OpenLDAP installation on Debian

Posted by docelic on Wed 19 Mar 2008 at 10:40

The purpose of this article is to give you a straight-forward, Debian-friendly way of installing and configuring OpenLDAP. By the end of this guide, you will have a functional LDAP server that will serve as a central authentication system for user logins onto all machines in the network, without the need to manually create users' accounts on individual machines.

However, for improved authentication security and a true networked solution, it is recommended to use LDAP in combination with Kerberos, with a matching Kerberos setup explained in the MIT Kerberos 5 Guide.

Table of Contents

Introduction

LDAP is a service that has been traditionally captivating system administrators' and advanced users' interest, but its (seemingly or not) high entry barrier and infrastructure requirements have been preventing many from using it.

LDAP has already been the topic of numerous publications. Here, we will present only the necessary summary; enough information to establish the context and to achieve practical results.

You do not need to follow any external links; however, the links have been provided both throughout the article and listed all together at the end, to serve as pointers to more precise technical treatment of individual topics.

The role of LDAP within a network

OpenLDAP is an open source implementation of the Lightweight Directory Access Protocol. Directory itself is a tree-structured, read-optimized database. Yellow pages or a phonebook are good associations to have in mind, even though LDAP is much more powerful.

We will use OpenLDAP to provide a central authentication location for user logins anywhere on the network, with their home directories being automatically created on their first access to individual machines.

This guide can be followed standalone to make OpenLDAP both perform authentication and serve user meta data. However, using LDAP for authentication as shown here is not secure due to plain text connections made to the LDAP server and passwords travelling over the wire. It is therefore advised to use LDAP in combination with a superior and secure network authentication mechanism Kerberos, as explained in another article from the series, the MIT Kerberos 5 Guide. That said, let's move onto our

LDAP setup.

From a technical perspective, LDAP directory consists of a set of hierarchically organized *entries*. Each entry belongs to certain *Object Classes* and contains various `key=value` pairs called *attributes*.

Each entry is uniquely identified by a *Distinguished name* ("DN"). DN is formed as a list of components, separated by commas, that provide "full path" to the entry, starting from the top of the tree. For example, company Example, Inc. would have the root of the tree in `dc=example,dc=com`. A person employed by Example, Inc. would then have a corresponding LDAP entry with DN `cn=person,ou=People,dc=example,dc=com`. Which attributes may or may not be present under an entry is then governed by the entry's objectClasses.

You might notice the individual components themselves, such as `cn=person` above, are also formed as `key=value` pairs. Those "keys", `cn`, `ou` and `dc`, stand for `Common Name`, `Organizational Unit` and `Domain Component`. They are a part of every-day LDAP terminology that you will get used to.

ObjectClasses, attributes, syntaxes, matching rules and other details of the tree structure are loaded at LDAP server startup by reading the configured *schema* files.

Let's quickly identify LDAP-specific elements in the metadata retrieval process:

- LDAP is not in any way related to traditional system usernames or other data. However, part of its functionality in our setup will consist in storing information traditionally found in Unix files `/etc/passwd` and `/etc/group`, thus making that data network-accessible at a centralized location.

  People's login names will be used in pairing people with the corresponding information in the LDAP tree. For example, username `person` will map to an LDAP entry `uid=person,ou=People,dc=example,dc=com`.

- LDAP can be configured to contain user passwords. Passwords can be used both for authenticating as specific users and gaining access to protected entries, and for verifying whether the user knows the correct password.

  When a user opens a LDAP client and intends to browse the directory, his password is used to establish his identity and the set of privileges. When LDAP is configured to perform user authentication, his password is only used to perform a connection to the LDAP directory — successful connection ("bind") implies the user knew the correct password.

You can find the complete LDAP documentation at the OpenLDAP website. For an authoritative OpenLDAP book, see Gerald Carter's `LDAP System Administration` by O'Reilly.

Glue layers: integrating LDAP with system software

NSS

On all GNU/Linux-based platforms, NSS is available for network data retrieval configuration. NSS is an implementation of the `Name Service Switch` mechanism.

NSS will allow for inclusion of LDAP into the "user data" path of all services, regardless of whether they natively support LDAP or not.

You can find the proper introduction (and complete documentation) on the NSS website. Also take a look at the nsswitch.conf(5) manual page.

