Eleven SSH Tricks



Run remote GUI applications and tunnel any Net connection securely using a free utility that's probably already installed on your system.

by Daniel R. Allen

SSH is the descendant of rsh and rlogin, which are non-encrypted programs for remote shell logins. Rsh and rlogin, like telnet, have a long lineage but now are outdated and insecure. However, these programs evolved a surprising number of nifty features over two decades of UNIX development, and the best of them made their way into SSH. Following are the 11 tricks I have found useful for squeezing the most power out of SSH.

Installation and Versions

OpenSSH is the most common free version of SSH and is available for virtually all UNIX-like operating systems. It is included by default with Debian, SuSE, Red Hat, Mandrake, Slackware, Caldera and Gentoo Linux, as well as OpenBSD, Cygwin for Windows and Mac OS X. This article is based on OpenSSH, so if you are using some other version, check your documentation before trying these tricks.

X11 Forwarding

You can encrypt X sessions over SSH. Not only is the traffic encrypted, but the DISPLAY environment variable on the remote system is set properly. So, if you are running X on your local computer, your remote X applications magically appear on your local screen.

Turn on X11 forwarding with `ssh -X host`. You should use X11 forwarding only for remote computers where you trust the administrators. Otherwise, you open yourself up to X11-based attacks.

A nifty trick using X11 forwarding displays images within an xterm window. Run the web browser w3m with the in-line image extension on the remote machine; see the Debian package w3m-img or the RPM w3m-imgdisplay. It uses X11 forwarding to open a borderless window on top of your xterm. If you read your e-mail remotely using SSH and a text-based client, it then is possible to bring up in-line images over the same xterm window.

Config File

SSH looks for the user config file in ~/.ssh/config. A sample might look like:

```
ForwardX11 yes
Protocol 2,1
```

Using `ForwardX11 yes` is the same as specifying `-X` on the command line. The Protocol line tells SSH to try SSH2 first and then fall back to SSH1. If you want to use only SSH2, delete the `,1`.

The config file can include sections that take effect only for certain remote hosts by using the Host option. Another useful config file option is User, which specifies the remote user name. If you often log in to a machine with `ssh -l remoteuser remotehost` or `ssh remoteuser@remotehost`, you can shorten this by placing the following lines in your config file:

```
Host remotehost
ForwardX11 yes
User remoteuser

Host *
ForwardX11 no
```

Now, you can type `ssh remotehost` to log on as user remoteuser with the ForwardX11 option turned on. Otherwise, ForwardX11 is turned off, as recommended above. The asterisk matches all hosts, including hosts already matched in a Host section, but only the first matching option is used. Put specific Host sections before generic sections in your config file.

A system-wide SSH config file, /etc/ssh/ssh_config, also is available. SSH obtains configuration data in the following order: command-line options, user's configuration file and system-wide configuration file. All of the options can be explored by browsing man ssh_config.

Speeding Things Up: Compression and Ciphers

SSH can use gzip compression on any connection. The default compression level is equivalent to approximately 4× compression for text. Compression is a great idea if you are forwarding X sessions on a dial-up or slow network. Turn on compression with `ssh -C` or put `Compression yes` in your config file.

Another speed tweak involves changing your encryption cipher. The default cipher on many older systems is triple DES (3DES), which is slower than Blowfish and AES. New versions of OpenSSH default to Blowfish. You can change the cipher to blowfish with `ssh -c blowfish`.

Cipher changes to your config file depend on whether you are connecting with SSH1 or SSH2. For SSH1, use `Cipher blowfish`; for SSH2, use:

```
Ciphers blowfish-cbc,aes128-cbc,3des-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
```

Port Forwarding

Ports are the numbers representing different services on a server; such as port 80 for HTTP and port 110 for POP3. You can find the list of standard port numbers and their services in /etc/services. SSH can translate transparently all traffic from an arbitrary port on your computer to a remote server running SSH. The traffic then can be forwarded by SSH to an arbitrary port on another server. Why would you want to do this? Two reasons: encryption and tunneled connections.

