      /etc/openldap/keys/server.cert
TLSCertificateKeyFile   /etc/openldap/keys/server.key
TLSCACertificateFile    /etc/openldap/keys/ca.cert

# Define SASL properties (optional)
sasl-realm      green
sasl-host       brick.skills-1st.co.uk
sasl-secprops   minssf=112

#######################################################
# database definition
#######################################################

database        ldbm
suffix          "dc=example,dc=org"
rootdn          "cn=Manager,dc=example,dc=org"

# Cleartext passwords, especially for the rootdn, should
# be avoided
# See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication is encouraged.
rootpw          secret

# The database directory MUST exist prior to running slapd
# AND should only be accessible by the slapd/tools.
# Mode 700 is recommended.
directory       /usr/local/var/openldap-ldbm

# Indices to maintain
index   uid             eq
index   uidNumber       eq
index   gidNumber       eq
index   cn              eq
index   memberUid       eq
index   ipServicePort   eq

# Access Control

# Users can change their own passwords
# Everyone can read everything except passwords
```

```
access to attrs=userPassword
        by self write
        by anonymous auth
        by * none

access to *
        by * read

#######################################################################
# end of slapd.conf
#######################################################################
```

Appendix 2: Samba config

```
[global]
        netbios name = vm15
        workgroup = S2
        encrypt passwords = Yes
        preferred master = Yes
        wins support = Yes
        log level = 1

        # Make this a PDC
        domain logons = Yes
        os level = 34
        local master = yes
        domain master = yes

        # LDAP params
        ldap admin dn = cn=manager,dc=example,dc=org
        ldap server = brick.skills-1st.co.uk
        ldap suffix = dc=example,dc=org
        ldap ssl = start tls

# Required share on PDC
[netlogon]
        comment = Domain logon service
        path = /var/samba/netlogon
        public = no
        writeable = no
        browsable = no

# Profile store for NT/2k
[profile]
        path = /var/samba/profile
        create mask = 0600
        directory mask = 0700
        nt acl support = no
        read only = no

[homes]
        guest ok = no
        read only = no
```