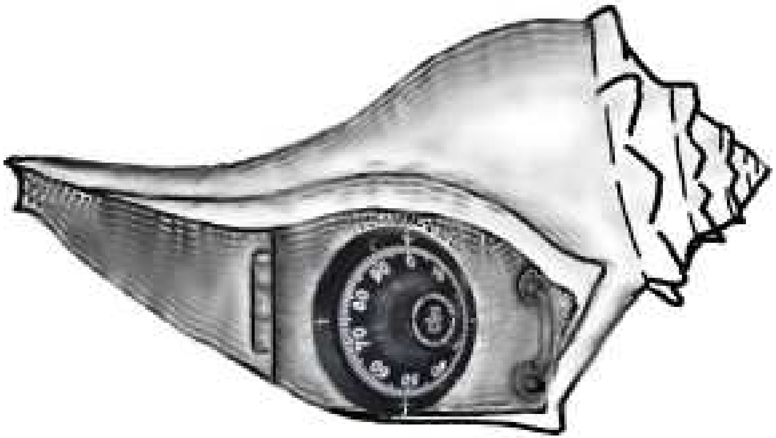

Chapter 10

Advanced SSH Configuration

OpenSSH is the encrypted network service everyone uses to do remote logins and network copying. We've already used it several times to do file transfer and the like. There is a lot more to OpenSSH than that though, and we want to now show you some of the advanced features of this wonderful tool.

10.1 Installing and Configuring SSH

On `apt` systems like Debian and Conectiva do:

```
<root>: apt-get install ssh-server
```

On RedHat do:

```
<root>: cd /mnt/cdrom/RedHat/RPMS/
<root>: rpm -Uhv ssh-server*
  -or-
<root>: up2date  ssh-server
```

If you need to compile OpenSSH, you will need to get and compile OpenSSL also. These can be found on OpenSSL.org and OpenSSH.com. First we must compile and install OpenSSL (always compile as a regular user and then install as root, unless instructed otherwise):

```
<bash>: tar xzvf openssl-0.9.6c.tar.gz
<bash>: cd openssl-0.9.6c ; ./config ; make ; su
 Password: *******
<root>: make install ; exit
```

SAMPLE
October 6, 2004 -- 23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

Now make the SSH software:

```
<bash>: tar xzvf openssh-3.4.6c.tar.gz ; cd openssh-3.4.6c
<bash>: ./configure --prefix=/usr --sysconfdir=/etc
<bash>: make ; su
Password: *******
<root>: make install ; exit
```

10.1.1 Generating the System Host Keys

Each ssh server has a host key in addition to any user keys generated. The system's public key is transferred to the user and placed in the /home/joe/.ssh/known_hosts or /home/joe/.ssh/known_hosts, depending on whether the key is protocol 1 or protocol 2 respectively. Host keys contain no passphrases.

The format for /home/joe/.ssh/known_hosts contains the following elements, each space separated:

```
hostnames bits exponent modulus comment
```

If for some reason the host key becomes damaged or compromised, you can create or re-create them in the following way:

```
<root>: ssh-keygen -t rsa -f /etc/ssh/ssh_host_key -N ""
<root>: ssh-keygen -t dsa -f /etc/ssh/ssh_dsa_key -N ""
```

Sometimes hosts get re-installed and the user will have the old copy of the host key and get this error message:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!               @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
Add correct host key in /home/joe/.ssh/known_hosts to get rid of
this message. Offending key in /home/joe/.ssh/known_hosts:4
```

Figure 10.1: Possible SSH Man-In-The-Middle Attack.

You probably want to edit your /home/joe/.ssh/known_hosts and remove the entry for the server you wish to connect to. Then you can retry your connection.

SAMPLE October 2004 -- 2:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.1.2 Server Configuration

The server configuration that we will strive for in this section is the least permissive possible while still allowing connections we need. With this in mind, we want other hosts to connect to us, use X11 forwarding, and use only SSH protocol 2 (the most secure). Below is a sample of the most relevant server configuration entries in /etc/ssh/sshd_config:

```
Port 22
Protocol 2
PermitRootLogin no
StrictModes yes      # StrictMode checks for file permissions etc.
X11Forwarding yes
X11DisplayOffset 10
DSAAuthentication yes
RSAAuthentication no # since it only works for PV 1.
RhostsAuthentication no
IgnoreRhosts yes     # ignor user's ~/.rhosts and ~/.shosts
```

Figure 10.2: An Abbreviated SSH Server Configuration File

Finally, start your server in the traditional way for your distro, e.g.

