TELE301 Lab16 - The Secure Shell

Department of Telecommunications

May 7, 2002

Contents

1 Introduction

Today we learn the powerful voodoo that is the realm of the Unix Network Administrator, muhahahaha. This is more commonly known as SSH, the secure shell, and for Unix/Linux admins, is often the tool de'jour for administering servers and clients. Sadly, we're still stuck with other methods for network equipment such as routers etc, that aren't as secure, or fun `:^)`

Since most of the interesting things to do with SSH is as the user, you will do all of your work today on your workstations. That way, we can more easily play with things like X-Forwarding.

2 OpenSSH

The SSH server process is started simply by running the `sshd` process as root. Note that if you restart the server, or even HUP the server, the clients will be disconnected.

3 Replacing Telnet

The most basic use of ssh is much like telnet. This will be you're first encounter with `ssh`. Log into your server using this command.

```
client$ ssh server
# If your username is different, use this way
client$ ssh user@server
```

4 Logging in without a password

Okay, that was easy enough. Let's get onto logging in without using a password. First, we need to create a key for ourselves.

```
client$ ssh-keygen -b 1024 -t dsa
# Make sure to give a good passphrase
```

Okay, now that we have a public and private key, we need to put the key into our `~/.ssh/authorized_keys2` file. Since your home directories are shared on the TELE network, you can just put the *public* key into that file on your current workstation. It'll then be available for the ssh server to read. You may need to create that file.

Great, now we need to test it.

```
client$ ssh server
Enter passphrase for ~/.ssh/id_dsa:
```

Remember, this is the passphrase you chose when creating the key. Don't create private keys without passphrases unless you absolutely have to. Also remember that this is asking for the *local* private key's passphrase, not the remote end (like with a password).

You should now have access. Isn't ssh fun? No? Well we'll see later on.

5 SSH Agent

Entering passphrases can be a royal PITA (if you don't know what that means, look it up, its a standard computer industry acronym). Lets employ `ssh-agent` to give the passphrase to `ssh` when it asks for it. `ssh-agent` starts its first parameter, with some environment variables set so programs run from that child shell will know where to find the agent. By using exec, we replace the current running shell with an `ssh-agent` enabled one.

```
client$ exec ssh-agent bash
```

5.1 SSH Add

Right now, the agent has no keys, to add, delete, and list keys, we can use the `ssh-add` program.

```
# This will list the fingerprints (hashes) of loaded keys.
client$ ssh-add -l

# This will add a key. By default, it should try to load
# id_dsa, id_rsa, and identity in ~/.ssh. You may need
# to tell it what key to load explicitly though.
client$ ssh-add
Enter passphrase for ~/.ssh/id_dsa:

# List the keys again
client$ ssh-add -l

# We can delete all keys
client$ ssh-add -D

# If stdin cannot be read, a GUI program ssh-askpass
# will be run. Unfortunately, Slackware doesn't ship with
# ssh-askpass, but you can run it in an xterm if you need
# to.
client$ ssh-add < /dev/null
```

Okay, now that the agent is again loaded, lets try logging in without using a password. Now's a good chance to replace the r-commands.

6 Replacing the Berkeley r-commands

All the Berkeley r-commands used host-based authentication, and did not request a password. Although SSH can do host-based authentication, using the Berkeley mechanism, or a more secure cryptographic mechanism, you should use user based authentication instead. We can now do this, since we've got passwordless access.

`rlogin` was just like telnet, without a password.

```
client$ ssh server
```

To run a non-interactive command, just supply the command at the end of the command to start ssh. Here we start the `w` program on "server". This replaces the need for `rsh`.

```
client$ ssh server.localdomain w
```

We can extend this useful function by piping it something. Here we use `tar` on a remote machine to back up a directory, and save to a local archive. This means that no temporary (possibly very large) files are stored on the server.

```
ssh server tar -zcv -C /tmp/ forsshtar/ | \
   cat > /tmp/forsshtar.gz
```

To run an interactive command (that takes input from the user), in the old days, we might use `rexec`. Interactive terminal-based programs, like `top`, or `pine`, require a PTY (pseudo-terminal). The -t argument requests a psuedo-terminal.

```
client$ ssh -t server.localdomain top
```

To copy a file, we might use `rcp`. We can use `scp` to do this instead. Remember the ':', else you'll end up with a file called user@host. Recursive copying is possible, for directories, using the -r parameter.

```
client$ scp user@host:/etc/hosts /tmp
```

7 sftp

FTP is not a protocol that can have cryptography wrapped around it, due to it dual-stream nature. We can use `sftp` to provide the feel of a very basic looking ftp client, though it is, in no way, the FTP protocol.

```
client$ sftp server
# Type '?' if you don't know how to use ftp.
```

8 X Forwarding

This trick was useful to people who wanted to work on their assignment when they couldn't access the lab. Lets say you want to use OpenOffice.org on the TELE network, but you're on a different Linux box (or Windows box with CygWin, OpenSSH, and XFree86 installed).

