OpenLDAP Server

The Lightweight Directory Access Protocol, or LDAP, is a protocol for querying and modifying a X.500-based directory service running over TCP/IP. The current LDAP version is LDAPv3, as defined in RFC4510, and the implementation in Ubuntu is OpenLDAP."

So the LDAP protocol accesses LDAP directories. Here are some key concepts and terms:

1. A LDAP directory is a tree of data *entries* that is hierarchical in nature and is called the Directory Information Tree (DIT).
2. An entry consists of a set of *attributes*.
3. An attribute has a *type* (a name/description) and one or more *values*.
4. Every attribute must be defined in at least one *objectClass*.
5. Attributes and objectclasses are defined in *schemas* (an objectclass is actually considered as a special kind of attribute).
6. Each entry has a unique identifier: its *Distinguished Name* (DN or dn). This, in turn, consists of a *Relative Distinguished Name* (RDN) followed by the parent entry's DN.
7. The entry's DN is not an attribute. It is not considered part of the entry itself.

The terms *object*, *container*, and *node* have certain connotations but they all essentially mean the same thing as *entry*, the technically correct term.

For example, below we have a single entry consisting of 11 attributes where the following is true:

- DN is "cn=John Doe,dc=example,dc=com"
- RDN is "cn=John Doe"
- parent DN is "dc=example,dc=com"

```
dn: cn=John Doe,dc=example,dc=com
cn: John Doe
givenName: John
sn: Doe
telephoneNumber: +1 888 555 6789
telephoneNumber: +1 888 555 1232
mail: john@example.com
manager: cn=Larry Smith,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
```

The above entry is in *LDIF* format (LDAP Data Interchange Format). Any information that you feed into your DIT must also be in such a format. It is defined in RFC2849.

Although this guide will describe how to use it for central authentication, LDAP is good for anything that involves a large number of access requests to a mostly-read, attribute-based (name:value) backend. Examples include an address book, a list of email addresses, and a mail server's configuration.

Installation

Install the OpenLDAP server daemon and the traditional LDAP management utilities. These are found in packages *slapd* and *ldap-utils* respectively.

The installation of slapd will create a working configuration. In particular, it will create a database instance that you can use to store your data. However, the suffix (or base DN) of this instance will be determined from the domain name of the localhost. If you want something different, edit `/etc/hosts` and replace the domain name with one that will give you the suffix you desire. For instance, if you want a suffix of *dc=example,dc=com* then your file would have a line similar to this:

```
127.0.1.1       hostname.example.com    hostname
```

You can revert the change after package installation.

This guide will use a database suffix of *dc=example,dc=com*.

Proceed with the install:

```
sudo apt install slapd ldap-utils
```

Since Ubuntu 8.10 slapd is designed to be configured within slapd itself by dedicating a separate DIT for that purpose. This allows one to dynamically configure slapd without the need to restart the service. This configuration database consists of a collection of text-based LDIF files located under `/etc/ldap/slapd.d`. This way of working is known by several names: the slapd-config method, the RTC method (Real Time Configuration), or the cn=config method. You can still use the traditional flat-file method (slapd.conf) but it's not recommended; the functionality will be eventually phased out.

Ubuntu now uses the *slapd-config* method for slapd configuration and this guide reflects that.

During the install you were prompted to define administrative credentials. These are LDAP-based credentials for the *rootDN* of your database instance. By default, this user's DN is *cn=admin,dc=example,dc=com*. Also by default, there is no administrative account created for the slapd-config database and you will therefore need to authenticate externally to LDAP in order to access it. We will see how to do this later on.

Some classical schemas (cosine, nis, inetorgperson) come built-in with slapd nowadays. There is also an included "core" schema, a pre-requisite for any schema to work.

Post-install Inspection

The installation process set up 2 DITs. One for slapd-config and one for your own data (dc=example,dc=com). Let's take a look.

1. This is what the slapd-config database/DIT looks like. Recall that this database is LDIF-based and lives under `/etc/ldap/slapd.d`:

```
/etc/ldap/slapd.d/
/etc/ldap/slapd.d/cn=config
/etc/ldap/slapd.d/cn=config/cn=module{0}.ldif
/etc/ldap/slapd.d/cn=config/cn=schema
/etc/ldap/slapd.d/cn=config/cn=schema/cn={0}core.ldif
/etc/ldap/slapd.d/cn=config/cn=schema/cn={1}cosine.ldif
/etc/ldap/slapd.d/cn=config/cn=schema/cn={2}nis.ldif
/etc/ldap/slapd.d/cn=config/cn=schema/cn={3}inetorgperson.ldif
/etc/ldap/slapd.d/cn=config/cn=schema.ldif
/etc/ldap/slapd.d/cn=config/olcBackend={0}hdb.ldif
/etc/ldap/slapd.d/cn=config/olcDatabase={0}config.ldif
/etc/ldap/slapd.d/cn=config/olcDatabase={-1}frontend.ldif
/etc/ldap/slapd.d/cn=config/olcDatabase={1}hdb.ldif
/etc/ldap/slapd.d/cn=config.ldif
```

Do not edit the slapd-config database directly. Make changes via the LDAP protocol (utilities).

