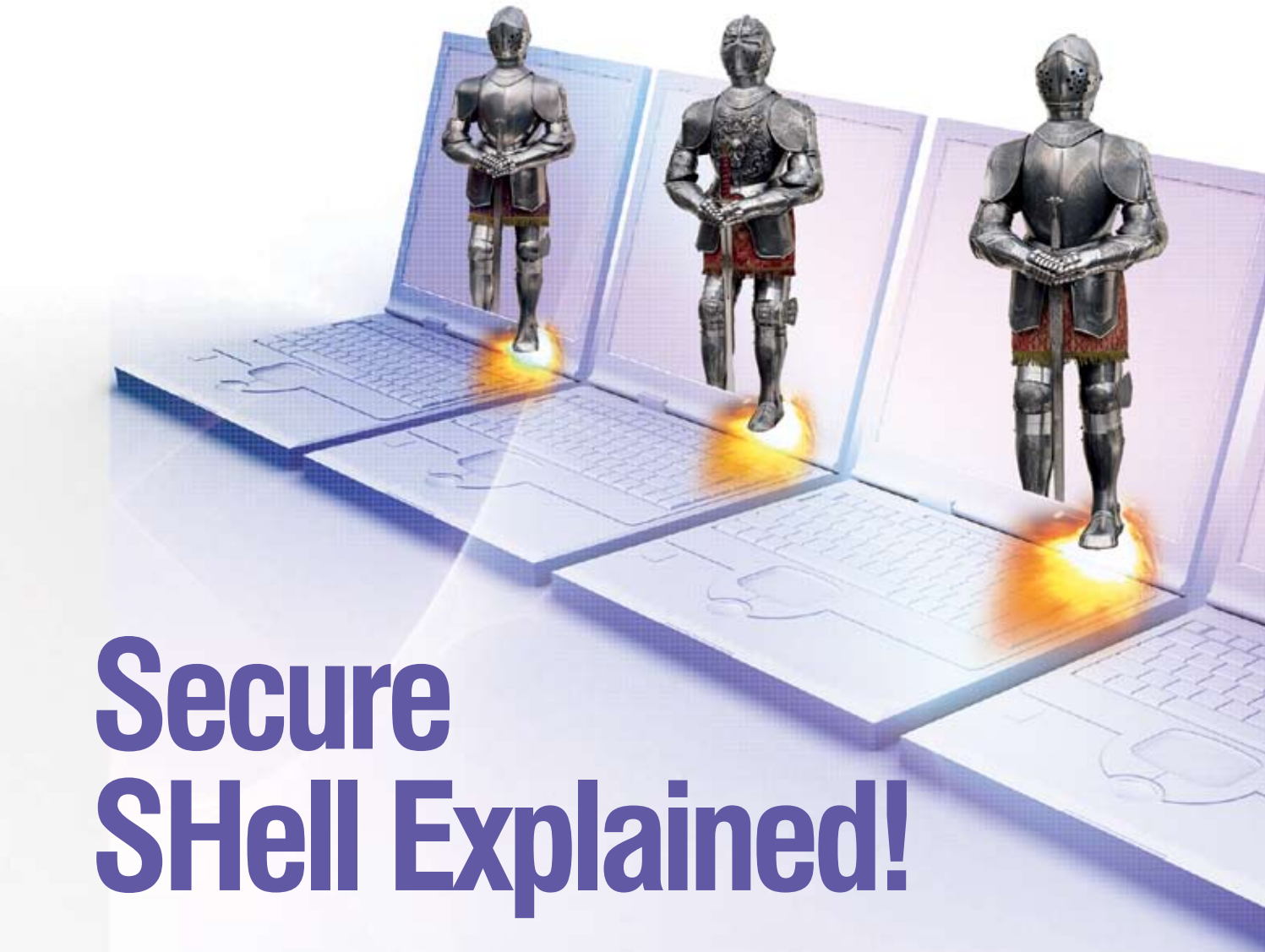

Secure SHell Explained!