Encryption

Many applications use protocols where passwords and data are sent as clear text. These protocols include POP3, IMAP, SMTP and NNTP. SSH can encrypt these connections transparently. Say your e-mail program normally connects to the POP3 port (110) on mail.example.net. Also, say you can't SSH directly to mail.example.net, but you have a shell login at shell.example.net. You can instruct SSH to encrypt traffic from port 9110 (chosen arbitrarily) on your local computer and send it to port 110 on mail.example.net, using the SSH server at shell.example.net:

```
ssh -L 9110:mail.example.net:110 shell.example.net
```

That is, send local port 9110 to mail.example.net port 110, over an SSH connection to shell.example.net.

Then, simply tell your e-mail program to connect to port 9110 on localhost. From there, data is encrypted, transmitted to shell.example.net over the SSH port, then decrypted and passed to mail.example.net over port 110. As a neat side effect, as far as the POP3 dæmon on mail.example.net knows, it is accepting traffic from shell.example.net.

Tunneled Connections

SSH can act as a bridge through a firewall whether the firewall is protecting your computer, a remote server or both. All you need is an SSH server exposed to the other side of the firewall. For example, many DSL and cable-modem companies forbid sending e-mail from your own machine over port 25 (SMTP).

Our next example is sending mail to your company's SMTP server through your cable-modem connection. In this example, we use a shell account on the SMTP server, which is named mail.example.net. The SSH command is:

```
ssh -L 9025:mail.example.net:25 mail.example.net
```

Then, tell your mail transport agent to connect to port 9025 on localhost to send mail. This exercise should look quite similar to the last example; we are tunneling from local port 9025 to mail.example.net port 25 over mail.example.net. As far as the firewall sees, it is passing normal SSH data on the normal SSH port, 22, between you and mail.example.net.

A final example is connecting through an ISP firewall to a mail or news server inside a restricted network. What would this look like? In fact, it would be the same as the first example; mail.example.net can be walled away inside the network, inaccessible to the outside world. All you need is an SSH connection to a server that can see it, such as shell.example.net. Is that neat or what?

Limitations/Refinements to Port Forwarding

If a port is reassigned on a computer (the local port in the examples above), every user of that computer sees the reassigned port. If the local system has multiple users, tunnel only from unused, high-numbered ports to avoid confusion. If you want to forward a privileged local port (lower than 1024), you need to do so as root. Forwarding a lower-numbered port might be useful if a program won't let you change its port, such as standard BSD FTP.

By default, a tunneled local port is accessible only to local users and not by remote connection. However, any user can make the tunneled port available remotely by using the -g option. Again, you can do this to privileged ports only if you are root.

Any user who can log in with SSH can expose any port inside a private network to the outside world using port forwarding. As an administrator, if you allow incoming SSH connections, you're really allowing incoming connections of any kind. You can configure the OpenSSH dæmon to refuse port forwarding with `AllowTcpForwarding no`, but a determined user can forward anyway.

A config file option is available to forward ports; it is called LocalForward. The first port-forwarding example given above could be written as:

```
Host forwardpop
Hostname shell.example.com
LocalForward 9110 mail.example.com:110
```

This way, if you type `ssh forwardpop` you receive the same result as in the first example. This example uses

the Host command described above and the HostName command, which specifies a real hostname with which to connect.

Finally, a command similar to LocalForward, called RemoteForward, forwards a port from the computer to which you are connected, to your computer. Please read the ssh_config man pages to find out how.

Piping Binary Data to a Remote Shell

Piping works transparently through SSH to remote shells. Consider:

```
cat myfile | ssh user@desktop lpr

tar -cf - source_dir | \
ssh user@desktop 'cat > dest.tar'
```

The first example pipes myfile to lpr running on the machine named desktop. The second example creates a tar file and writes it to the terminal (because the tar file name is specified as dash), which is then piped to the machine named desktop and redirected to a file.

Running Remote Shell Commands

With SSH, you don't need to open an interactive shell if you simply want some output from a remote command, such as:

```
ssh user@host w
```

This command runs the command **w** on host as user and displays the result. It can be used to automate commands, such as:

```
perl -e 'foreach $i (1 .. 12) \
{print `ssh server$i "w"`}'
```

Notice the back-ticks around the SSH command. This uses Perl to call SSH 12 times, each time running the command **w** on a different remote host, server1 through server12. In addition, you need to enter your password each time SSH makes a connection. However, read on for a way to eliminate the password requirement without sacrificing security.

