Command Line: ssh & scp

Secure Transmissions

The Unix heritage programs Telnet and FTP provide facilities for connecting and transferring data to other machines on the network. But they are insecure, as they transfer data in the clear. The Secure Shell tools provide a secure alternative.

BY HEIKE JURZIK

SSH (Secure *SH*ell) is both the protocol name and the implementation, that is the program itself. It is a secure method of logging on to other computers using an encrypted connection, of running commands (including X applications) on the remote machine, and of copying data between systems. Similar non-encrypting protocols such as *telnet* or *ftp* send both data and passwords across the wire in the clear.

Currently two incompatible versions of the SSH protocol exist – SSH 1.X and SSH 2.X. Version 1.0 was released in 1995 by Tatu Ylönen, a Finnish developer, and became the accepted standard. Use of the software was free up to version 1.2.12, but licensing restrictions were applied later [2]. OpenSSH [3] is based on the last free version, and is the subject of ongoing development and enhancement. Originally an OpenBSD only project, OpenSSH is now available for other Unix platforms and has spoken both SSH 1.X and 2.X since version 2.1.0. OpenSSH uses the OpenSSL [5] crypto-routines and is part of most Linux distributions.

The Right Connection

On the client side SSH comprises *ssh/slogin* as a functional replacement for *telnet*, *rlogin*, or *rsh* and *scp* for file copying operations (replacing *ftp* and *rcp*). *sshd* (the SSH daemon) provides the server side. In addition to this there are some management tools, such as *ssh-keygen* (a key creation utility), *ssh-agent* (to provide key management and login automation), *ssh-add* (for registering new keys with the SSH agent), and *make-ssh-known-hosts* (allows you to create a list of known public host keys for a domain). *ssh* must be running both on your own machine and the other end of the connection; to be more precise, to open a connection from A to B, machine A must have the client software and B must be running the daemon. To open a secure connection to another machine, you can enter either of the following:

```
ssh UserID@RemoteHost
ssh -l UserID RemoteHost
```

Userid@ is only required if you have a different username on the remote system. When a connection is opened (via TCP port 22) computers A and B will exchange protocol versions. If the protocol versions are incompatible, the connection cannot be completed:

Listing 1: Generating a DSA Key

```
huhn@asteroid:~$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/huhn/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/huhn/.ssh/id_dsa.
Your public key has been saved in /home/huhn/.ssh/id_dsa.pub.
The key fingerprint is:
72:d5:ad:76:cf:e4:89:a0:8e:f9:57:66:87:55:cd:cc
```

```
Protocol major versions differ:
2 vs. 1
```

The *-1* or *-2* options let you switch versions quite painlessly. You can add a command to run on establishing the *ssh* connection. It should be placed in quotes to prevent the shell from parsing it:

```
huhn$ ssh plutarch "ls lang"
huhn@plutarch's password:
English/   German/
```

It is also possible to remotely launch applications that take command of the screen, although doing so does require an additional parameter to specify a "pseudo-terminal". The *-t* flag tells SSH to encrypt output from the terminal and display it in decrypted form on your screen: *ssh -t plutarch "mutt"*

If you want to encrypt an X application and launch it on a remote machine, both the client and the server side will need to implement X forwarding. The server side configuration file *sshd_config* is located in the */etc/ssh* directory. As is the case for all system global settings, only the administrator *root* is permitted to modify it. The appropriate entry is called *X11Forwarding yes*. The client configuration is located in the same directory and called *ssh_config* (without a "d"). Most distributions will have an entry like:

```
#   ForwardX11 no
```

To enable forwarding *root* needs to change the line to *ForwardX11 yes*

One alternative is to enable this feature temporarily using the *-X* option. To launch an X application like Mozilla remotely, either use *ssh [UserID@]RemoteHost* or *ssh -X [UserID@]RemoteHost* to log on and enter the appropriate command: *mozilla &*. The ampersand leaves the process running in the background, so the terminal stays free to accept other commands. One practical feature of X forwarding is the fact that the **DISPLAY** variable is automatically configured cor-

rectly – so the connection is not only secure, but also simple to establish.

Secure Transport

The second major field of application for SSH, besides secure login, is file transfer. Using *scp* to copy a file is almost as simple as using *cp* for local copying. A call to *scp* typically looks like :

```
scp file UserID@RemoteHost:
```

Assuming that the usernames match, you can leave out the *UserID*. If you do not want to simply copy the file to your home directory, you can specify the path after the colon (*scp file UserID@RemoteHost:/tmp/archive/*).

The *-r* parameter allows you to copy complete directory trees recursively, and the *cp* option *-p* ("preserve" user and group privileges) is also available. Of course, this only makes sense if user and group IDs are identical on both systems.