Here're some insights into SSH (Secure Shell), an essential tool for accessing remote machines.

SSH is used to access or log in to a remote machine on the network, using its hostname or IP address. It's a secure network data exchange protocol that came up as a replacement for insecure protocols like *rsh*, *telnet*, *ftp*, etc. It encrypts the bi-directional data transfers using cryptographic algorithms, making the data transfers secure. Hence, it is free from password theft or from the sniffing of packets being transferred over a network.

Some of the highlights of the SSH protocol are:

- Compression
- Public key authentication
- Port forwarding
- Tunnelling
- X11 forwarding
- File transfer

SSH runs as a service daemon to facilitate remote log-ins.

To install the SSH server on Debian-based distros, key in the following command:

```
# apt-get install openssh-server
```

Although the default port for SSH is 22, you can also configure it to run with other custom ports.

To perform remote log-ins, we require an SSH client. There are lots of SSH clients available, and they can be installed on Debian-based system as follows:

```
# apt-get install openssh-client.
```

It is possible to access remote UNIX/Linux machines from any other OSs using some SSH clients. For example, it is possible to remotely log in to a UNIX box from Windows using the SSH client called Putty [*www.putty.org*].

Basic operations

We can remotely log in to a machine by issuing the following command:

```
slynux@gnubox:~$ ssh user@hostname
```

Here, 'user' is an existing user on the remote machine 'hostname', so you need to replace the two with relevant information. [You can also use an IP address instead of a hostname to log in.] Hitting the *Enter* key now will result in a prompt for the user's password; and after entering it, you will get the remote user's shell prompt.

Alternately, we can also provide the following command:

```
slynux@gnubox:~$ ssh hostname
```

...which is equivalent to:

```
slynux@gnubox:~$ ssh slynux@hostname
```

...i.e., if the user name of the one trying to remotely log in is the same as the current user, there is no need to specify the user name explicitly.

Sometime systems administrators will configure the SSH daemon to listen to a non-standard port such as 422, instead of 22. This is done for security reasons—to make it difficult for an unauthorised person to easily find which post number the SSH daemon is listing to.

In cases where we need to perform the SSH log-in via a non-standard port, we can specify the port number explicitly using the *-p* option:

```
slynux@gnubox:~$ ssh -p 422 slynux@hostname
```

The initial key discovery

When you connect to an SSH server for the first time, you will be asked to verify the server's key. When the users continue confirming 'yes', it will attach the server key with the hostname and store it in the *~/.ssh/known_hosts* file. After the initial probe for the server verification, it will check this *known_hosts* file to verify the authority of the server to which the SSH client is requesting a connection to.

```
slynux@gnubox:~$ ssh slynux@192.168.1.2
The authenticity of host '192.168.1.2 (192.168.1.2)' can't be established.
RSA key fingerprint is 6d:92:2c:f1:74:e7:a9:21:64:57:90:6f:72:3e:a3:18.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.2' (RSA) to the list of known hosts.
slynux@192.168.1.2's password:
Last login: Sun May 17 21:04:29 2009 from slynux-laptop
slynux@gnubox:~$
```

This initial key discovery process is to ensure security. It is possible for an attacker to steal information from the remote user log-in by impersonating the server, i.e., if the attacker can provide a server with the same host name and user authentication, the user connecting from the remote machine will be logged into a fraud machine and data may be stolen.

Each server will have a randomly generated RSA server key.

