July 2000
Draft Version 8 Revision 0-4

A Theoretical Approach to Synchronizing Qubes with LDAP

This Technical Note describes a basic setup and working solution for synchronizing a set of Qube systems without persistent Internet connections with a centralized LDAP server. This solution will **only** handle the addition and removal of accounts from either system. It could conceivably be expanded to work for modification of email accounts if the files containing the desired information were handled in a similar same way as `/etc/passwd`, `Passwd.0`, and `Passwd.1`.

Table of Contents

0.1 Applicable Products

This Note applies only to the Qube and Qube 2 server appliance family. All Technical Notes are located in `ftp://ftp.cobaltnet.com/pub/developer/TechNotes/`

1 Three Possible Solutions

This Technical Note lists three possible solutions. Of the three, the first has been most fully developed.

1.1 Solution 1

This file contains instructions on setting up a server-client pair to demonstrate a possible solution to using an LDAP server to provide user updates on a Qube 2, which does not have a persistent connection.

To demonstrate, assume the presence of two Cobalt Server Appliances: a RaQ 3 and Qube 2 respectively.

The first step is setting up the server side of the LDAP demonstration, which requires the installation of the OpenLDAP-1.2.9-1.i386.pkg on a Cobalt RaQ 3 Server Appliance. This can be done using the Cobalt web-based administrative graphical user interface (GUI). Please refer to your *Cobalt User's Guides* for help.

Note
It is also possible to install the server end on a Qube 2; this implementation makes use of the OpenLDAP package that was created for Cobalt's i386 products.

After completing the aforementioned software installation, log into the RaQ 3 system via telnet and `su root` to get to the `#` prompt. Edit the `/home/programs/openldap/etc/openldap/slapd.conf` file to contain an LDBM database definition of the form:

```
#######################################################################
# ldbm database definitions
#######################################################################
database        ldbm
suffix          "o=<YOUR ORGANIZATION>, c=<YOUR COUNTRY ABBREVIATION>"
rootdn          "cn=Manager, o=<YOUR ORGANIZATION>, c=<YOUR COUNTRY ABBREVIATION>"
rootpw          secret
directory       /home/programs/openldap/tmp
```

In the demonstration, our sample organization will be “Educational Services” and our country will be France, with an abbreviation of “FR”. This means that our final LDBM database definition looks like the following:

```
#######################################################################
# ldbm database definitions
#######################################################################
database        ldbm
suffix          "o=Educational Services, c=FR"
rootdn          "cn=Manager, o=Educational Services, c=FR"
rootpw          secret
directory       /home/programs/openldap/tmp
```

Please note that the directory keyword points at the directory on the local hard drive where the database and all indices will be stored. The default value is `/var/tmp`, but in this example, it has been changed to `/home/programs/openldap/tmp`, so that it is stored on the largest partition available on the Cobalt server.

This change of location does require the user to manually create the specified directory. Type the command `mkdir /home/program/openldap/tmp` at the telnet prompt after logging onto the system as *root*.

Also, notice that the *root* password is set to the word *secret*. While this is fine for testing purposes, it is strongly suggested that the password is changed to something more secure before beginning production operations.

After performing the necessary modifications to the `slapd.conf` file, start the server daemon. To do this, execute the following command from the telnet prompt:

```
su root -c '/home/programs/openldap/libexec/slapd -f /home/programs/openldap/etc/
openldap/slapd.conf'
```

Next is basic initialization of the LDAP directory. OpenLDAP uses a program called `ldapadd` to add entries and `ldapadd` expects the input to be in LDIF format. The easiest way to do this is create a file in the correct format, which will be piped to `ldapadd`.