2. This is what the slapd-config DIT looks like via the LDAP protocol:

On Ubuntu server 14.10, and possibly higher, the following command may not work due to a bug

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config dn

dn: cn=config

dn: cn=module{0},cn=config

dn: cn=schema,cn=config

dn: cn={0}core,cn=schema,cn=config

dn: cn={1}cosine,cn=schema,cn=config

dn: cn={2}nis,cn=schema,cn=config

dn: cn={3}inetorgperson,cn=schema,cn=config

dn: olcBackend={0}hdb,cn=config

dn: olcDatabase={-1}frontend,cn=config

dn: olcDatabase={0}config,cn=config

dn: olcDatabase={1}hdb,cn=config
```

Explanation of entries:

1. *cn=config*: global settings
2. *cn=module{0},cn=config*: a dynamically loaded module
3. *cn=schema,cn=config*: contains hard-coded system-level schema
4. *cn={0}core,cn=schema,cn=config*: the hard-coded core schema
5. *cn={1}cosine,cn=schema,cn=config*: the cosine schema
6. *cn={2}nis,cn=schema,cn=config*: the nis schema
7. *cn={3}inetorgperson,cn=schema,cn=config*: the inetorgperson schema
8. *olcBackend={0}hdb,cn=config*: the 'hdb' backend storage type
9. *olcDatabase={-1}frontend,cn=config*: frontend database, default settings for other databases
10. *olcDatabase={0}config,cn=config*: slapd configuration database (cn=config)
11. *olcDatabase={1}hdb,cn=config*: your database instance (dc=examle,dc=com)

3. This is what the dc=example,dc=com DIT looks like:

```
ldapsearch -x -LLL -H ldap:/// -b dc=example,dc=com dn

dn: dc=example,dc=com

dn: cn=admin,dc=example,dc=com
```

Explanation of entries:

1. *dc=example,dc=com*: base of the DIT
2. *cn=admin,dc=example,dc=com*: administrator (rootDN) for this DIT (set up during package install)

Modifying/Populating your Database

Let's introduce some content to our database. We will add the following:

1. a node called *People* (to store users)
2. a node called *Groups* (to store groups)
3. a group called *miners*
4. a user called *john*

Create the following LDIF file and call it `add_content.ldif`:

```
dn: ou=People,dc=example,dc=com
```

```
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
ou: Groups

dn: cn=miners,ou=Groups,dc=example,dc=com
objectClass: posixGroup
cn: miners
gidNumber: 5000

dn: uid=john,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: john
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 10000
gidNumber: 5000
userPassword: johnldap
gecos: John Doe
loginShell: /bin/bash
homeDirectory: /home/john
```

It's important that uid and gid values in your directory do not collide with local values. Use high number ranges, such as starting at 5000. By setting the uid and gid values in ldap high, you also allow for easier control of what can be done with a local user vs a ldap one. More on that later.

Add the content:

```
ldapadd -x -D cn=admin,dc=example,dc=com -W -f add_content.ldif

Enter LDAP Password: ********
adding new entry "ou=People,dc=example,dc=com"

adding new entry "ou=Groups,dc=example,dc=com"

adding new entry "cn=miners,ou=Groups,dc=example,dc=com"

adding new entry "uid=john,ou=People,dc=example,dc=com"
```

We can check that the information has been correctly added with the *ldapsearch* utility:

```
ldapsearch -x -LLL -b dc=example,dc=com 'uid=john' cn gidNumber

dn: uid=john,ou=People,dc=example,dc=com
cn: John Doe
gidNumber: 5000
```

Explanation of switches:

1. *-x:* "simple" binding; will not use the default SASL method
2. *-LLL:* disable printing extraneous information
3. *uid=john:* a "filter" to find the john user
4. *cn gidNumber:* requests certain attributes to be displayed (the default is to show all attributes)

Modifying the slapd Configuration Database

The slapd-config DIT can also be queried and modified. Here are a few examples.

1. Use *ldapmodify* to add an "Index" (DbIndex attribute) to your *{1}hdb,cn=config* database (dc=example,dc=com). Create a file, call it `uid_index.ldif`, with the following contents:

```
dn: olcDatabase={1}hdb,cn=config
add: olcDbIndex
olcDbIndex: uid eq,pres,sub
```

Then issue the command:

```
sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f uid_index.ldif

modifying entry "olcDatabase={1}hdb,cn=config"
```

You can confirm the change in this way:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={1}hdb)' olcDbIndex

dn: olcDatabase={1}hdb,cn=config
olcDbIndex: objectClass eq
olcDbIndex: uid eq,pres,sub
```

2. Let's add a schema. It will first need to be converted to LDIF format. You can find unconverted schemas in addition to converted ones in the `/etc/ldap/schema` directory.