```
<root>: /etc/init.d/sshd start
```

10.1.3 Client Configuration

SSH client configuration is controlled by /etc/ssh/ssh_config and should only permit protocol version 2 and allow X11 forwarding. The entries to change are obvious, so go ahead and make these now. Note that the default values are commented out.

```
Protocol 2
ForwardX11 yes
#   RhostsAuthentication no
#   RhostsRSAAuthentication no
RSAAuthentication no
#   PasswordAuthentication yes
#   CheckHostIP yes
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/id_dsa
```

Figure 10.3: An Abbreviated SSH Client Configuration File

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.2 SSH Client Use

There are a few preliminaries to get out of the way. First stop is key generation.

10.2.1 Generating SSH Client Keys

Before we can really use the power of SSH, we need to generate keys for the user account. We do this by using the `ssh-keygen` command.

```
<bash>: ssh-keygen -d
Generating public/private dsa key pair.
Enter file in which to save the key (/home/joe/.ssh/id_dsa):
Enter passphrase (empty for no passphrase): **********
Enter same passphrase again: **********
Your identification has been saved in /home/joe/.ssh/id_dsa.
Your public key has been saved in /home/joe/.ssh/id_dsa.pub.
The key fingerprint is:
da:07:3f:49:a3:b6:65:da:06:c7:53:9f:b6:28:06:0f joe@astro.marx
```

You should now see the public and private dsa keys `~/.ssh/id_dsa.pub` and `~/.ssh/id_dsa` respectively:

```
<bash>: ls -l .ssh/
total 8
-rw-------    1 joe  users   736 Apr 28 23:30 id_dsa
-rw-r--r--    1 joe  users   609 Apr 28 23:30 id_dsa.pub
```

The file `~/.ssh/id_dsa.pub` contains the DSA public key for authentication. The contents of this file should be added to `~/.ssh/authorized_keys` on all machines where the user wishes to log in using public key authentication. There is no need to keep the contents of this file secret.

Once distributed, ssh will perform a random number challenge based on the public key, which the originating client must decrypt using the private key. This all happens transparently to you, but the result is that no passwords or passphrases are directly sent over the network, even in encrypted form.

SAMPLE October 6, 2004 -- 23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

Let us test this by copying ~/.ssh/id_dsa.pub to your remote server and putting the contents into the ~/.ssh/authorized_keys files:

```
<bash>: scp ~/.ssh/id_dsa.pub server.marx:.ssh/authorized_keys
Are you sure you want to continue connecting (yes/no)? yes
Permanently added 'server.marx,192.168.0.109' (DSA) to known hosts
joe@server.marx's password:
id_dsa.pub           100% |******************|   609       00:00
```

Now we can test this out by trying to ssh into the server. We expect now to be asked for the ***passphrase*** and NOT the password (note we change the prompt to emphasize the user and host):

```
<joe@localhost>: ssh server.marx
Enter passphrase for key '/home/joe/.ssh/id_dsa': ********
<joe@server>: hostname
server.marx
```

It works! So far so good. Lets do some exercises and then talk about typcial ssh applications.

Ex 10.58: Why is passphrase authentication better than password authentication?

Ex 10.59: (hard) Use ethereal to try to see what is being sent in the ssh challenge. Report your findings below.

Now that the heavy lifting is out of the way, we can finally use SSH to do some encrypted communications. We will go through the most common uses in their order of complexity.

10.2.2 Typical SSH Use

Ex 10.60: The simplest of commands is to connect to remote host as before:

```
<joe@localhost>: ssh server.marx
Enter passphrase for key '/home/joe/.ssh/id_dsa': ********
```

Ex 10.61: Next we connect to remote as another user using user@server.marx. Ask your neighbor to set up an account for mark with password "mark". Note that since you dont have your public key there yet, you have to use a password.