PAM

Likewise, on all GNU/Linux-based platforms, another piece of the puzzle, Linux-PAM, is available for service-specific authentication configuration. Linux-PAM is an implementation of PAM ("`Pluggable Authentication Modules`") from Sun Microsystems.

Network services, instead of having hard-coded authentication interfaces and decision methods, invoke PAM through a standard, pre-defined interface. It is then up to PAM to perform any and all authentication-related work, and report the result back to the application.

Exactly how PAM reaches the decision is none of the service's business. In traditional set-ups, that is most often done by asking and verifying usernames and passwords. In advanced networks, that could be retina scans or — Kerberos tickets, as explained in another article from the series, MIT Kerberos 5 Guide.

You can find the proper introduction (and complete documentation) on the Linux-PAM website. Pay special attention to the PAM Configuration File Syntax page. Also take a look at the Linux-PAM(7) and pam(7) manual pages.

Conventions

It's quite disappointing when you are not able to follow the instructions found in the documentation. Let's agree on a few points before going down to work:

- Our platform of choice, where we will demonstrate a practical setup, will be Debian GNU.
- Install sudo. Sudo is a program that will allow you to carry out system administrator tasks from your normal user account. All the examples in this article requiring root privileges use sudo, so you will be able to copy-paste them to your shell.

  ```
  su -c 'apt-get install sudo'
  ```

  If asked for a password, type in the root user's password.

  To configure sudo, add the following line to your `/etc/sudoers`, replacing `$USERNAME` with your login name:

  ```
  $USERNAME ALL=(ALL) NOPASSWD: ALL
  ```

- Debian packages installed during the procedure will ask us a series of questions through the so-called *debconf* interface. To configure debconf to a known state, run:

  ```
  sudo dpkg-reconfigure debconf
  ```

  When asked, answer *interface*=`Dialog` and *priority*=`low`.

- Monitoring log files is crucial in detecting problems. The straight-forward, catch-all routine to this is opening a terminal and running:

  ```
  cd /var/log; sudo tail -f daemon.log sulog user.log auth.log debug kern.log syslog dmesg me
  ```

  The command will keep printing log messages to the screen as they arrive.

- Our test system will be called `monarch.spinlock.hr` and have an IP address of `192.168.7.12`. Both the server and the client will be installed on the same machine. However, to differentiate between client and server roles where relevant, the client will be referred to as `monarch.spinlock.hr` and the server as `ldap.spinlock.hr`. The following addition will be made to `/etc/hosts` to completely support this scheme:

  ```
  192.168.7.12    monarch.spinlock.hr krb.spinlock.hr ldap.spinlock.hr monarch krb ldap
  ```

OpenLDAP

Server installation

The following procedure uses the libdb library version `4.6`. If this version is not available in your Debian repository, use `libdb4.4` or `libdb4.2`.

```
sudo apt-get install slapd ldap-utils libldap2 libdb4.6
```

Debconf answers for reference:

```
Omit OpenLDAP server configuration? No
DNS domain name: spinlock.hr
Organization name? spinlock.hr
Administrator password: password
Confirm password: password
Database backend to use: BDB
Do you want the database to be removed when slapd is purged? No
Allow LDAPv2 protocol? No
```

As soon as the installation is done, the OpenLDAP server (command **slapd**) will start.

Initial configuration

Our OpenLDAP server is already running, so let's first configure `/etc/ldap/ldap.conf`, a common configuration file for all LDAP clients. This will allow us to run **ldapsearch** and other commands without having to list all the basic parameters by hand each time.

Enable the following two lines in `/etc/ldap/ldap.conf`, creating the file if necessary:

```
BASE  dc=spinlock, dc=hr

URI ldap://192.168.7.12/
```

Then, let's edit the server's configuration file, */etc/ldap/slapd.conf*, to fine-tune its behavior.

Make sure all the schema files are enabled:

```
include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/cosine.schema
include         /etc/ldap/schema/nis.schema
include         /etc/ldap/schema/inetorgperson.schema
```

Change the verbosity level from `0` or "`none`" to `256`:

```
loglevel 256
```

Search for line "`index objectClass eq`" and add another search index. In particular combinations, it may be possible to receive no results when the searched entries are not indexed, so this step is important:

```
index           objectClass eq
index           uid         eq
```

To make the new index option apply, run the following three commands.

```
sudo invoke-rc.d slapd stop
sudo slapindex
sudo chown openldap:openldap /var/lib/ldap/*
sudo invoke-rc.d slapd start
```

Initial test

It's already the time to test the installation. Our OpenLDAP server does not contain much information, but a basic read operation can be performed normally.

