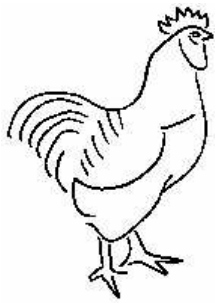
CLTC Documentation Sheet 5:

Configuring Remote Server Access



Developed by The 'Free Range' Community–Linux Training Centre Project -
Version 1.0, January 2003. http://www.fraw.org.uk/cltc/

No, pardon; 'tis a secret must be locked within the teeth and the lips:
but this I can let you understand, the greater file of the subject
held the duke to be wise.

Shakespeare, *Measure for Measure*, Act 3, Scene 2.

This sheet looks at configuring the main report access protocols: *remote access* (using the `rlogin`, `rsh` and `rcp` commands), `telnet` and *secure shell* (SSH) for accessing the functions of the server, and *file transfer protocol* (FTP) for moving files on an off the server. These protocols can be very useful for maintaining a local network, but they are often used for the remote maintenance of rented servers. We therefore use these on the CLTC as a means of teaching server management.

Accessing a remote machine

There are a number of Linux commands that can access the functions of another machine on the network. These functions are horribly insecure, so they are not the sort of service you would open on an Internet-connected machine. But on a private local network they can be very useful for system maintenance and configuration.

The main remote access are:

- `rlogin` (remote login) – login to the remote machine, just as if you were doing a console login on your remote machine.
- `rsh` (remote shell) – remotely execute a command on a system, returning the output form the command to your console.
- `rcp` (remote copy) – allows access to the file system of a remote machine in order to copy data to or from it.

To enable remote access you must activate the various daemons on the machine that control remote access using the `ntsysv` utility. To enable the new daemon you have to restart the `xinetd` service with the command:

```
/etc/rc.d/init.d/xinetd restart
```

The `rlogin` daemon controls remote login. `rsh` controls the remote execution of commands, such as `rsh`, `rcp`, `rsync` (a more advanced version of `rcp`) and others.

The `rlogin` command is fairly straightforward to implement. Enable the daemon and you can login to the remote system. The other remote access commands require that you configure a file within the home directory of the target user(s). This file, called `.rhosts` (see below). Note the preceding 'dot' – this is called a 'dot-file' and is hidden in the usual directory listings, unless you specifically request to see

An example `.rhosts` file (for 'alpha')

```
192.168.66.1 beta
192.168.66.1 gamma
192.168.66.1 delta
192.168.66.1 epsilon
```

the dot files. The .rhosts file must contain an entry for each user, and the machine that they are resident upon, in order to validate their access. Otherwise you will receive a terse '*permission denied*'.

Note also that, after creating a .rhosts file, you should change its permissions to make it accessible only to the local user. You do this using the command:

```
chmod 600 .rlogin
```

Once you have configure a .rhosts file not only do you activate the remote access commands. You can also rlogin without having to provide a password. Note in the example .rhosts file given on the previous page that *alpha* is not listed. This is because this file is for the *alpha* user account. If you included '*alpha*' then *alpha* wouldn't need a password to rlogin. On other user accounts you would insert an entry for alpha, and remove that of the local user.

On the CLTC we have a problem in that the client machines do not have a fixed IP. Therefore we illustrate remote access by having each client rlogin to their own accounts on the server. From there, with their fixed IP on the server, they can remote access each other's user accounts.

Configuring telnet access

telnet works in a similar way to rlogin – it enable the remote login to a server. However, because it is an access protocol rather than just a console connection to the server, it supports more functions than rlogin, and is far more secure. telnet is also more flexible in its use and acceptance of control characters.

telnet can also be used not only to log into user accounts, but to attempt transactions with a particular port on a machine. Therefore it can be used to test services (e.g., email or FTP) by manually exchanging data with them.

telnet is run by the telnetd daemon. Using the ntsysv utility you check the 'telnet' option to enable telnet access to the server.

Configuring 'secure shell' access

The most commonly used system for remotely logging in over public networks, where security is crucial, is secure shell, or *SSH*. Secure shell works like telnet, but all communications are encrypted to prevent the disclosure of security-critical information during transit over the network.

We enable the secure shell daemon, sshd, on the server using the ntsysv utility. To activate the daemon we use the command:

```
/etc/rc.d/init.d/sshd start
```

On the server side, various options can be configured. These are set in the configuration files contained in the directory /etc/ssh. However, one thing to take note of is that if you have a .rhosts file in a user's home directory, this will override authentication. For this reason you can use a file with the same format called .shosts that allows access without authentication using secure shell, but blocks attempts for access with the usual remote access commands.

Secure shell has various modes of authentication (see the ssh man page for full details). The most secure option is to set-up an encryption key for use with secure shell. This is the most secure method of logging in and transacting data – however it is a little complex to organise.

To begin with the user must generate an ssh *key pair*. To do this we use the program ssh-keygen. There are various versions of this program that work slightly differently. The example given in the box on the following page requires that you specify the type of key, whilst some versions default to a particular key type.

Assuming we must specify the key type, we run the program using the command:

```
ssh-keygen -t rsa
```

This runs the key generating program to create a 'type' (-t) RSA-2 key. The process for creating the key using this program is shown in the box on the next page.