```
<joe@localhost>: ssh mark@server.marx
mark@server.marx's password: ********
```

Install your public key on mark@server.marx as above and try again using passphrases.

SAMPLE
October 6, 2004--23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.2.3 Secure Copy

Secure Copy is part of the SSH utilities that allows encrypted copying between networked systems. It has the power to copy single files or full directories. The general syntax is

```
scp user1@source.host:/path/of/file user2@destination.host:/path/of/source
```

If user1 and user2 have the same login on both machines and it the same as that of the local machine, the names can be omitted.

The options that seem the most useful are

- ⋆ -p : Preserve modification times, access times, and modes
- ⋆ -r : Recursively copy entire directories.
- ⋆ -v : Be verbose for debugging.

These exercises outline some typical usage.

Ex 10.62: This one copies a single file from the server.marx to local:

```
<bash>: scp server.marx:/tmp/acl.tgz .
acl.tgz                                  100%  579KB 390.9KB/s   00:01
```

Ex 10.63: Copy a directory from server.marx to local, and preserve all attributes:

```
<bash>: scp -rp server.marx:/tmp/www .
blacklist.txt                             100% 6160    73.4KB/s   00:00
goldlist.txt                              100%    9     0.1KB/s   00:00
```

Ex 10.64: Same as above, but move the destination to `/tmp`:

```
<bash>: scp -rp server.marx:/tmp/www /tmp/
blacklist.txt                             100% 6160    73.4KB/s   00:00
goldlist.txt                              100%    9     0.1KB/s   00:00
```

Ex 10.65: Same as above, reverse the role of source and destination: This copies the file `/tmp/xyz` to the server's `/tmp` dir:

```
<bash>: scp -rp /tmp/xyz server.marx:/tmp/.
blacklist.txt                             100% 6160    73.4KB/s   00:00
goldlist.txt                              100%    9     0.1KB/s   00:00
```

SAMPLE
October 6, 2004 23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.3 The SSH Agent and Keychain

Typing passphrases every time you need to connect to a server is a real hastle. Why not just create a service that does that for you?

10.3.1 Manual Use of SSH Agent

Well there is such a service; its called `ssh-agent`, and it works with the `ssh-add` which adds keys to the agent. Here is an example:

```
<joe@localhost>:  ssh-agent /bin/bash
<joe@localhost>:  ssh-add
 Enter passphrase for /home/joe/.ssh/id_dsa: ********
 Identity added: /home/joe/.ssh/id_dsa (/home/joe/.ssh/id_dsa)
 Identity added: /home/joe/.ssh/identity (joe@local.marx)
<joe@localhost>:  ssh remote.marx
<joe@server>:
```

10.3.2 Using Keychain to Manage SSH Agent

What we did before with `ssh-agent` is better, but it suffers from several afflictions. First, it is confined to the shell we were using, so we would have to start it in each shell. Secondly, `ssh-agent` won't work with cron jobs since cron doesn't inherit your environment, and won't let you type in passphrases. Cron is too important to ignor, especially for admins.

The fine makers of Gentoo Linux have created a program that allows you global access to `ssh-agent` information. The program is called ***keychain***, and it asks you for your passphrase one time only and can then use it in every shell you start afterwards. Thus, you only have to type in your passphrase one time per login and also eliminates muliple copies of `ssh-agent` from running concurrently.

To use keychain, you must first invoke it with your private keys as arguements:

```
<joe@localhost>:  keychain ~/.ssh/id_dsa
 KeyChain 2.0.2; http://www.gentoo.org/projects/keychain
 Copyright 2002 Gentoo Technologies, Inc.; Distributed under the GPL
 * Found existing ssh-agent at PID 26744
 * 1 more keys to add...
 Enter passphrase for .ssh/id_dsa: *************
 Identity added: .ssh/id_dsa (.ssh/id_dsa)
```

SAMPLE October 6, 2004 23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

What ***Keychain*** did is to write the environment variables needed for authentication (via `ssh-agent`) in `~/.keychain/$HOSTNAME-sh` if you use bash and `~/.keychain/$HOSTNAME-csh` if you use tcsh. This is what you see when you list the directory:

```
<joe@localhost>: ls -l .keychain
total 8
-rw-------    1 joe joe       80 Apr 29 23:29 local.marx-csh
-rw-------    1 joe joe      110 Apr 29 23:29 local.marx-sh
```

Now all you have to do is put 2 lines into your startup scripts, and enter your passphrase one time! Here is a sample `~/.bash_profile` that has keychain enabled:

```
#! /bin/bash
# First invoke keychain on your private keys:
/usr/bin/keychain ~/.ssh/id_rsa ~/.ssh/id_dsa
# Next, source the files that keychain created:
source ~/.keychain/$HOSTNAME-sh
.... other stuff ....
```

Figure 10.4: Sample Keychain Code in `.bash_profile` for Bash

Here is the similar lines for ***tcsh***:

```
#! /bin/tcsh
/usr/bin/keychain ~/.ssh/id_rsa ~/.ssh/id_dsa
# Next, source the files that keychain created:
source ~/.keychain/$HOSTNAME-csh
.... other stuff ....
```

Figure 10.5: Sample Keychain Code in `.login` for Tcsh

Exercises

Ex 10.66: Install and setup keychain for your account.

Ex 10.67: Test keychain by logging out and back in.
Can you ssh to server.marx now without entering a passphrase?
If not, something went wrong above.

Ex 10.68: Add keychain support to your fetchmail script above and test it (See below).

SAMPLE
October 6, 2

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

```
#! /bin/bash
source ~/.keychain/$HOSTNAME-sh > /dev/null
/usr/bin/fetchmail -v
```

Figure 10.6: Sample Fetchmail Script with Keychain Support

10.3.3 Agent Forwarding

One security flaw with SSH agent is that if an intruder gets root access on the system, your decrypted keys can be gotten from ssh-agent. This will compromise your network.

One way to avoid this is to only use `ssh-agent` from trusted (secure) hosts via ***SSH Agent Forwarding***. Agent forwarding works by allowing remote ssh sessions to contact ssh-agent running on a secure host. You implement this by changing a line in local.marx's `/etc/ssh/ssh_config` file:

```
ForwardAgent Yes
```

Figure 10.7: Agent Forwarding in `/etc/ssh/ssh_config`

Assume we have 3 machines: local.marx server.marx and untrusted.marx. We want to first ssh to server.marx, and then ssh into untrusted.marx without entering passphrases. For this to work correctly, your public key from local.marx must be on server.marx and on untrusted.marx.

SAMPLE, October 6, 2004--23:19

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

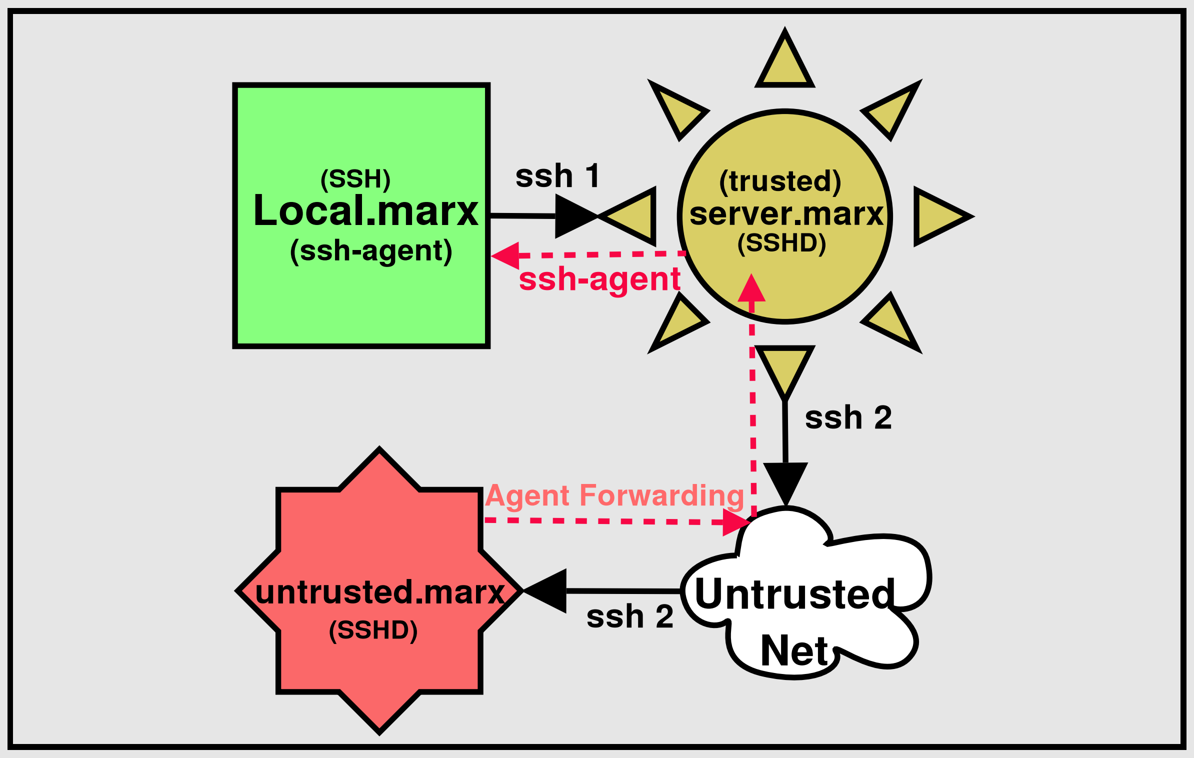

Figure 10.8: SSH Agent Forwarding

Since untrusted.marx is insecure, we don't want to expose our keys to it ever, nor expose our ssh-agent to it. This accomplishes that goal. We test our system by ssh'ing to server.marx, then to untrusted.marx, making sure no passphrase or passwords are asked for.