   1. It is not trivial to remove a schema from the slapd-config database. Practice adding schemas on a test system.
   2. Before adding any schema, you should check which schemas are already installed (shown is a default, out-of-the-box output):

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=schema,cn=config dn

dn: cn=schema,cn=config

dn: cn={0}core,cn=schema,cn=config

dn: cn={1}cosine,cn=schema,cn=config

dn: cn={2}nis,cn=schema,cn=config

dn: cn={3}inetorgperson,cn=schema,cn=config
```

In the following example we'll add the CORBA schema.

1. Create the conversion configuration file `schema_convert.conf` containing the following lines:

```
include /etc/ldap/schema/core.schema
include /etc/ldap/schema/collective.schema
include /etc/ldap/schema/corba.schema
include /etc/ldap/schema/cosine.schema
include /etc/ldap/schema/duaconf.schema
include /etc/ldap/schema/dyngroup.schema
include /etc/ldap/schema/inetorgperson.schema
include /etc/ldap/schema/java.schema
include /etc/ldap/schema/misc.schema
include /etc/ldap/schema/nis.schema
include /etc/ldap/schema/openldap.schema
include /etc/ldap/schema/ppolicy.schema
include /etc/ldap/schema/ldapns.schema
include /etc/ldap/schema/pmi.schema
```

2. Create the output directory `ldif_output`.
3. Determine the index of the schema:

```
slapcat -f schema_convert.conf -F ldif_output -n 0 | grep corba,cn=schema

cn={1}corba,cn=schema,cn=config
```

When slapd ingests objects with the same parent DN it will create an *index* for that object. An index is contained within braces: *{X}*.

4. Use *slapcat* to perform the conversion:

```
slapcat -f schema_convert.conf -F ldif_output -n0 -H \
ldap:///cn={1}corba,cn=schema,cn=config -l cn=corba.ldif
```

The converted schema is now in `cn=corba.ldif`

5. Edit `cn=corba.ldif` to arrive at the following attributes:

```
dn: cn=corba,cn=schema,cn=config
...
cn: corba
```

Also remove the following lines from the bottom:

```
structuralObjectClass: olcSchemaConfig
entryUUID: 52109a02-66ab-1030-8be2-bbf166230478
creatorsName: cn=config
createTimestamp: 20110829165435Z
entryCSN: 20110829165435.935248Z#000000#000#000000
modifiersName: cn=config
modifyTimestamp: 20110829165435Z
```

Your attribute values will vary.

6. Finally, use *ldapadd* to add the new schema to the slapd-config DIT:

```
sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f cn\=corba.ldif

adding new entry "cn=corba,cn=schema,cn=config"
```

7. Confirm currently loaded schemas:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=schema,cn=config dn

dn: cn=schema,cn=config

dn: cn={0}core,cn=schema,cn=config

dn: cn={1}cosine,cn=schema,cn=config

dn: cn={2}nis,cn=schema,cn=config

dn: cn={3}inetorgperson,cn=schema,cn=config

dn: cn={4}corba,cn=schema,cn=config
```

For external applications and clients to authenticate using LDAP they will each need to be specifically configured to do so. Refer to the appropriate client-side documentation for details.

Logging

Activity logging for slapd is indispensible when implementing an OpenLDAP-based solution yet it must be manually enabled after software installation. Otherwise, only rudimentary messages will appear in the logs. Logging, like any other slapd configuration, is enabled via the slapd-config database.

OpenLDAP comes with multiple logging subsystems (levels) with each one containing the lower one (additive). A good level to try is *stats*. The slapd-config man page has more to say on the different subsystems.

Create the file `logging.ldif` with the following contents:

```
dn: cn=config
changetype: modify
replace: olcLogLevel
olcLogLevel: stats
```

Implement the change:

```
sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f logging.ldif
```

This will produce a significant amount of logging and you will want to throttle back to a less verbose level once your system is in production. While in this verbose mode your host's syslog engine (rsyslog) may have a hard time keeping up and may drop messages:

```
rsyslogd-2177: imuxsock lost 228 messages from pid 2547 due to rate-limiting
```

You may consider a change to rsyslog's configuration. In `/etc/rsyslog.conf` put:

You may consider a change to rsyslog's configuration. In /etc/rsyslog.conf, put:

```
# Disable rate limiting
# (default is 200 messages in 5 seconds; below we make the 5 become 0)
$SystemLogRateLimitInterval 0
```

And then restart the rsyslog daemon:

```
sudo systemctl restart syslog.service
```

Replication

The LDAP service becomes increasingly important as more networked systems begin to depend on it. In such an environment, it is standard practice to build redundancy (high availability) into LDAP to prevent havoc should the LDAP server become unresponsive. This is done through *LDAP replication*.

Replication is achieved via the *Syncrepl* engine. This allows changes to be synchronized using a *Consumer - Provider* model. The specific kind of replication we will implement in this guide is a combination of the following modes: *refreshAndPersist* and *delta-syncrepl*. This has the Provider push changed entries to the Consumer as soon as they're made but, in addition, only actual changes will be sent, not entire entries.

Provider Configuration

Begin by configuring the *Provider*.