To ensure security, in cases where the server key changes, the SSH client will issue a serious warning reporting that the host identification has failed and that it will stop the log-in process.

```
slynux@gnubox:~$ ssh slynux@192.168.1.2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@
@    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)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
```

```
cd:41:70:30:48:07:16:81:e5:30:34:66:f1:56:ef:db.
Please contact your system administrator.
Add correct host key in /home/slynux/.ssh/known_hosts to get rid of this
message.
Offending key in /home/slynux/.ssh/known_hosts:24
RSA host key for localhost has changed and you have requested strict checking.
Host key verification failed.
```

If we're certain about the key identification chance of the remote machine, we can remove the corresponding server key entry from our *~/.ssh/known_hosts* file. Following which, the next time you try to log in, you will be asked for a key verification again and the server key will be again registered in the *known_hosts* file.

Executing remote commands

The main purpose of SSH is to execute commands remotely. As we have already seen, immediately after a successful SSH log-in, we're provided with the remote user's shell prompt from where we can execute all sorts of commands that the remote user is allowed to use. This 'pseudo' terminal session will exist as long as you're logged in.

It is also possible to execute commands on a one-at-a-time basis without assigning a pseudo-terminal, as follows:

```
slynux@gnubox:~$ ssh slynux-laptop 'uname -a'
slynux@slynux-laptop's password:
Linux slynux-laptop 2.6.28-9-generic #31-Ubuntu SMP Wed Mar 11 15:43:58 UTC
2009 i686 GNU/Linux
slynux@gnubox:~$
```

Note that we're back at our local shell prompt. The syntax is: *ssh user@hostname 'commands in quote'*.

Input/output redirection

Piping is a nifty feature provided by the shell. If you aren't already familiar with it, have a look at the basics of piping in the following section.

Piping is used for input and output redirection. In *nix shells, we can redirect input/output in different ways, as follows:

```
echo "Test" > file
```

Here the output text stream ("Test") is directed to a file. Thus the stream is stored to a file named file. '>' is the output redirection operator.

Now, take a look at the following command:

```
cat < file
```

Here, input is directed to the *cat* command. *cat* performs the concatenation of the input stream. Here the input is a file named *file*. '<' is an input redirection operator that directs the input stream to the specified command. Here it directs the input text stream from the file to the cat.

Finally, take a look at the following command:

```
echo hello | command1 | command2
```

Here, '|' is the piping operator. It uses the output of one command as the input of another. We can use any number of pipes serially, i.e., the output of one command appears as the input of another, and the output of *this second* command appears as the input of the third command and so on. Thus, the net result will be a serial application of these commands on data, one after the other.

For example:

```
slynux@slynux-laptop:~$ echo "hello" | tr -d 'l'
"heo"
```

All of the above input/output redirection operations can also be performed using SSH commands. Let us look at the possibilities:

```
slynux@gnubox:~$ ssh slynux-laptop 'cat /etc/passwd | grep root'
slynux@gnubox:~$ ssh slynux-laptop 'cat /etc/passwd' > file.txt
slynux@gnubox:~$ ssh slynux-laptop 'cat > directed.txt' < file.txt
```

You can also club compression utilities along with SSH:

```
slynux@gnubox:~$ ssh slynux-laptop 'tar -czf - file.txt' > file.tar.gz
```

In the above command, we have used *tar -czf* to create a tarball file. *'tar -czf - file.txt'* has - [hyphen] as the file name. When a hyphen is provided as a filename, it implies that the output is not written to a file; instead, it is redirected to standard output.

Now, to list the files in the compressed archive, run the following command:

```
slynux@gnubox:~$ tar -ztf file.tar.gz
file.txt
```

The SSH protocol also supports data transfer with compression—which comes in handy when bandwidth is an issue. Use the *-C* option with the *ssh* command to enable compression:

```
slynux@gnubox:~$ ssh -C user@hostname
```

File transfer

SSH also offers the file transfer facility between machines on the network and is highly secure, with SSH being an encrypted protocol. Also, the transfer speed can be improved by enabling compression. Two significant data transfer utilities that use the SSH protocol are SCP and SFTP.

SCP stands for Secure Copy. We can use it to copy files from a local machine to a remote machine, a remote machine to a local machine, and a remote machine to another remote machine.