We can use X-Forwarding to use GUI programs over the network. Remember, this is what X was designed for. By tunnelling it though ssh, we remove all the nasty-ness of the unencrypted X11 protocol. We can explicitly request X forwarding using the -X (capital X) to ssh.

```
# Don't do this first one if you're already in
# the TELE network, as you are for todays lab.
ssh -X crosstalk.otago.ac.nz
ssh -X someworkstation.localdomain
swriter &
# OR
scalc &
```

REMEMBER: If you print, you'll print to the TELE printer. This is a common trap to fall into. It is possible to use port forwarding to forward jobs over the network to the client though.

9 Basic Local Port Forwarding

Port forwarding is one of the most complex parts of SSH. You can forward either local or remote ports. If you forward the local port, the server (which accepts connections) will be on the remote side, and if you forward the remote port, the server will on the local side. Confused? Don't worry, only really tricky people generally use this. Most often local forwarding is used.

As an example of local forwarding, we'll have a very simple server. To create our server, we will use a tool called NetCat (`nc`). This is a program that does lots of things network related, one thing it can do is to listen for a connection, and then send data. Lets say we want to send data over a secure connection (via a ssh tunnel) to a client.

I'll go over port forwarding as a demo, then you can have a go. Don't worry too much if you don't understand the finer points, its not something that is easy to understand the first time. This is probably because of the awkward syntax.

```
# This makes it so that each user has a different port.
server$ PORT=$((10000+‘uid -u‘))
server$ echo $PORT
# This will send the output of the uptime program to
# the network client.
server$ uptime | nc -l -p $PORT &
# This will show the ports that are being listened to.
# Make sure $PORT is listed.
server$ netstat -at

# Test it to begin with, without tunnelling.
client$ telnet server PORT # Change PORT to whatever it
                           # is on the server
# You should have received the output of uptime.
# You can use ^], then ‘‘quit’’ to exit.
# You could also use nc for the client
# nc server PORT

# Restart nc, since it exits once it finished sending
server$ uptime | nc -l -p $PORT

# From your client, log in via ssh and setup up a
# forwarding. Change PORT to the port on the server
# Let LOCALHOST be a high numbered port that has
# nothing listening on it.
client$ ssh -L $LOCALPORT:localhost:$PORT server

# From another xterm on the client, test the forwarded
# connection.
client$ netstat -at
client$ telnet localhost $LOCALPORT

# You should have received the output of uptime.
# Log out from your SSH session.
```

This feature can be controlled globally using the AllowTcpForwarding keyword in `sshd_config`, or using the no-port-forwarding keyword in the users' `authorized_keys2` file.

10 Agent Forwarding

Agent forwarding is useful if you log into intermediate machines to get to your destination. It allows ssh (on the intermediate machines) to ask the user's machine for the passphrase, which means that you don't need to provide passphrases when using the intermediate machines.

To allow this to work, you put your public key (for the account on your workstation), into the `authorized_keys2` file on all of the intermediate and end machines. You then use the -A argument to `ssh` to enable agent forwarding.

Don't worry about practicing this, its not something that is use a great deal. Although it is very useful, just like port forwarding.

This feature cannot be disabled globally without disabling Public Key Authentication. It can be disabled on a per-key basis using no-agent-forwarding.

11 Other server configuration parameters

For your assignment, you will need to setup an SSH server. You should now read the `sshd` man page. A light skim to aquaint yourself with it should suffice. Man pages are there for reference, you don't need to memorise them.

12 TCP Wrappers

You can control (if it was compile with this feature, most are) what hosts may access `sshd`, using TCP Wrappers. SSH is commonly open to the world, depending on the server, of course. You should decide on an access policy, whether people can access any machine from the internet, or whether they must first log into another machine.

```
sshd: ALL
```

13 Resources

There is a lot of information on SSH, and the various ways to use it that the budding guru should know about. Here are some I know of.

- "SSH: The Definitive Guide", by Daniel J. Barrett & Richard E. Silverman. Published by O'Reilly. ISBN: 0-596-00011-1

- www.openssh.org
- www.cygwin.com
- Manual pages: ssh(1), sshd(8), ssh-agent(1), ssh-add(1), scp(1), sftp(1), sftp-server(8)
- “A Backup System Using SSH”, by TheLinuxAngel homepages.paradise.net.nz/ cameronk/docs/
- “Using SSH Port Forwarding to Print at Remote Locations”, by Rory Krause. Linux Journal, February 2002. www.linuxjournal.com