1. Create an LDIF file with the following contents and name it `provider_sync.ldif`:

```
# Add indexes to the frontend db.
dn: olcDatabase={1}hdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: entryCSN eq
-
add: olcDbIndex
olcDbIndex: entryUUID eq

#Load the syncprov and accesslog modules.
dn: cn=module{0},cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: syncprov
-
add: olcModuleLoad
olcModuleLoad: accesslog

# Accesslog database definitions
dn: olcDatabase={2}hdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcHdbConfig
olcDatabase: {2}hdb
olcDbDirectory: /var/lib/ldap/accesslog
olcSuffix: cn=accesslog
olcRootDN: cn=admin,dc=example,dc=com
olcDbIndex: default eq
olcDbIndex: entryCSN,objectClass,reqEnd,reqResult,reqStart

# Accesslog db syncprov.
dn: olcOverlay=syncprov,olcDatabase={2}hdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpNoPresent: TRUE
olcSpReloadHint: TRUE

# syncrepl Provider for primary db
dn: olcOverlay=syncprov,olcDatabase={1}hdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpNoPresent: TRUE

# accesslog overlay definitions for primary db
dn: olcOverlay=accesslog,olcDatabase={1}hdb,cn=config
objectClass: olcOverlayConfig
objectClass: olcAccessLogConfig
olcOverlay: accesslog
olcAccessLogDB: cn=accesslog
```

```
olcAccessLogDB: cn=accesslog
olcAccessLogOps: writes
olcAccessLogSuccess: TRUE
# scan the accesslog DB every day, and purge entries older than 7 days
olcAccessLogPurge: 07+00:00 01+00:00
```

Change the rootDN in the LDIF file to match the one you have for your directory.

2. The *apparmor* profile for slapd will not need to be adjusted for the accesslog database location since `/etc/apparmor.d/local/usr.sbin.slapd` contains:

   ```
   /var/lib/ldap/ r,
   /var/lib/ldap/** rwk,
   ```

   Create a directory, set up a databse config file, and reload the apparmor profile:

   ```
   sudo -u openldap mkdir /var/lib/ldap/accesslog
   sudo -u openldap cp /var/lib/ldap/DB_CONFIG /var/lib/ldap/accesslog
   sudo systemctl reload apparmor.service
   ```

3. Add the new content and, due to the apparmor change, restart the daemon:

   ```
   sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f provider_sync.ldif
   sudo systemctl restart slapd.service
   ```

The Provider is now configured.

Consumer Configuration

And now configure the *Consumer*.

1. Install the software by going through Installation. Make sure the slapd-config databse is identical to the Provider's. In particular, make sure schemas and the databse suffix are the same.

2. Create an LDIF file with the following contents and name it `consumer_sync.ldif`:

   ```
   dn: cn=module{0},cn=config
   changetype: modify
   add: olcModuleLoad
   olcModuleLoad: syncprov

   dn: olcDatabase={1}hdb,cn=config
   changetype: modify
   add: olcDbIndex
   olcDbIndex: entryUUID eq
   -
   add: olcSyncRepl
   olcSyncRepl: rid=0 provider=ldap://ldap01.example.com bindmethod=simple
   binddn="cn=admin,dc=example,dc=com"
    credentials=secret searchbase="dc=example,dc=com" logbase="cn=accesslog"
    logfilter="(&(objectClass=auditWriteObject)(reqResult=0))" schemachecking=on
    type=refreshAndPersist retry="60 +" syncdata=accesslog
   -
   add: olcUpdateRef
   olcUpdateRef: ldap://ldap01.example.com
   ```

   Ensure the following attributes have the correct values:

   1. *provider* (Provider server's hostname -- ldap01.example.com in this example -- or IP address)
   2. *binddn* (the admin DN you're using)
   3. *credentials* (the admin DN password you're using)
   4. *searchbase* (the database suffix you're using)
   5. *olcUpdateRef* (Provider server's hostname or IP address)
   6. *rid* (Replica ID, an unique 3-digit that identifies the replica. Each consumer should have at least one rid)

3. Add the new content:

   ```
   sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f consumer_sync.ldif
   ```

You're done. The two databases (suffix: dc=example,dc=com) should now be synchronizing.

Testing

Once replication starts, you can monitor it by running

```
ldapsearch -z1 -LLLQY EXTERNAL -H ldapi:/// -s base -b dc=example,dc=com contextCSN

dn: dc=example,dc=com
contextCSN: 20120201193408.178454Z#000000#000#000000
```

on both the provider and the consumer. Once the output (`20120201193408.178454Z#000000#000#000000` in the above example) for both machines match, you have replication. Every time a change is done in the provider, this value will change and so should the one in the consumer(s).

If your connection is slow and/or your ldap database large, it might take a while for the consumer's *contextCSN* match the provider's. But, you will know it is progressing since the consumer's *contextCSN* will be steadly increasing.

If the consumer's *contextCSN* is missing or does not match the provider, you should stop and figure out the issue before continuing. Try checking the slapd (syslog) and the auth log files in the provider to see if the consumer's authentication requests were successful or its requests to retrieve data (they look like a lot of ldapsearch statements) return no errors.

To test if it worked simply query, on the Consumer, the DNs in the database:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b dc=example,dc=com dn
```

You should see the user 'john' and the group 'miners' as well as the nodes 'People' and 'Groups'.

Access Control

The management of what type of access (read, write, etc) users should be granted to resources is known as *access control*. The configuration directives involved are called *access control lists* or ACL.

When we installed the slapd package various ACL were set up automatically. We will look at a few important consequences of those defaults and, in so doing, we'll get an idea of how ACLs work and how they're configured.

To get the effective ACL for an LDAP query we need to look at the ACL entries of the database being queried as well as those of the special frontend database instance. The ACLs belonging to the latter act as defaults in case those of the former do not match. The frontend database is the second to be consulted and the ACL to be applied is the first to match ("first match wins") among these 2 ACL sources. The following commands will give, respectively, the ACLs of the hdb database ("dc=example,dc=com") and those of the frontend database:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={1}hdb)' olcAccess

dn: olcDatabase={1}hdb,cn=config
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous
              auth by dn="cn=admin,dc=example,dc=com" write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to * by self write by dn="cn=admin,dc=example,dc=com" write by *
  read
```

The rootDN always has full rights to its database. Including it in an ACL does provide an explicit configuration but it also causes slapd to incur a performance penalty.