For the local machine to remote machine file transfer, we use the following:

```
scp local_file_path user@remote_host:destination_file_path
```

For a remote machine to local machine transfer:

```
scp user@remote_host:remote_file_path local_file_path
```

For a remote machine to remote machine transfer:

```
scp user1@remote_host1 user2@remote_host2
```

You can even use wildcards to select files:

```
scp :/home/slynux/*.txt /home/gnubox/scp_example/
```

SFTP stands for Secure File Transfer Protocol. It is a secure implementation of the traditional FTP protocol with SSH as the backend. Let us take a look at how to use the *sftp* command:

```
sftp user@hostname
```

For example:

```
slynux@slynux-laptop:~$ sftp slynux-laptop
Connecting to slynux-laptop...
slynux@slynux-laptop's password:
sftp> cd django
sftp> ls -l
drwxr-xr-x    2 slynux   slynux      4096 Apr 30 17:33 website
sftp> cd website
sftp> ls
__init__.py    __init__.pyc    manage.py      settings.py    settings.pyc
urls.py        urls.pyc        view.py        view.pyc
sftp> get manage.py
Fetching /home/slynux/django/website/manage.py to manage.py
/home/slynux/django/website/manage.py        100%  542     0.5KB/s   00:01
sftp>
```

If the port for the target SSH daemon is different from the default port, we can provide the port number explicitly as an option, i.e., *-oPort=port_number*.

- Some of the commands available under *sftp* are:
- *cd*—to change the current directory on the remote machine
- *lcd* —to change the current directory on localhost
- *ls*—to list the remote directory contents
- *lls*—to list the local directory contents
- *put*—to send/upload files to the remote machine from the current working directory of the localhost
- *get*—to receive/download files from the remote machine to the current working directory of the localhost

sftp also supports wildcards for choosing files based on patterns.

SSH over GUI file managers

In GNOME, we can use the SSH protocol to navigate remote filesystems in the Nautilus file manager. It works as a GUI implementation of *sftp*. Type *ssh://user@hostname[:port]* at the address bar. It will prompt you for the password of the 'user' and then mount the remote filesystem. After that, we can navigate the filesystem just as with locally mounted disk data.

As for KDE users, you can use the *fish* protocol in Dolphin or Konqueror to browse remote filesystems. Type *fish://user@hostname[:port]* in the location bar and press *Enter*. It will again prompt for the remote user's password.

Running XWindow applications remotely

Well, the good news is that SSH is also a good enough protocol that can aid you to run applications other than terminal utilities remotely, with the help of X11 forwarding. To enable X11 forwarding, add the following line in */etc/ssh/ssh_config*, the configuration file.

```
ForwardX11 yes
```

Now to launch the GUI apps remotely, execute *ssh* commands with the *-X* option. For example:

```
ssh -X slynux-laptop 'vlc'
```

Port forwarding

One of the significant uses of SSH is port forwarding. SSH allows you to forward ports from client to server and server to client. There are two types of port forwarding: local and remote. In local port forwarding, ports from the client are forwarded to server ports. Thus the locally forwarded port will act as the proxy port for the port on the remote machine.

To establish local port forwarding, use the following code:

```
ssh -L local_port:remote_host:remote_port
```

For example:

```
ssh -L 2020:slynux.org:22
```

Here, it forwards local port 2020 to slynux.org's ssh port 22. Thus, we can use:

```
ssh localhost -p 2020
```

...instead of:

```
ssh slynux.org
```

In remote port forwarding, ports from the server are forwarded to a client port. Thus ports on the remote host will act as the proxy for ports on the local machine.

The significant application of remote forwarding is that, suppose you have a local machine that lies inside an internal network connected to the Internet through a router

or gateway—if we want to access the local machine from outside the network, it is impossible to access it directly. But by forwarding the local ports to a remote host, we can access the local machine through ports of the remote host.