Your Own Key

When transferring data between computers, or logging on to remote systems on a regular basis, continual password prompts can be a nuisance; and the interactive password prompt rules out scripting standard tasks. SSH provides a facility for depositing a key on a target system allowing it to check your logon credentials without requiring a password. The *ssh-keygen* (key generation) program allows you to create the required key (Listing 1). Protocol version 1.X uses **RSA** keys, whereas version 2.X uses **DSA** encryption. The *-t* option specifies the type of key.

The "passphrase" prompt in this listing has nothing to do with your password on the target or local system – instead the passphrase is used to protect your keypair (which comprises a public and private key). This kind of protection is useful if you want to deploy SSH on a laptop, for example. If your laptop is stolen, and the thief discovers your SSH key on the hard disk, it could be used to compromise another system.

The keypair is stored in the *.ssh* subdirectory (note the dot) below your home directory, comprising *id_dsa* and *id_dsa.pub*. The first file is your secret, private key (a binary file that cannot be manipulated using an editor). The second file with the *.pub* suffix contains your public key. You can copy it to a target system to provide automatic logon facilities on that machine:

- Copy *~/.ssh/id_dsa.pub* to your home on the target system: *scp ~/.ssh/id_dsa.pub [UserID@]RemoteHost:*
- Log on to the target system and check whether a *.ssh* directory already exists; if not, use *mkdir ~/.ssh* to create it
- Change to the directory (on the target machine, *cd ~/.ssh*) and add the file you copied from your machine to the *authorized_keys* file (this file may already exist): *cat ~/id_dsa.pub >> authorized_keys* (if the file does not exist, this command will create it).
- Delete the *id_dsa.pub* from the main directory (and only there!), as you no longer need it: *rm ~/id_dsa.pub*

Ensure the *.ssh* directory has the correct privileges assigned, that is, *700* for the directory itself (readable, writable, accessible to the user, and no-one else). Additionally, the home directory should not provide write privileges to the group (*g*) or others (*o*); just to make sure, you might like to run *chmod go-w ~* and *chmod 700 ~/.ssh*.

The *ssh-agent* and *ssh-add* programs allow you to automate querying the passphrase. The SSH agent uses *ssh-add* to store any passphrases you supply, removing the need for password prompts when you launch *ssh*. This is particularly useful, if the *ssh* connection is frequently disestablished and re-established.

First launch the SSH agent: *ssh-agent*. It returns a number of command lines that you may need to copy and run to set several environment variables:

```
SSH_AUTH_SOCK=/tmp/ssh-➥
XXOOM4II/agent.32162; export ➥
SSH_AUTH_SOCK;
SSH_AGENT_PID=32163; export ➥
SSH_AGENT_PID;
echo Agent pid 32163;
```

Of course, we only included this procedure, where *ssh-agent* is used to output the command lines, to illustrate the required step. In a real-life situation you could use eval to evaluate the output from ssh-agent instead. To do so, enter *eval `ssh-agent`*. The agent needs to be running before you can use *ssh-add*:

```
huhn$ ssh-add -l
The agent has no identities.
```

Calling *ssh-add* without any additional parameters tells the tool to search the *~/.ssh* directory for private keys, and prompts you to enter the passphrase for each key it finds.

```
huhn$ ssh-add
Enter passphrase for /home/➥
huhn/.ssh/id_dsa: *****
Identity added: ➥
/home/huhn/.ssh/id_dsa ➥
(/home/huhn/.ssh/id_dsa)
```

The program learns the passphrase during this step:

```
huhn$ ssh-add -l
1024 f3:c9:b6:5d:23:3a:9d:61:➥
50:19:63:3c:e8:22:7c:86 ➥
/home/huhn/.ssh/id_dsa (DSA)
```

You can now log on to the target machines (from the current shell and any other shells where you have set *SSH_AUTH_SOCK* and *SSH_AGENT_PID* correctly) without entering a password. ■

GLOSSARY

DISPLAY: *This environment variable specifies the X display X Window programs will be displayed on. It typically defaults to ":0", meaning that the local (first) X server is accessed. In the case of an SSH login with X forwarding from computer1 to computer2, the variable is set to something like "computer1:10.0".*

RSA: *An encryption algorithm named after its creators Rivest, Shamir, and Adleman. It was developed in 1977.*

DSA: *OpenSSH uses the "Digital Signature Algorithm" as its encryption algorithm. This algorithm was published in 1994 by the National Institute of Standards and Technology (NIST).*

INFO

[1] *http://www.ssh.com/*

[2] *http://www.openssh.com/history.html*

[3] *http://www.openssh.org/*

[4] *http://www.employees.org/~satch/ssh/faq/ssh-faq.html*

[5] *http://www.openssl.org/*