Once the key pair has been created the 'public' half of the key must be securely transported to the server. This is not a problem on the CLTC. In real life, this would be done via encrypted

Configuring SSH for use in remote access

This is the process to create a key pair using ssh-keygen:

```
mobbsey@laptop:~/.ssh> ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mobbsey/.ssh/id_rsa): <return>
Enter passphrase (empty for no passphrase): <text hidden>
Enter same passphrase again: <text hidden>
Your identification has been saved in /home/mobbsey/.ssh/id_rsa.
Your public key has been saved in /home/mobbsey/.ssh/id_rsa.pub.
The key fingerprint is:
73:cc:20:cd:e7:5f:01:ae:7b:98:06:e3:15:8c:dc:8b mobbsey@laptop
```

The file /home/mobbsey/.ssh/id_rsa.pub must now be securely transported to the server and renamed authorized_keys2.

This is the process to login using ssh once the key has been installed:

```
mobbsey@laptop:~/.ssh> ssh cltc.lan
Enter passphrase for key '/home/mobbsey/.ssh/id_rsa': <text hidden>
Last login: Sat Jan 11 03:54:11 2003 from 192.168.66.5
[mobbsey@epsilon mobbsey]$
```

email, or using an FTP program that 'wrapped' the information using the SSH protocol (under Linux, there is a program that does this called sftp, although its a command-line only program).

Once the key is installed in the remote home directory it must be appropriately renamed. This is dependent upon the type of key you have created. In the example above, because we are using an RSA-2 key, we rename the file, *authorized_keys2*.

Now we can login. We do this with the command:

```
ssh cltc.lan
```

If the key is improperly installed you will be asked for the password for the remote account. Failure is usually due to not properly naming the key at the remote end. If all's well you'll be asked for the key passphrase (see example above). After entering the passphrase, if it is correct, you are logged in.

For an example of what ssh does, see the box on the right. This shows the difference in the datastream between an rlogin or telnet based login and an ssh login. As you can see the information is completely scrambled. This is very important when passing secure information over the connection to the remote server. For example, for security reasons, the

The effect of using *ssh* vs *rlogin*

For the following experiment, a directory listing (the ls command) was requested remotely, and the data packets were 'sniffed' from the network using the *ethereal* network utility:

The actual text received at the console was:

```
Desktop  dhcpd.conf  host.conf
hosts  httpd  ircd-2.10.3-
1.i386.rpm  ircd.conf  named
named.conf  outgoing  ssh
```

The packets from the rlogin connection (with colour control characters removed) contained:

```
Desktop  dhcpd.conf host.conf
hosts  httpd ircd-2.10.3-1.i3
86.rpm  ircd.conf  named named.conf
outgoing
```

The packets from the ssh connection contained:

```
.@Ð.¥!..¤ÂlJ..E..d.o@.@..¾À¨B.À¨B..
..4.¶...jQs.)à.........×..."Ù..ÚÁÓ
Æ.½O...@.ìõ...z.@ý-+%..h3ËjýwT.
"¢Ûý.Cã..{
```

root user is not allowed to log in remotely. We therefore have to log in from an ordinary user account and then use the su command to give ourselves root access. However, this requires that we send the root password over the public network. SSH prevents disclosure of the root password.

Configuring FTP services

When logged into a server we can issue commands. But we can't easily move files. We can use rcp, as outlined earlier, but this is not a secure service to run on an Internet-connected server. Therefore, in addition to login services, we have to configure the File Transfer Protocol (FTP) service to allow the movement of data securely between two computers.

FTP is important for server management. But it is also widely used online as a means of maintaining a large store of files for public access. However, instead of *user-based FTP*, these systems use *anonymous FTP*. We configure both types of FTP service on the CLTC.

For user-based (or 'real' mode) FTP, with Red Hat Linux, we use the wu-ftp (Washington University FTP) daemon, although there are a number of others available. For anonymous FTP we use the anonftp package which contains additional programs to run the anonymous FTP service alongside wu-ftpd. The standard 'out-of-the-box' configuration for wu-ftpd works adequately for real-mode FTP. This allows each user to access their own home directory via FTP. What we have to do is configure the system to accept anonymous FTP.

The anonymous FTP system has its base within the /var/ftp directory. Within this directory is a subdirectory called pub which, by convention, contains the 'public' FTP files. Now we have to create an upload directory for the anonymous FTP system. This is quite complex because we have to create a user account to be responsible for the administration of the upload directory. Therefore we add a new user called *ftpadm*. Then we create the upload directory as part of the /var/ftp directory and assign the correct permissions to it.

We do all this by entering the following commands:

```
useradd ftpadm
mkdir /var/ftp/upload
chown -R ftpadm.ftpadm
       /var/ftp/upload
chmod -R 3773 /var/ftp/upload
```

The final stage is to edit the FTP configuration file, /etc/ftpaccess. As a general housekeeping change, first of all scroll down and find the 'email address' section. Then change *root@localhost* to *root@cltc.lan*.

Next, we have to enable anonymous FTP transactions. First of all find the line –

```
class all real,guest,anonymous *
```

and change it to –

```
class user real,guest *
```

Now, below the line you have just amended, insert the following three lines:

```
class anonftp anonymous *
upload class=anonftp /var/ftp
     /upload yes ftpadm ftpadm
          0440 nodir
noretrieve /var/ftp
allow-retrieve /var/ftp/pub
```

Now, for testing purposes, we need to create a file to download using the command:

```
echo "1234" > /var/ftp/pub/testfile
```

Now you can open an FTP client program and try opening a connection in real mode, providing the real name and password of an account on the server.

Next, try to connect using the user name '*anonymous*' and a password that is a valid email address such as '*email@address.org*'. You should now see a list of directories. You should be able to enter the pub directory and download the file called '*testfile*'.

Now go back to the directory where your started, and then enter the directory called upload. You should now be able to upload '*testfile*' to this directory, but the moment the transaction is complete you will not see it stored there (this is a feature of anonymous upload directories to prevent nasty files being

deposited in pubic places).

Installation of the FTP server is now complete.