```
<joe@localhost>: ssh server.marx
 Welcome to server.marx!
 Last login: Fri May  2 23:03:23 2003 from 192.168.1.9
<joe@server>: ssh untrusted.marx
 Welcome to untrusted.marx!
 Last login: Fri May  2 22:09:39 2003 from 192.168.1.20
```

If you have to enter any passwords or passphrases, we have a problem.

Exercises

Ex 10.69: Make sure your Agent Forwarding works as described. Work with your peers and their admins to make sure they have the right settings.

Ex 10.70: Discuss scenarios that use Agent Forwarding to secure your network at work or home.

SAMPLE -- October 6, 2004 --

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

Ex 10.71: Draw a diagram of your proposed network with Agent Forwarding indicated.

10.3.4 Dangers of Agent Forwarding

Agent Forwarding is no panacea. It has several drawbacks from a security point of view. The following is paraphrased from the September 25, 1999 man page of ssh (1):

```
The options are as follows:
-A Enables forwarding of the authentication agent connection.

  Agent forwarding should be enabled with caution.  Users with the
  ability to bypass file permissions on the remote host (for the agent's
  Unix-domain socket) can access the local agent through the forwarded
  connection.  An attacker cannot obtain key material from the agent,
  however they can perform operations on the keys that enable them to
  authenticate using the identities loaded into the agent.
```

Figure 10.9: SSH manpage on Agent Forwarding Insecurities.

As with any technology, it must be used with respect and caution.

SAMPLE October 6, 2004-- 23:

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.4 SSH X11 Forwarding

By far, the easiest way to use X service on a remote host is via `ssh`'s port forwarding mechanism. For this, `ssh` creates an encrypted tunnel between the remote and local host. Then the remote X server looks like its local.

To activate X11 forwarding, the server on the remote side must have that feature enabled. The configuration file is normally `/etc/ssh/sshd_config`. The 2 important configuration keys are at the bottom:

```
# This is ssh server systemwide configuration file.
Port 22
Protocol 2,1
ListenAddress 0.0.0.0
HostKey /etc/ssh/ssh_host_key
HostDsaKey /etc/ssh/ssh_host_dsa_key
PermitRootLogin no
StrictModes yes
X11Forwarding yes
X11DisplayOffset 10
```

Once these are set you can ssh into the remote host and fire up your X11 applications with impunity. Here is an example:

```
<bash>: ssh -X chico.marx
Last login: Sun Mar  2 21:50:23 2003 from groucho.marx
<chico>: xterm &
<chico>: echo $DISPLAY
chico.marx:10.0
```

And it is all encrypted! **How cool is that?**

Ex 10.72: Use the X11 forwarding to login to the remote server and start an Xterm interactively on localhost. An ssh tunnel is created between the 2 machines and a local DISPLAY variable (on server.marx) is setup that connects to the originating client's X-server:

```
<joe@localhost>: ssh server.marx
Enter passphrase for key '/home/joe/.ssh/id_dsa': ********
<joe@server>: echo $DISPLAY
localhost:10.0
<joe@server>: xterm &
```

SAMPLE
October 6, 2004

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

You should now have an xterm that is from joe@server.marx. Notice that the DISPLAY variable is strangely set to a local value of server.com. How then does it show up in your client box? Discuss with others and do some investigation. Once convinced, remove the foreign xterm.

10.4.1 Dangers of X11 Forwarding

As usual, each technology has its weaknesses that can be exploited by those with the ability and desire. X11 Forwarding, if cracked, can lead to every conceivable problem that is inherent in X11 itself including every sort of X11 highjacking. Here is a section paraphrased directly from the SSH manual page:

```
The options are as follows:
  -x  Disables X11 forwarding.
  -X  Enables X11 forwarding. Use with caution. Users with the
      ability to bypass file permissions on the remote host can access
      the local X11 display through the forwarded connection. An attacker
      may then be able to perform activities such as keystroke
      monitoring.
```

Figure 10.10: Options Section of the SSH Manual.

Also, users must be carefull not to set the DISPLAY variable by hand (unless you ***really*** know what you are doing) to any value besides the default value assigned by ssh. From another part of the same ssh manual, again paraphrased:

```
ENVIRONMENT
   Ssh will normally set the DISPLAY environment variable which indicates
  the location of the X11 server. It is automatically set by ssh to point
  to a value of the form ‘‘hostname:n’’ where hostname indicates the host
  where the shell runs. The user should normally not set DISPLAY explicitly,
  as that will render the X11 connection insecure (and requires manually
  copying of any required authorization cookies).
```

Figure 10.11: Environment Section of the SSH Manual.

As always, use this technology with exteme caution.

SAMPLE October 2004-2

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.4.2 Exercises

Ex 10.73: Display the xauth information on server.marx. What can you conclude about xauth and X11 forwarding if anything?

Ex 10.74: Log into your server's machine with X11 Forwarding enabled.

Ex 10.75: Start an xterm from the remote machine. Check that the DISPLAY variable on the remote box is set correctly.

Ex 10.76: Think of 2 ways X11 Forwarding can be exploited to obtain system passwords.

Ex 10.77: How might you guard against keystroke monintoring in the case that X11 forwarding is compromised?

Ex 10.78: Search the security page for X11 exploits at OpenSSH's site: `http://www.openssh.org/security.html`.

Ex 10.79: Search the security page for Agent Forwarding exploits at OpenSSH's site: `http://www.openssh.org/security.html`. Discuss issues with your neighbor.

SAMPLE
October 6, 2004-- 23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.5 SSH Tunneling

An SSH tunnel is usually an ssh session from a local port to a remote server port. Once established, you can communicate to that server via your localhost/port numbers instead of direct communication. This is very usefull for creating secure services out of normally insecure ones.

To tunnel in SSH, you use the -L option as follows:

```
ssh -L LocalPort:RemoteHost:RemotePort  [user@]host
```

To test this, we will tunnel the servers SMTP port 25. First start the tunnel:

```
<joe@localhost>: ssh -L 1234:server.marx:25  server.marx
```

Once you are logged into server.marx, open another local xterm and use telnet like this:

```
<joe@localhost>: telnet  localhost  1234
Connected to localhost.
Escape character is '^]'.
220 server.marx ESMTP Postfix (Postfix-20010228-pl02)
```

The fact that you get to the mail server on server.marx indicates that you have sucessfully tunneled via SSH. Exit from this telnet session by hitting '^]' and then hit 'q' once inside the telnet command mode.

One common application for ssh tunneling is to use it to secure POP mail. Normally POP works over unsecure connections and allows hackers to see all of your mail and even your password. The tunnel you need for this application looks like this:

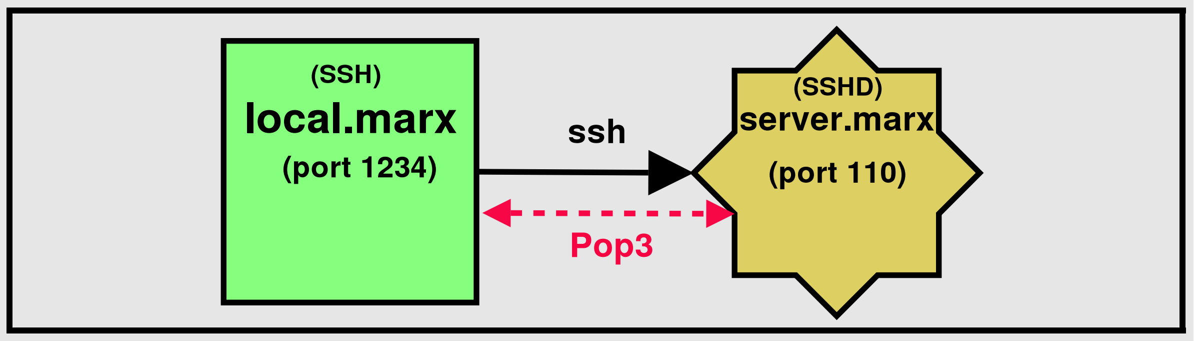

Figure 10.12: SSH Tunneling Pop3

SAMPLE
October 2004-23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

The command needed for this is:

```
<joe@localhost>: ssh -L 1234:server.marx:110 -N server.marx
```

In the other terminal you can start your usual pop client which is pointed to localhost:1234. If you dont use Netscape or Mozilla you can use the `fetchmail` program's ***preconnect*** directive, which can automate the tunnel for you in its ~/.fetchmailrc file:

```
<bash>: vi .fetchmailrc
```

```
#  Sample .fetcmailrc file
#
defaults
      user joey with pass "NoWayJose" is joe here
      no rewrite
poll localhost with protocol pop3 and port 1234 timeout 100:\
preconnect "ssh -C -f joey@server.com -L 1234:server.com:110 sleep 40"
```

Figure 10.13: Sample .fetchmailrc file

Notice the backslash character which indicates line continuation.

10.5.1 Exercises

Here are some exercises to bring these concepts home.

Ex 10.80: Install a POP3 server on server.marx, and configure it for the Postfix mailer.

Ex 10.81: Install `fetchmail` if not installed already.

Ex 10.82: Create an SSH tunnel by hand and pop your test mail.

Ex 10.83: Create a `~/.fetchmailrc` file as above and test by running:

```
<joe@localhost>: fetchmail
```

Ex 10.84: Create a cron utility that tunnels fetchmail every 12 minutes.

SAMPLE
October 6, 2004 -- 23:

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.6 Working in Unpredictable Environments

In an ideal world, all your clients would run the latest version of SSH and use ssh protocol 2. That would make life much safer and easier. Unfortunately, we don't live in an ideal world (yet), so we need to consider working with systems that are inhomogenious, crippled, or outdated.

10.6.1 Allowing Both Protocols

Many Windows and Linux systems run older SSH clients that are incompatible with SSH protocol 2. These clients often can not be configured in any way since SSH is hardcoded into the application. The fact that they are not using a modern version of SSH means that they should be considered untrustworthy clients in every sense of the word.

It is possible to accept both protocol 1 and 2 by changing the Protocol line on the server's `/etc/ssh/sshd_config` to this:

```
Protocol 2,1
```

10.6.2 Don't Trust Users' Known Hosts File

Furthermore, clients that don't use ssh protocol 2 should not be considered particularly trustworthy, since there are many know exploits in Protocol 1. The addition of IgnoreUserKnownHosts to `/etc/ssh/sshd_config` will force the client to acknowledge the connection and effectively means the server doesn't trust the client user's `~/.ssh/known_hosts` for RhostsRSAAuthentication. The two modified lines look like this now:

```
Protocol 2,1
IgnoreUserKnownHosts yes
```

SAMPLE

October 6, 2004-23:49

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM

10.6.3 Adding Iron

If you must allow protocol 1, here are some suggested configuration parameters for the server that will beef up security:

```
# Allow both proto 1 and 2
Protocol 2,1
# Never allow root to ssh directly
PermitRootLogin no
# Don't read ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# Don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
IgnoreUserKnownHosts yes
# Disable tunneled clear text passwords, make em use passphrases!
PasswordAuthentication no
# Be a StrictMode disciplinarian
PermitEmptyPasswords no
StrictModes yes
X11Forwarding no
# Only allow certain users and groups
AllowUsers joe bob elaine
AllowGroups admin
```

Figure 10.14: A /etc/ssh/sshd_config that accepts both protocol 1 and 2

Of course these settings don't guarantee your ssh server will be uncompromisable, but they should help.

SAMPLE

October 6, 2004-

Copyright © 2004 Fortuitous Technologies, Inc.

SAMPLE -- SAMPLE -- SAMPLE -- SAMPLE -- SAM