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={-1}frontend)' olcAccess

dn: olcDatabase={-1}frontend,cn=config
olcAccess: {0}to * by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,
              cn=external,cn=auth manage by * break
olcAccess: {1}to dn.exact="" by * read
olcAccess: {2}to dn.base="cn=Subschema" by * read
```

The very first ACL is crucial:

```
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous
              auth by dn="cn=admin,dc=example,dc=com" write by * none
```

This can be represented differently for easier digestion:

```
to attrs=userPassword
        by self write
        by anonymous auth
        by dn="cn=admin,dc=example,dc=com" write
        by * none
        by * none

to attrs=shadowLastChange
        by self write
        by anonymous auth
        by dn="cn=admin,dc=example,dc=com" write
        by * none
```

This compound ACL (there are 2) enforces the following:

1. Anonymous 'auth' access is provided to the *userPassword* attribute for the initial connection to occur. Perhaps counter-intuitively, 'by anonymous auth' is needed even when anonymous access to the DIT is unwanted. Once the remote end is connected, howerver, authentication can occur (see next point).
2. Authentication can happen because all users have 'read' (due to 'by self write') access to the *userPassword* attribute.
3. The *userPassword* attribute is otherwise unaccessible by all other users, with the exception of the rootDN, who has complete access to it.
4. In order for users to change their own password, using `passwd` or other utilities, the *shadowLastChange* attribute needs to be accessible once a user has authenticated.

This DIT can be searched anonymously because of 'by * read' in this ACL:

```
to *
        by self write
        by dn="cn=admin,dc=example,dc=com" write
        by * read
```

If this is unwanted then you need to change the ACLs. To force authentication during a bind request you can alternatively (or in combination with the modified ACL) use the 'olcRequire: authc' directive.

As previously mentioned, there is no administrative account created for the slapd-config database. There is, however, a SASL identity that is granted full access to it. It represents the localhost's superuser (root/sudo). Here it is:

```
dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
```

The following command will display the ACLs of the slapd-config database:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={0}config)' olcAccess

dn: olcDatabase={0}config,cn=config
olcAccess: {0}to * by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,
              cn=external,cn=auth manage by * break
```

Since this is a SASL identity we need to use a SASL *mechanism* when invoking the LDAP utility in question and we have seen it plenty of times in this guide. It is the EXTERNAL mechanism. See the previous command for an example. Note that:

> 1. You must use *sudo* to become the root identity in order for the ACL to match.
> 2. The EXTERNAL mechanism works via *IPC* (UNIX domain sockets). This means you must use the *ldapi* URI format.

A succinct way to get all the ACLs is like this:

```
sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcAccess=*)' olcAccess olcSuffix
```

There is much to say on the topic of access control. See the man page for slapd.access.

TLS

When authenticating to an OpenLDAP server it is best to do so using an encrypted session. This can be accomplished using Transport Layer Security (TLS).

Here, we will be our own *Certificate Authority* and then create and sign our LDAP server certificate as that CA. Since *slapd* is compiled using the *gnutls* library, we will use the *certtool* utility to complete these tasks.

> 1. Install the *gnutls-bin* and *ssl-cert* packages:
>
> ```
> sudo apt install gnutls-bin ssl-cert
> ```
>
> 2. Create a private key for the Certificate Authority:
>
> ```
> sudo sh -c "certtool --generate-privkey > /etc/ssl/private/cakey.pem"
> ```
>
> 3. Create the template/file `/etc/ssl/ca.info` to define the CA:

```
cn = Example Company
ca
cert_signing_key
```

4. Create the self-signed CA certificate:

```
sudo certtool --generate-self-signed \
--load-privkey /etc/ssl/private/cakey.pem \
--template /etc/ssl/ca.info \
--outfile /etc/ssl/certs/cacert.pem
```

5. Make a private key for the server:

```
sudo certtool --generate-privkey \
--bits 1024 \
--outfile /etc/ssl/private/ldap01_slapd_key.pem
```

> Replace *ldap01* in the filename with your server's hostname. Naming the certificate and key for the host and service that will be using them will help keep things clear.

6. Create the `/etc/ssl/ldap01.info` info file containing:

```
organization = Example Company
cn = ldap01.example.com
tls_www_server
encryption_key
signing_key
expiration_days = 3650
```

The above certificate is good for 10 years. Adjust accordingly.