Let's see how remote port forwarding is executed:

```
ssh  -R remoteport:remotehost:localport
```

For example:

```
ssh -R 2020:slynux.org:22
```

To SSH to the local machine from outside the internal network, we can make use of *slynux.org* as *ssh slynux.org:2020*.

SOCKS4 proxy

SSH has an interesting feature called Dynamic Port forwarding with which the SSH TCP connection will work as a SOCKS4 proxy. By connecting to the given port, it handles SOCKS data transfer requests.

An important application of Dynamic Port forwarding is the following case.

Let's suppose you have a machine on a network that is connected to the Internet and you have another machine on the same network that does not have any Internet connection. By using SSH Dynamic port forwarding, you can easily access the Internet by setting up the machine with an Internet connection to act as the SOCKS4 proxy using an SSH tunnel.

For dynamic port forwarding, use the following command:

```
ssh -D 3000 remotehost
```

Now, in your browser, specify proxy settings as:

- SOCKS4
- host: localhost
- port: 3000

To enable the DNS service in Firefox, navigate the *about:config* page and set...

```
network.proxy.socks_remote_dns = true
```

Automatic key authentication

Each time you access the other machine for the remote execution of some command, it probes for the password. This is not desirable when we need to automate tasks. If we need to shut down or reboot all the machines on the LAN, it is impractical to type the user password for command execution on each of the machines. There should be some mechanism that handles automatic authentication without probing for a password.

The solution for this hurdle is public key authentication, for which we will generate a public key from the machine we need to execute remote commands. That public key will be copied to each of the remote machines. Thus each time when we execute remote commands, it will perform a user authentication by verifying the public key and it no more probes for the passwords.

Generate the public key as follows:

```
slynux@slynux-laptop:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/slynux/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/slynux/.ssh/id_rsa.
Your public key has been saved in /home/slynux/.ssh/id_rsa.pub.
The key fingerprint is:
0e:04:3d:e3:2a:54:8c:47:ae:10:9a:96:41:be:c1:8f slynux@slynux-laptop
```

Now we have the public key in the file *~/.ssh/id_rsa.pub*.

```
slynux@slynux-laptop:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAuj6N7/juQ8CUtdmFP816Xn4iEI
j73pO7+xHPgIBFZGgxg8yeYZmU7zBjCUAcSXx/NhRiF7YytozhvWk+n92DBFL6
U62lrukqtB/WdZRHh2w1JH4adC3hCDSlJglaxt5WoZK4aFzjGRCbdTBxC2rELQu
u9z9qowzQ8bU3WdO8UK0+U0/u8XSWXvWE4W2THAlWFTRjp+KDX33Ms9u
IYyx/h3Tx5voPSxV6cYBZfh5kJMzEoYDBCUpua6uHV4zDfJFNnN6Sdpt3213FY/
cGRvT1vBCRDSmQd0Xkq2hU8npCfz0rQjXqGPuuzfVW8Ie6yRQQPtqXc3/
J5UMglgumgDgw== slynux@slynux-laptop
```

To implement auto authentication, append the public key in the *~/.ssh/authorized_keys* file in each of the remote machines where we need to perform auto authentication.

Appending the key can be performed manually or it can be automated using an *ssh* command as follows:

```
ssh remote_host "cat >> .ssh/authorized_keys" < ~/.ssh/id_rsa.pub
```

Finally, let us write a single loop shell script to reboot all the switched-on machines in the network.

```
#!/bin/bash
base_ip="192.168.0." ;

for machine in $base_ip{1..255};
do
        ping -c2 $machine &> /dev/null ;
        if [ $? -eq 0 ];
        then
            ssh $machine reboot ;
        fi
done
```

That's it about the secure shell. Hope you enjoyed this tutorial. Till we meet again, happy hacking! END

By: Sarath Lakshman

The author is a FOSS enthusiast interested in QT programming and technology. He is fond of reviewing the latest OSS tools and distros.