SSH: The Secure Shell

One of the great advantages of having a machine connected to the network is that you don't have to be sitting in front of it in order to work on it. Before the days of computer networking, a "remote" login meant having a dumb terminal (a display terminal that provided keyboard input and screen output, but no data processing capability) connected to a computer sitting far away, using a long serial line. Due to the physical restrictions on serial line transmissions, this meant you could be, at most, several hundred feet away. When computers started to be networked, one of the first tasks early computer scientists worked out was creating a way for users to log in over the network. Over the years, many different protocols and methods have been used. Our current best option is ssh, the **s**ecure **sh**ell.

The Secure Shell

The ssh protocol was invented to correct many of the problems associated with earlier protocols, such as *telnet*. The major advantage that ssh has over telnet is that ssh encrypts your data, which makes it much more difficult for people to eavesdrop on your transmissions. While we'll be concentrating on the usage of *OpenSSH*, there are many different programs that implement the ssh protocol on the client and server sides. Earlier, we installed the OpenSSH client suite which allows you to log into other networked machines and even transfer files to them. Later in this course, we'll install the OpenSSH server which will allow you to log into your Linux Learning Environment machine from elsewhere.

The main component of the OpenSSH client software suite is the **ssh** command. **ssh** allows you to log into other servers remotely. As a system administrator, you will probably spend most of your time logging into remote machines from your desk, rather than working directly on the console. Managing an ssh client is now required of every sysadmin. Since you've already seen the primary function of the ssh client, we will spend some time here focusing on one of the other features of the ssh client, as well as a couple of other tools that ship with the OpenSSH client suite.

If you've never tried to ssh *from* cold, you'll need to create the **.ssh/** directory. Open a new Terminal session and log in to cold, and then create the .ssh/ directory.

INTERACTIVE SESSION:

```
cold1:~$ mkdir .ssh
cold1:~$ ls -al .ssh
total 48
drwx------  2 username webusers  4096 Nov  5 13:29 .
drwx------ 33 username webusers 16384 Nov  3 17:14 ..
```

SSH Keys

The SSH protocol allows you to use a *public/private key pair* instead of a password in order to authenticate on remote machines. A public/private key pair is a set of cryptographic keys that can be used to encode and decode information. As the names of the keys imply, one is intended to be shared with the public and can, in theory, be read by anyone without compromising security. ALternatively, the private key must be kept secret and should only be readable by you. The public key cannot be used to derive the private key. Any data that is encrypted with the public key can then be decrypted using the private key. OpenSSH uses this property to authenticate you based on a public/private key pair. When you try to initiate a connection, the OpenSSH server on the remote end uses your public key to encrypt a message. It sends that message back to you on your local machine where your ssh client decrypts the message using your private key. Next, your client sends the decrypted message back to the server. Then the server compares the message it received, to the unencrypted version of the message it sent you. If they match, it confirms that the public/private key pair match, and you are authorized. If they do not match, then you do not have the private key that matches the public key on the other end and your connection request is denied.

In practice, you will use a tool that ships with the OpenSSH client suite to generate this key pair. Then you'll distribute your public key to machines that you plan on logging into in the future. Specifically, the contents of the public key are added to the file **~/.ssh/authorized_keys** on the remote host. You can add a passphrase to your private key in order to keep anyone else from taking it and pretending to be you. If you store your private key on a machine that others may log into, using the passphrase is the better option. If you plan to store your private key on a machine that only you can log into, security is less of a concern and you can omit a passphrase. If you supply a passphrase, you will be prompted to enter it to unlock your private key for use by the ssh client. One possible application of an ssh private key without a password would be to allow automated scripts to log into remote machines and perform actions. We'll cover this kind of functionality more in the scripting course, but for now we'll focus on using ssh keys for authentication. Let's generate an ssh public/private key pair using the **ssh-keygen** program. For now, the defaults are sufficient, so just press **Enter** when prompted for a filename to save the key in—but in general, you'll want supply a passphrase:

INTERACTIVE SESSION:

```
[username@username-m0 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_rsa):
Created directory '/home/username/.ssh'.
Enter passphrase (empty for no passphrase): (doesn't appear as you type)
Enter same passphrase again: (doesn't appear as you type)
Your identification has been saved in /home/username/.ssh/id_rsa.
Your public key has been saved in /home/username/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
5d:e3:03:74:0b:18:4c:f2:c0:10:90:5c:37:69:09:b3
username@username-m0.unix.useractive.com
The key's randomart image is:
+--[ RSA 2048]----+
| ..+*===ooo .    |
|  o  +==o. o .   |
|    E.  . . +    |
|         . + .   |
|        S . o    |
|               . |
|                 |
|                 |
|                 |
+-----------------+
```

You now have a public/private key pair in the directory ~/.ssh. Take a look and see what's there:

INTERACTIVE SESSION:

```
[username@username-m0 ~]$ ls -l .ssh
total 8
-rw------- 1 username  username  1743 Mar 13 14:44 id_rsa
-rw-r--r-- 1 username  username   424 Mar 13 14:44 id_rsa.pub
```

Notice the permissions. The file "id_rsa.pub," your public key, is readable by everyone. The file "id_rsa," your private key, is readable only by you. This is exactly what we want. Now we just need to transfer your public key to a remote host, cold.useractive.com, to test out the key pair you just generated. We will use a neat trick to send the file from one host to another using ssh:

INTERACTIVE SESSION:

```
[username@username-m0 ~]$ cat .ssh/id_rsa.pub | ssh
cold.useractive.com cat - ">>" .ssh/authorized_keys
username@cold.useractive.com's password:
```

Let's take a closer look:.