7. Create the server's certificate:

```
sudo certtool --generate-certificate \
--load-privkey /etc/ssl/private/ldap01_slapd_key.pem \
--load-ca-certificate /etc/ssl/certs/cacert.pem \
--load-ca-privkey /etc/ssl/private/cakey.pem \
--template /etc/ssl/ldap01.info \
--outfile /etc/ssl/certs/ldap01_slapd_cert.pem
```

Create the file `certinfo.ldif` with the following contents (adjust accordingly, our example assumes we created certs using https://www.cacert.org):

```
dn: cn=config
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ssl/certs/cacert.pem
-
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ssl/certs/ldap01_slapd_cert.pem
-
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ssl/private/ldap01_slapd_key.pem
```

Use the *ldapmodify* command to tell slapd about our TLS work via the slapd-config database:

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f /etc/ssl/certinfo.ldif
```

Contratry to popular belief, you do not need *ldaps://* in `/etc/default/slapd` in order to use encryption. You should have just:

```
SLAPD_SERVICES="ldap:/// ldapi:///"
```

> LDAP over TLS/SSL (ldaps://) is deprecated in favour of *StartTLS*. The latter refers to an existing LDAP session (listening on TCP port 389) becoming protected by TLS/SSL whereas LDAPS, like HTTPS, is a distinct encrypted-from-the-start protocol that operates over TCP port 636.

Tighten up ownership and permissions:

```
sudo adduser openldap ssl-cert
sudo chgrp ssl-cert /etc/ssl/private/ldap01_slapd_key.pem
sudo chmod g+r /etc/ssl/private/ldap01_slapd_key.pem
sudo chmod o-r /etc/ssl/private/ldap01_slapd_key.pem
```

Restart OpenLDAP:

```
sudo systemctl restart slapd.service
```

Check your host's logs (/var/log/syslog) to see if the server has started properly.

Replication and TLS

If you have set up replication between servers, it is common practice to encrypt (StartTLS) the replication traffic to prevent evesdropping. This is distinct from using encryption with authentication as we did above. In this section we will build on that TLS-authentication work.

The assumption here is that you have set up replication between Provider and Consumer according to Replication and have configured TLS for authentication on the Provider by following TLS.

As previously stated, the objective (for us) with replication is high availablity for the LDAP service. Since we have TLS for authentication on the Provider we will require the same on the Consumer. In addition to this, however, we want to encrypt replication traffic. What remains to be done is to create a key and certificate for the Consumer and then configure accordingly. We will generate the key/certificate on the Provider, to avoid having to create another CA certificate, and then transfer the necessary material over to the Consumer.

1. On the Provider,

   Create a holding directory (which will be used for the eventual transfer) and then the Consumer's private key:

   ```
   mkdir ldap02-ssl
   cd ldap02-ssl
   sudo certtool --generate-privkey \
   --bits 1024 \
   --outfile ldap02_slapd_key.pem
   ```

   Create an info file, `ldap02.info`, for the Consumer server, adjusting its values accordingly:

   ```
   organization = Example Company
   cn = ldap02.example.com
   tls_www_server
   encryption_key
   signing_key
   expiration_days = 3650
   ```

   Create the Consumer's certificate:

   ```
   sudo certtool --generate-certificate \
   --load-privkey ldap02_slapd_key.pem \
   --load-ca-certificate /etc/ssl/certs/cacert.pem \
   --load-ca-privkey /etc/ssl/private/cakey.pem \
   --template ldap02.info \
   --outfile ldap02_slapd_cert.pem
   ```

   Get a copy of the CA certificate:

   ```
   cp /etc/ssl/certs/cacert.pem .
   ```

   We're done. Now transfer the `ldap02-ssl` directory to the Consumer. Here we use scp (adjust accordingly):

   ```
   cd ..
   scp -r ldap02-ssl user@consumer:
   ```

2. On the Consumer,

   Configure TLS authentication:

   ```
   sudo apt install ssl-cert
   sudo adduser openldap ssl-cert
   sudo cp ldap02_slapd_cert.pem cacert.pem /etc/ssl/certs
   sudo cp ldap02_slapd_key.pem /etc/ssl/private
   sudo chgrp ssl-cert /etc/ssl/private/ldap02_slapd_key.pem
   sudo chmod g+r /etc/ssl/private/ldap02_slapd_key.pem
   sudo chmod o-r /etc/ssl/private/ldap02_slapd_key.pem
   ```

   Create the file `/etc/ssl/certinfo.ldif` with the following contents (adjust accordingly):

   ```
   dn: cn=config
   add: olcTLSCACertificateFile
   olcTLSCACertificateFile: /etc/ssl/certs/cacert.pem
   -
   add: olcTLSCertificateFile
   ```

```
olcTLSCertificateFile: /etc/ssl/certs/ldap02_slapd_cert.pem
-
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ssl/private/ldap02_slapd_key.pem
```

Configure the slapd-config database:

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f certinfo.ldif
```

Configure `/etc/default/slapd` as on the Provider (SLAPD_SERVICES).

3. On the Consumer,

   Configure TLS for Consumer-side replication. Modify the existing *olcSyncrepl* attribute by tacking on some TLS options. In so doing, we will see, for the first time, how to change an attribute's value(s).