Authentication

How does SSH authenticate that you should be allowed to connect? Here are some options:

- By hostnames only: uses .rhosts file; insecure; disabled by default.
- By hostnames and host-key checking.
- The S/Key one-time password system.
- Kerberos: private-key encryption with time-expired “tickets”.

- Smart card.
- Password prompt.
- Public key.

The most common authentication method is by password prompt, which is how most SSH installations are run out of the box.

However, public key encryption is worth investigating; it is considerably more secure than passwords, and by using it you can do away with all or most of your password typing.

Briefly, public key encryption relies on two keys: a public key to encrypt, which you don't keep secret, and a private key to decrypt, which is kept private on your local computer. The general idea is to run `ssh-keygen` to generate your keys. Press Return when it asks you for a passphrase. Then copy your public key to the remote computer's authorized_keys file.

The details depend on whether the computer to which you are connecting uses SSH1 or SSH2. For SSH1 type `ssh-keygen -t rsa1`, and copy ~/.ssh/identity.pub to the end of the file ~/.ssh/authorized_keys on the remote computer. For SSH2, type `ssh-keygen -t rsa`, and copy ~/.ssh/id_rsa.pub to the end of the file ~/.ssh/authorized_keys on the remote computer. This file might be called ~/.ssh/authorized_keys2, depending on your OpenSSH version. If the first one doesn't work, try the second. The payoff is you can log in without typing a password.

You can use a passphrase that keeps the private key secret on your local computer. The passphrase encrypts the private key using 3DES. At no time is your passphrase or any secret information sent over the network. You still have to enter the passphrase when connecting to a remote computer.

Authentication Agent

You might wonder: if we want to use a passphrase, are we stuck back where we started, typing in a passphrase every time we log in? No. Instead, you can use a passphrase, but type it only once instead of every time you use the private key. To set up this passphrase, execute `ssh-agent` when you first start your session. Then execute `ssh-add`, which prompts for your passphrase and stores it in memory, not on disk. From then on, all connections authenticating with your private key use the version in memory, and you won't be asked for a password.

Your distribution may be set up to start ssh-agent when you start X. To see if it's already running, enter `ssh-add -L`. If the agent is not running already, you need to start it, which you can do by adding it to your .bash_login, logging out and logging back in again.

Authentication Agent Forwarding

If you connect from one server to another using public key authentication, you don't need to run an authentication agent on both. SSH automatically can pass any authentication requests coming from other servers, back to the agent running on your own computer. This way, it never passes your secret key to the remote computer; rather, it performs authentication on your computer and sends the results back to the remote computer.

To set up authentication agent forwarding, simply run `ssh -A` or add the following line to your config file:

```
ForwardAgent yes
```

You should use authentication agent forwarding only if you trust the administrators of the remote computer; you risk them using your keys as if they were you. Otherwise, it is quite secure.

Traveling with SSH Java Applet

Many people carry a floppy with PuTTY or another Windows SSH program, in case they need to use an unsecured computer while traveling. This method works if you have the ability to run programs from the floppy drive. You also can download PuTTY from the web site and run it.

Another alternative is putting an SSH Java applet on a web page that you can use from a browser. An excellent Java SSH client is Mindterm, which is free for noncommercial use. You can find it at www.appgate.com/mindterm.

Conclusion

An SSH configuration can go wrong in a few places if you are using these various tricks. You can catch many problems by using `ssh -v` and watching the output. Of course, none of these tricks is essential to using SSH. Eventually, though, you may encounter situations where you're glad you know them. So give a few of them a try.

Daniel R. Allen (da@coder.com) discovered UNIX courtesy of a 1,200-baud modem, a free local dial-up and a guest account at MIT, back when those things existed. He has been an enthusiastic Linux user since 1995. He is president of Prescient Code Solutions, a software consulting company in Kitchener, Ontario and Ithaca, New York.