Here is a sample LDIF file:

```
dn: o=Educational Services, c=FR
objectClass: organization

dn: district=one, o=Educational Services, c=FR
objectClass: schoolDistrict

dn: district=two, o=Educational Services, c=FR
objectClass: schoolDistrict

dn: district=three, o=Educational Services, c=FR
objectClass: schoolDistrict

dn: school=one, district=one, o=Educational Services, c=FR
objectClass: schoolCampus

dn: school=two, district=one, o=Educational Services, c=FR
objectClass: schoolCampus

dn: username=student0, school=one, district=one, o=Educational Services, c=FR
username: student0
fullname: Student 0
password: abc123
```

```
objectClass: schoolStudent

dn: username=student1, school=one, district=one, o=Educational Services, c=FR
username: student1
fullname: Student 1
password: abc123
objectClass: schoolStudent

dn: username=student2, school=one, district=one, o=Educational Services, c=FR
username: student2
fullname: Student 2
password: abc123
objectClass: schoolStudent

dn: username=student3, school=one, district=one, o=Educational Services, c=FR
username: student3
fullname: Student 3
password: abc123
objectClass: schoolStudent

dn: username=student4, school=one, district=one, o=Educational Services, c=FR
username: student4
fullname: Student 4
password: abc123
objectClass: schoolStudent

dn: username=student5, school=one, district=one, o=Educational Services, c=FR
username: student5
fullname: Student 5
password: abc123
objectClass: schoolStudent

dn: username=studentA, school=two, district=one, o=Educational Services, c=FR
username: studentA
fullname: Student A
password: abc123
objectClass: schoolStudent

dn: username=studentB, school=two, district=one, o=Educational Services, c=FR
username: studentB
fullname: Student B
password: abc123
objectClass: schoolStudent

dn: username=studentC, school=two, district=one, o=Educational Services, c=FR
username: studentC
fullname: Student C
password: abc123
objectClass: schoolStudent

dn: username=studentD, school=two, district=one, o=Educational Services, c=FR
username: studentD
fullname: Student D
password: abc123
objectClass: schoolStudent
```

Cut and paste the above text to a file named `/home/programs/openldap/test.ldif`. Then, execute the following command:

```
/home/programs/openldap/bin/ldapadd -D "cn=Manager, o=Educational Services, c=FR" \
                                    -h <HOST IP> \
                                    -w secret < /home/programs/openldap/test.ldif /
home/programs/openldap/test.ldif
```

where `<HOST IP>` is replaced by the IP address of the RaQ 3 machine that is running OpenLDAP.

A test population is now in your directory. In simplest terms, we have created three school districts under our main organization. The first of the three districts contains two schools, and those schools have six and four students respectively.

Note
We have finished with the server side of the test system, and will continue with the installation on the Qube 2.

As with the RaQ 3, start up your Qube 2 and go through the basic configuration steps. Next, install the following packages:

- `OpenLDAP-1.2.9-1.mips.pkg`
- `OpenLDAP_Demo-1.0-1.mips.pkg`

via the Cobalt software maintenance GUI.

After successfully installing both packages, telnet into the Qube 2 as the administrator and then `su` to `root` so you are at the `#` prompt. If you look at the directory listing, you will see a file named `ldap.pl`. This is the file that we will use for the demonstration.

Note
In the following paragraphs, where ever `<LDAP ADDR>` appears, replace it with the actual IP address of the RaQ 3 server that was previously set up.

Type the following command:

```
perl ldap.pl "<LDAP ADDR>" "school=one, district=one, o=Educational Services, c=FR"
```

You will see output from the program indicating that it has added a number of users to the system. You can verify this by going to the Cobalt GUI and looking at users currently listed on the system.

A quick explanation: If you remember, we previously populated the LDAP server. We, in fact, created a school “one” in district “one” that had 6 students. The LDAP server was polled and those students were added since they were not currently present on the Qube 2. This demonstrates behavior when new students are added to a school's section of the LDAP server.

For the next demonstration, use the Qube 2 interface and add one or more users to the system. After completing this task, execute the following command:

```
perl ldap.pl "<LDAP ADDR>" "school=one, district=one, o=Educational Services, c=FR"
```

You might notice that it is the same command we previously executed, more or less with the Qube saying it would like to update it's user list using the information for school one, district one, etc. This time, though, you will note that the program said it was deleting users. This can be verified, as before, through the interface.

The explanation for this action is as follows: when the polling was done against the LDAP server, the Qube found users which were present on it, but of which the LDAP server had no knowledge. As such, they were removed from the Qube. This would be similar to student(s) transferring to another school, graduating, or other exit actions.

There was some additional text output saying that there were users who could be modified. This means that the Qube 2 found students that both it and the LDAP server were aware of. It would be possible to *update* the information for every similar match on the Qube, but this can cause some problems. For example, consider the case where a student forgets their password on their school's Qube, and the local administrator assigns them a new one. Since we have no realistic way of knowing whether the Qube's or the LDAP server's copy of the student's information was most current, we could not know whether we should modify the student's account with information from the LDAP.