In LDAP terms, read operation is called a "search". To perform a search using generic command-line utilities, we have **ldapsearch** and **slapcat** available.

Ldapsearch (and other LDAP utilities prefixed "ldap") perform operations "online", using the LDAP protocol.

Slapcat (and other OpenLDAP utilities prefixed "slap") perform operations "offline", directly by opening files on the local filesystem. For this reason, they can only be ran locally on the OpenLDAP server and they require administrator privileges. When they involve writing to the database, the OpenLDAP server usually needs to be stopped first.

In the output of the two search commands, you will notice two LDAP entries, one representing the top level element in the three, and another representing the LDAP administrator's entry.

```
ldapsearch  -x

# extended LDIF
#
# LDAPv3
# base <dc=spinlock, dc=hr> (default) with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# spinlock.hr
dn: dc=spinlock,dc=hr
objectClass: top
objectClass: dcObject
objectClass: organization
o: spinlock.hr
dc: spinlock

# admin, spinlock.hr
dn: cn=admin,dc=spinlock,dc=hr
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

Now, in the following **slapcat** output, pay attention to the extra attributes not printed with **ldapsearch**. One of those is `userPassword`, which is not shown to anonymous readers due to an appropriate access restriction in `/etc/ldap/slapd.conf`.

```
sudo  slapcat

dn: dc=spinlock,dc=hr
objectClass: top
objectClass: dcObject
objectClass: organization
o: spinlock.hr
dc: spinlock
structuralObjectClass: organization
entryUUID: 350a2db6-87d3-102c-8c1c-1ffeac40db98
creatorsName:
modifiersName:
createTimestamp: 20080316183324Z
modifyTimestamp: 20080316183324Z
entryCSN: 20080316183324.797498Z#000000#000#000000

dn: cn=admin,dc=spinlock,dc=hr
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
```

```
description: LDAP administrator
userPassword:: e2NyeXB0fVdSZDJjRFdRODluNHM=
structuralObjectClass: organizationalRole
entryUUID: 350b330a-87d3-102c-8c1d-1ffeac40db98
creatorsName:
modifiersName:
createTimestamp: 20080316183324Z
modifyTimestamp: 20080316183324Z
entryCSN: 20080316183324.804398Z#000000#000#000000
```

Creating basic tree structure

As explained, the LDAP database is structured as a tree. The top level element in the tree for your organization is often its domain name. In case of a domain `spinlock.hr`, the toplevel tree element would be `dc=spinlock,dc=hr`.

On the next level below, an organization is often divided into "organizational units", such as people, groups, hosts, services, networks, protocols etc.

Accordingly, to support people's Unix "meta data" in our LDAP directory, we will be interested in creating two of the mentioned organizational units, `People` and `Group`. The two will roughly correspond to the Unix `/etc/passwd` and `/etc/group` files.

Ldap data is interchanged in a textual format called LDIF. Command-line LDAP utilities receive and output data in this format. Note that LDIF stream can also contain commands, such as for adding, modifying or removing LDAP entries.

Knowing this, we'll put together a simple LDIF file, `/var/tmp/ou.ldif`, that will instruct the server to add the two organizational units. Pay attention to the empty lines separating the entries:

```
dn: ou=People,dc=spinlock,dc=hr
ou: People
objectClass: organizationalUnit

dn: ou=Group,dc=spinlock,dc=hr
ou: Group
objectClass: organizationalUnit
```

To load the LDIF file into the server, let's show an example using the offline tool, **slapadd**:

```
sudo invoke-rc.d slapd stop
sudo slapadd -c -v -l /var/tmp/ou.ldif
sudo invoke-rc.d slapd start
```

Let's use **ldapsearch** to verify the entries have been created.