OBSERVE:

`cat .ssh/id_rsa.pub | ssh cold.useractive.com cat - ">>" .ssh/authorized_keys`

We used **cat** to read the file **.ssh/id_rsa.pub**, and we **redirected cat's standard output stream to ssh**. ssh can execute an arbitrary command on the remote host, rather than executing a shell, so we let it. In this case, we **request that cat be executed on the remote host**. Giving cat the argument **-** on the remote host, instead of

giving cat a filename, instructs it to read from its standard input stream, which will come from the standard output stream of the cat on the localhost via ssh. Does that make sense. Pause and let that sink in for a second. Finally, we **tell cat, on the remote host, to append its standard output stream to the file .ssh/authorized_keys**. The double quotation marks are necessary here in order to keep the local shell from parsing the >> operator and redirecting the output to a local file. Fortunately for us, there are less complicated ways to transfer files to and from remote hosts.

Other OpenSSH Tools

The OpenSSH suite includes a couple of other tools that are useful for using and administering Linux systems: **sftp** and **scp**. both of these tools can transfer files securely between hosts, but their interfaces operate in vastly different ways. If you've used ftp before, then you're also familiar with the workings of sftp. It's similar to the way scp operates like regular cp.

SFTP

The **S**ecure **F**ile **T**ransfer **P**rotocol (sftp) tool that comes with OpenSSH works the same way as regular FTP, except that all of your traffic is encrypted using the SSH protocol. This is important because the FTP protocol sends all data in plain-text, meaning that anybody who can see your packets on the network can see exactly what you're doing. As with telnet, FTP may allow an intruder to steal your password without much difficulty. That's why you should*always* use SFTP rather than FTP if you have the option.

To connect to another host with sftp, you run **sftp *remote_host***. Go ahead and connect to cold.useractive.com using sftp now:

INTERACTIVE SESSION:

```
[username@username-m0 ~]$ sftp cold.useractive.com
Connecting to cold.useractive.com...
Enter passphrase for key '/home/username/.ssh/id_rsa':
sftp>
```

When you see the "sftp>" prompt, you're connected. Several of the commands available within the sftp shell are similar to commands available in bash. For example, **ls** lists files on the remote machine, **cd** changes directories on the remote machine, and so on. Type **help** for a full list of available commands. Commands that act on the remote host such as **ls** and **cd** have locally acting counterparts like **lls** and **lcd**. The **lls** command, for example, will show you files in the present working directory on your local machine. This allows you to review a list of files that you may want to upload using sftp. The two most important commands though, are **get** and **put**. These commands allow you to download and upload files, respectively. Let's test the **put** command after we use another feature of sftp to create a file on the local machine:

INTERACTIVE SESSION:

```
sftp> !touch sftp_test
sftp> put sftp_test
Uploading sftp_test to /users/username/sftp_test
sftp_test                                                   100%     0
0.0KB/s   00:00
```

The **!** operator allows you to execute an arbitrary command on your local machine from the sftp shell. In this case, we used it to touch the file sftp_test. The **get** command operates in a similar way. If you have any files in your home directory on cold, feel free to use them to try out the get command.

SCP

Secure **cp** (**scp**), like sftp, provides a more secure version of a standard command for use where security can be an issue. The general format of this command is **scp *source_file destination_file***. It's possible for the source file, destination file, or both to reside on remote machines. When the file is located on your local machine, you specify the filename as you would normally specify it to cp. The remote file format is similar, but you must also include the host name of the remote machine and optionally, a username that will be used to log into the machine. The remote file specification will look like this: *username@hostname:/path/to/file*. Similar to ssh, if you don't specify a username, the username that you are currently logged in with on your local machine will be used. Let's use scp to copy a file from your Linux Learning Environment machine to the OST login host, cold.useractive.com.

INTERACTIVE SESSION:

```
[username@username-m0 ~]$ touch scp_test
[username@username-m0 ~]$ scp scp_test
username@cold.useractive.com:~/
Enter passphrase for key '/home/username/.ssh/id_rsa':
scp_test                                                    100%     0
0.0KB/s   00:00
```

Note As with cp, you can specify a directory as the destination for the file copy. In this case, our directory is "~/", which is shorthand for your home directory.

Feel free to ssh to cold and verify that the file has copied over. You can also copy entire directories with scp by specifying the "-r" flag, "r" standing for "recursive." This behavior is identical to that of regular cp, which can use the "-r" flag to copy directories as well.

Source: http://courses.oreillyschool.com/sysadmin2/ssh.html