Now for something completely different. Type the following command:

```
perl ldap.pl "<LDAP ADDR>" "school=two, district=one, o=Educational Services, c=FR"
```

You should see the users previously created being deleted and then a new set of users being added to the system. We have told the Qube that it is a different school, and, as such, it replaced it's user list and information with that schools data. We can easily switch it back by executing our original command again:

```
perl ldap.pl "<LDAP ADDR>" "school=one, district=one, o=Educational Services, c=FR"
```

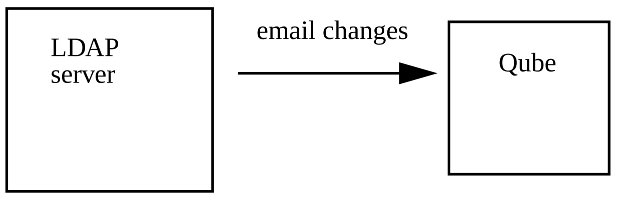
While this may seem like a nice trick there is an obvious problem with doing it (if there were ever a reason to do it, of course), which is the simple fact that when a user is deleted from the system, all of their information, that is, email, web pages, and other information, is lost. Recreating the user does not restore anything other than the default blank configuration for them.

Last but certainly not least: even though this file was being executed by hand, it is certainly possible to set up a `cron` job that executes it at a specific time every day. This would be the solution for the French school system. Also, we would need to design an interface for them and find out *exactly* how they are storing the student information (what the field names are, and others) in the LDAP servers they are using.

1.2 Solution 2

Solution 2 sends changes occurring on the Qube system to the LDAP server in the form of email and receives changes from the LDAP server on to the local Qubes.

Figure 1. Solution 2: Sending changes from the LDAP server to the Qube



The first check is done is to make certain that there is no incoming message from the LDAP server being processed (see “Proc 2: Updating from LDAP to Qube / Cobalt system” on page 6). This, in simplest form, which means that we sacrifice email account synchronization between the systems to ensure the integrity of said information. If we allowed both Process 1 and Process 2 to execute simultaneously, it is possible that we overwrite changes.

The processing itself simply takes the password file from *n* hours ago, creates a copy, and does a `diff` against the current one. This gives information on what account(s) (user and inclusively, email) have been added or removed in the previous *n* hours. The file and data stream created is then sent via email to the LDAP server. Immediately after it is created and sent, Process 1 exits and does not run again until the `cron` interval.

It is at this point the responsibility of the LDAP systems to receive the email and perform the necessary modifications to it's own data.

Process 2 is responsible for receiving changes that occurred on the LDAP server and implementing them on the local Qube / email system.

The initialization step consists of creating a specific account, which is responsible for receiving message from the LDAP system. This account has a `.forward` file, which points to our executable. As a result, whenever an email is received on this account, it is streamed over to the executable where the processing is done.

The first step upon execution of Process 2 is to ensure that Process 1 is not actively running. This is done for the same reasons as in Process 1, that is, to ensure that no collisions occur due to simultaneous file modifications by the two processes.

The second step consists of stepping through the incoming email after verifying authenticity, and then removing and adding accounts to the local host as indicated. Once this is complete, the final step is overwriting of the `Passwd.0` file with the information in `Passwd.1,` and termination.

1.3 Proc 1: Updating from Qube / Cobalt system to LDAP system

Process 1 implements this methodology:

1) Does file `Passwd.0` exist? If not, create as an empty file.
2) If Proc 2 is live, then abort immediately.
3) Create a copy of `/etc/passwd` named `Passwd.1`
4) Create a `diff` of `/etc/password` file and `Passwd.0` and send with appropriate header and checksums to LDAP system.

1.4 Proc 2: Updating from LDAP to Qube / Cobalt system

In the initial setup, the process creates a special (uniform across all Qubes in system) account with a `.forward` file pointing to a parsing script to be executed when email comes in under the account. It follows these rules:

1) If Proc 1 is live, then abort immediately.
2) Process file adding and modifying entries in local (Qube) email system.
3) Move `Passwd.1` to `Passwd.0.`

Note that a lot more information would in all likelihood be passed between the two machines, but the basic idea is workable considering the prevailing conditions such as non-persistent connections, desirability of a low modification factor for the user interface and controls, and so forth.

1.5 Solution 3

Solution 3 implements receiving changes that occurred on the LDAP server and implementing them on the local Qube.

The third method could work very similarly to the first but would have to be written in C or Java. It would consist of a single process written in one of the these languages, which would do the same sort of `diff` between previous and existing information records, and then make a true stream connection to the LDAP server to transmit those changes immediately.

This does have the benefit of probably being more robust and allowing faster modification and updates of information between the Qube(s) and the LDAP server, but would require someone fairly familiar with LDAP and the programming interfaces for it. Also, since it requires a direct connection to an LDAP server for Q.A. testing, we would have to set up an LDAP system in house that mimicked the France Telecom system. Final testing would have to wait until they had a production LDAP system completely in place. This would not be necessary with the first method since we could easily send test emails from any machine, as we are abstracted from LDAP.