```
ldapsearch  -x  ou=people

# extended LDIF
#
# LDAPv3
# base <dc=spinlock, dc=hr> (default) with scope subt
# filter: ou=people
# requesting: ALL
#

# People, spinlock.hr
dn: ou=People,dc=spinlock,dc=hr
ou: People
objectClass: organizationalUnit

# search result
search: 2
```

```
result: 0 Success

# numResponses: 2
# numEntries: 1
```

Creating user accounts

In the same manner we've created the two organizational units, let's create our first group and a person belonging to it. Again, we approach the problem by constructing and loading an LDIF file, `/var/tmp/user1.ldif`:

```
dn: cn=mirko,ou=group,dc=spinlock,dc=hr
cn: mirko
gidNumber: 20000
objectClass: top
objectClass: posixGroup

dn: uid=mirko,ou=people,dc=spinlock,dc=hr
uid: mirko
uidNumber: 20000
gidNumber: 20000
cn: Mirko
sn: Mirko
objectClass: top
objectClass: person
objectClass: posixAccount
objectClass: shadowAccount
loginShell: /bin/bash
homeDirectory: /home/mirko
```

To load the LDIF file into the server, let's show an example using the online tool, **ldapadd**. Since, as said previously, **ldapadd** uses the LDAP protocol, we'll have to bind to the server as system administrator and type in the correct password:

```
ldapadd -c -x -D cn=admin,dc=spinlock,dc=hr -W -f /var/tmp/user1.ldif

Enter LDAP Password: PASSWORD
adding new entry "cn=mirko,ou=group,dc=spinlock,dc=hr"

adding new entry "uid=mirko,ou=people,dc=spinlock,dc=hr"
```

To define the new user's password, let's run an online tool **ldappasswd**. This step is not needed if you plan to use LDAP in combination with Kerberos and therefore leave Kerberos to store passwords and perform authentication-related work, as explained in MIT Kerberos 5 Guide.

```
ldappasswd -x -D cn=admin,dc=spinlock,dc=hr -W -S uid=mirko,ou=people,dc=spinloc

New password: new user password
Re-enter new password: new user password
Enter LDAP Password:  admin password
Result: Success (0)
```

Let's use **ldapsearch** to verify the user entry has been created. Note that the password field, `userPassword`, will not be shown even if you created it, due to the default access restrictions in `/etc/ldap/slapd.conf`.

```
ldapsearch -x uid=mirko

# extended LDIF
#
# LDAPv3
# base <dc=spinlock, dc=hr> (default) with scope subtree
# filter: uid=mirko
# requesting: ALL
#

# mirko, people, spinlock.hr
dn: uid=mirko,ou=people,dc=spinlock,dc=hr
uid: mirko
uidNumber: 20000
gidNumber: 20000
cn: Mirko
sn: Mirko
objectClass: top
objectClass: person
objectClass: posixAccount
loginShell: /bin/bash
homeDirectory: /home/mirko

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

Congratulations! You have a working LDAP setup.

Graphical LDAP browsers

While the LDIF format shown represents the basis of data interchange in LDAP, for day to day work a graphical client may be preferred.

There are many LDAP GUI clients, only few of which seem useful for general-purpose LDAP editing. Two of those are gq, the "Gentleman's LDAP client", and luma.

Gq version 1.0.0 packaged in Debian has a bug in configuration saving code. The straightforward way to using it is:

- running gq on the same machine as the LDAP server ('localhost')
- following menu File —> Preferences —> Servers —> 'localhost' —> Edit, typing "`dc=`*`spinlock`*`,dc=`*`hr`*" under "General/Base DN" and "`cn=admin,dc=`*`spinlock`*`,dc=`*`hr`*" under "Details/Bind DN".
- back on the main program window, following Browse —> localhost, typing in admin password and browsing the LDAP tree

NSS configuration

Now that we have a new user created in LDAP, we should allow the system to see it. For example, let's test for existence of users `root` and `mirko`. The administrator will be present, while `mirko` will not:

```
id root
uid=0(root) gid=0(root) groups=0(root)

id mirko
id: mirko: No such user
```

To enable the system see LDAP accounts, we need to install and configure libnss-ldap (which may automatically install libpam-ldap as well):

```
sudo apt-get install libnss-ldap nscd
```

All debconf answers for reference:

```
LDAP server Uniform Resource Identifier: ldap://192.168.7.12/ (Note the "ldap://", NOT "ldap:
Distinguished name of the search base: dc=spinlock,dc=hr
LDAP version to use: 3
Does the LDAP database require login? No
Special LDAP privileges for root? No
Make the configuration file readable/writeable by its owner only? No
Make local root Database admin. No
Does the LDAP database require login? No
Local crypt to use when changing passwords. crypt
```

To configure the NSS module further, open `/etc/libnss-ldap.conf`. Locate and adjust the configuration lines as shown:

```
base dc=spinlock,dc=hr
uri ldap://192.168.7.12/
```

Finally, to activate the LDAP NSS module, edit `/etc/nsswitch.conf` by replacing the two lines mentioning `passwd` and `group` with the following:

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

Stop nscd, the Name Service Caching Daemon, but do leave it starting up automatically on the next system boot:

```
sudo invoke-rc.d nscd stop
```

Nscd is used to cache metadata locally, instead of querying the LDAP server each time. It is a very efficient service in the long run, but we have disabled it for the moment to always retrieve the data directly from the LDAP server.