   Create the file `consumer_sync_tls.ldif` with the following contents:

```
dn: olcDatabase={1}hdb,cn=config
replace: olcSyncRepl
olcSyncRepl: rid=0 provider=ldap://ldap01.example.com bindmethod=simple
 binddn="cn=admin,dc=example,dc=com" credentials=secret searchbase="dc=example,dc=com"
 logbase="cn=accesslog" logfilter="(&(objectClass=auditWriteObject)(reqResult=0))"
 schemachecking=on type=refreshAndPersist retry="60 +" syncdata=accesslog
 starttls=critical tls_reqcert=demand
```

   The extra options specify, respectively, that the consumer must use StartTLS and that the CA certificate is required to verify the Provider's identity. Also note the LDIF syntax for changing the values of an attribute ('replace').

   Implement these changes:

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f consumer_sync_tls.ldif
```

   And restart slapd:

```
sudo systemctl restart slapd.service
```

4. On the Provider,

   Check to see that a TLS session has been established. In `/var/log/syslog`, providing you have 'conns'-level logging set up, you should see messages similar to:

```
slapd[3620]: conn=1047 fd=20 ACCEPT from IP=10.153.107.229:57922 (IP=0.0.0.0:389)
slapd[3620]: conn=1047 op=0 EXT oid=1.3.6.1.4.1.1466.20037
slapd[3620]: conn=1047 op=0 STARTTLS
slapd[3620]: conn=1047 op=0 RESULT oid= err=0 text=
slapd[3620]: conn=1047 fd=20 TLS established tls_ssf=128 ssf=128
slapd[3620]: conn=1047 op=1 BIND dn="cn=admin,dc=example,dc=com" method=128
slapd[3620]: conn=1047 op=1 BIND dn="cn=admin,dc=example,dc=com" mech=SIMPLE ssf=0
slapd[3620]: conn=1047 op=1 RESULT tag=97 err=0 text
```

LDAP Authentication

Once you have a working LDAP server, you will need to install libraries on the client that will know how and when to contact it. On Ubuntu, this has been traditionally accomplished by installing the *libnss-ldap* package. This package will bring in other tools that will assist you in the configuration step. Install this package now:

```
sudo apt install libnss-ldap
```

You will be prompted for details of your LDAP server. If you make a mistake you can try again using:

```
sudo dpkg-reconfigure ldap-auth-config
```

The results of the dialog can be seen in `/etc/ldap.conf`. If your server requires options not covered in the menu edit this file accordingly.

Now configure the LDAP profile for NSS:

```
sudo auth-client-config -t nss -p lac_ldap
```

Configure the system to use LDAP for authentication:

```
sudo pam-auth-update
```

From the menu, choose LDAP and any other authentication mechanisms you need.

You should now be able to log in using LDAP-based credentials.

LDAP clients will need to refer to multiple servers if replication is in use. In `/etc/ldap.conf` you would have something like:

```
uri ldap://ldap01.example.com ldap://ldap02.example.com
```

The request will time out and the Consumer (ldap02) will attempt to be reached if the Provider (ldap01) becomes unresponsive.

If you are going to use LDAP to store Samba users you will need to configure the Samba server to authenticate using LDAP. See Samba and LDAP for details.

An alternative to the *libnss-ldap* package is the *libnss-ldapd* package. This, however, will bring in the *nscd* package which is problably not wanted. Simply remove it afterwards.

User and Group Management

The *ldap-utils* package comes with enough utilities to manage the directory but the long string of options needed can make them a burden to use. The *ldapscripts* package contains wrapper scripts to these utilities that some people find easier to use.

Install the package:

```
sudo apt install ldapscripts
```

Then edit the file `/etc/ldapscripts/ldapscripts.conf` to arrive at something similar to the following:

```
SERVER=localhost
BINDDN='cn=admin,dc=example,dc=com'
BINDPWDFILE="/etc/ldapscripts/ldapscripts.passwd"
SUFFIX='dc=example,dc=com'
GSUFFIX='ou=Groups'
USUFFIX='ou=People'
MSUFFIX='ou=Computers'
GIDSTART=10000
UIDSTART=10000
MIDSTART=10000
```

Now, create the `ldapscripts.passwd` file to allow rootDN access to the directory:

```
sudo sh -c "echo -n 'secret' > /etc/ldapscripts/ldapscripts.passwd"
sudo chmod 400 /etc/ldapscripts/ldapscripts.passwd
```

Replace “secret” with the actual password for your database's rootDN user.

The scripts are now ready to help manage your directory. Here are some examples of how to use them:

1. Create a new user:

   ```
   sudo ldapadduser george example
   ```

   This will create a user with uid *george* and set the user's primary group (gid) to *example*

2. Change a user's password:

   ```
   sudo ldapsetpasswd george
   Changing password for user uid=george,ou=People,dc=example,dc=com
   New Password:
   New Password (verify):
   ```

3. Delete a user:

   ```
   sudo ldapdeleteuser george
   ```

4. Add a group:

   ```
   sudo ldapaddgroup qa
   ```

5. Delete a group:

   ```
   sudo ldapdeletegroup qa
   ```

6. Add a user to a group:

   ```
   sudo ldapaddusertogroup george qa
   ```

   You should now see a *memberUid* attribute for the *qa* group with a value of *george*.

7. Remove a user from a group:

```
sudo ldapdeleteuserfromgroup george qa
```

The *memberUid* attribute should now be removed from the *qa* group.

8. The *ldapmodifyuser* script allows you to add, remove, or replace a user's attributes. The script uses the same syntax as the *ldapmodify* utility. For example:

```
sudo ldapmodifyuser george
# About to modify the following entry :
dn: uid=george,ou=People,dc=example,dc=com
objectClass: account
objectClass: posixAccount
cn: george
uid: george
uidNumber: 1001
gidNumber: 1001
homeDirectory: /home/george
loginShell: /bin/bash
gecos: george
description: User account
userPassword:: e1NTSEF9eXFsTFcyWlhwWkF1eGUybVdFWHZKRzJVMjFTSG9vcHk=

# Enter your modifications here, end with CTRL-D.
dn: uid=george,ou=People,dc=example,dc=com
replace: gecos
gecos: George Carlin
```

The user's *gecos* should now be “George Carlin”.

9. A nice feature of *ldapscripts* is the template system. Templates allow you to customize the attributes of user, group, and machine objects. For example, to enable the *user* template edit `/etc/ldapscripts/ldapscripts.conf` changing:

```
UTEMPLATE="/etc/ldapscripts/ldapadduser.template"
```

There are *sample* templates in the `/usr/share/doc/ldapscripts/examples` directory. Copy or rename the `ldapadduser.template.sample` file to `/etc/ldapscripts/ldapadduser.template`:

```
sudo cp /usr/share/doc/ldapscripts/examples/ldapadduser.template.sample \
/etc/ldapscripts/ldapadduser.template
```

Edit the new template to add the desired attributes. The following will create new users with an objectClass of inetOrgPerson:

```
dn: uid=<user>,<usuffix>,<suffix>
objectClass: inetOrgPerson
objectClass: posixAccount
cn: <user>
sn: <ask>
uid: <user>
uidNumber: <uid>
gidNumber: <gid>
homeDirectory: <home>
loginShell: <shell>
gecos: <user>
description: User account
title: Employee
```

Notice the *<ask>* option used for the *sn* attribute. This will make *ldapadduser* prompt you for its value.

There are utilities in the package that were not covered here. Here is a complete list:

```
ldaprenamemachine
ldapadduser
ldapdeleteuserfromgroup
ldapfinger
ldapid
ldapgid
ldapmodifyuser
ldaprenameuser
lsldap
ldapaddusertogroup
ldapsetpasswd
ldapinit
```

```
ldapaddgroup
ldapdeletegroup
ldapmodifygroup
ldapdeletemachine
ldaprenamegroup
ldapaddmachine
ldapmodifymachine
ldapsetprimarygroup
ldapdeleteuser
```

Backup and Restore

Now we have ldap running just the way we want, it is time to ensure we can save all of our work and restore it as needed.

What we need is a way to backup the ldap database(s), specifically the backend (cn=config) and frontend (dc=example,dc=com). If we are going to backup those databases into, say, `/export/backup`, we could use *slapcat* as shown in the following script, called `/usr/local/bin/ldapbackup`:

```
#!/bin/bash

BACKUP_PATH=/export/backup
SLAPCAT=/usr/sbin/slapcat

nice ${SLAPCAT} -n 0 > ${BACKUP_PATH}/config.ldif
nice ${SLAPCAT} -n 1 > ${BACKUP_PATH}/example.com.ldif
nice ${SLAPCAT} -n 2 > ${BACKUP_PATH}/access.ldif
chmod 640 ${BACKUP_PATH}/*.ldif
```

These files are uncompressed text files containing everything in your ldap databases including the tree layout, usernames, and every password. So, you might want to consider making `/export/backup` an encrypted partition and even having the script encrypt those files as it creates them. Ideally you should do both, but that depends on your security requirements.

Then, it is just a matter of having a cron script to run this program as often as we feel comfortable with. For many, once a day suffices. For others, more often is required. Here is an example of a cron script called `/etc/cron.d/ldapbackup` that is run every night at 22:45h:

```
MAILTO=backup-emails@domain.com
45 22 * * *  root    /usr/local/bin/ldapbackup
```

Now the files are created, they should be copied to a backup server.

Assuming we did a fresh reinstall of ldap, the restore process could be something like this:

```
sudo systemctl stop slapd.service
sudo mkdir /var/lib/ldap/accesslog
sudo slapadd -F /etc/ldap/slapd.d -n 0 -l /export/backup/config.ldif
sudo slapadd -F /etc/ldap/slapd.d -n 1 -l /export/backup/domain.com.ldif
sudo slapadd -F /etc/ldap/slapd.d -n 2 -l /export/backup/access.ldif
sudo chown -R openldap:openldap /etc/ldap/slapd.d/
sudo chown -R openldap:openldap /var/lib/ldap/
sudo systemctl start slapd.service
```

Resources

1. The primary resource is the upstream documentation: www.openldap.org
2. There are many man pages that come with the slapd package. Here are some important ones, especially considering the material presented in this guide:

   ```
   slapd
   slapd-config
   slapd.access
   slapo-syncprov
   ```

3. Other man pages:

   ```
   auth-client-config
   pam-auth-update
   ```

4. Zytrax's LDAP for Rocket Scientists; a less pedantic but comprehensive treatment of LDAP
5. A Ubuntu community OpenLDAP wiki page has a collection of notes

6. O'Reilly's LDAP System Administration (textbook; 2003)
7. Packt's Mastering OpenLDAP (textbook; 2007)

Previous Next