Finally, verify LDAP users are now visible:

```
id  mirko
uid=20000(mirko) gid=20000(mirko) groups=20000(mirko)
```

PAM configuration

The final step in this article pertains to integrating LDAP into the system authentication procedure.

Let's install and configure libpam-ldap. (You might have already done this step automatically, during libnss-ldap installation — in that case Debian will just report the package is already installed).

```
sudo apt-get install libpam-ldap
```

Debconf answers for reference:

```
Make local root Database admin. No
Does the LDAP database require login? No
Local crypt to use when changing passwords. crypt
```

To configure the PAM module, open `/etc/pam_ldap.conf`. Locate and adjust the configuration lines as shown:

```
base dc=spinlock,dc=hr
uri ldap://192.168.7.12/
```

Now let's configure Linux-PAM itself. PAM configuration is quite fragile, so use the provided examples that have been verified to work. For any modifications, you will want to look at PAM Configuration File Syntax and pay special attention to seemingly insignificant variations — with PAM, they often make a whole world of difference.

PAM will require the user to be present *either* in the local password file *or* in LDAP and to know the correct password, for the authentication process to continue.

Note that authentication through LDAP, as shown here, is not secure, due to connections to the LDAP server being made in plain text and passwords travelling over the wire.

Instead of encrypting the connection to the LDAP server, the PAM files shown below also support Kerberos for authentication, if you've installed Kerberos as explained in MIT Kerberos 5 Guide. In that case, modify the PAM lines as noted in file comments and the authentication will be performed in a completely secure and superior manner using Kerberos.

/etc/pam.d/common-account

```
account sufficient      pam_unix.so
account required        pam_ldap.so

# Enable if using Kerberos:
#account required        pam_krb5.so
```

/etc/pam.d/common-auth

```
# Disable the three lines if using Kerberos:
auth [success=1 default=ignore] pam_unix.so nullok_secure
auth required pam_ldap.so use_first_pass
auth required pam_permit.so

# Enable the three lines if using Kerberos:
#auth    sufficient        pam_unix.so nullok_secure
#auth    sufficient        pam_krb5.so use_first_pass
#auth    required          pam_deny.so
```

/etc/pam.d/common-session

```
session required        pam_unix.so
session required        pam_mkhomedir.so skel=/etc/skel/ umask=0022

# Enable if using Kerberos:
#session  optional  pam_krb5.so minimum_uid=1000
```

Logging in into the system

Having everything adjusted as shown, login to the system should succeed as user *mirko*:

```
Login: mirko
Password: password

Debian GNU/Linux tty5

Creating directory '/home/mirko'.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
mirko@host:~$
```

Conclusion

At this point, you have a functional LDAP installation.

You can rely on LDAP for central network authentication and sharing of user metadata (user IDs, group IDs, real names, group memberships, etc.).

However, as said above, the authentication through LDAP is not done in a network-secure manner, due to plain text connections and passwords travelling over the wire. To solve that problem, using Kerberos for network authentication instead of LDAP is recommended, as explained in the previous article in the series, the MIT Kerberos 5 Guide.

When users authenticate successfully, they will be logged in and placed in their home directory. However, in a central network authentication scheme, where users are not created on individual machines, their corresponding home directories will not exist. This problem is taken care of by the `pam_mkhomedir` module above, which automatically creates missing home directories.

The newest version of this article can always be found at http://techpubs.spinlocksolutions.com/dklar/ldap.html.

Davor Ocelic
http://www.spinlocksolutions.com/

Links

Platforms:
GNU
Debian GNU

Kerberos:
OpenLDAP

Glue layer:
NSS
Linux-PAM

Related infrastructural technologies:
MIT Kerberos

Commercial support:
Spinlock Solutions

Misc:
DocBook