

SSH - Uncrackable

Adam B. Gurno

2005-10-06

Contents

1 License	2
2 About SSH	3
2.1 What is SSH? What is OpenSSH?	3
2.2 Why do we need SSH	3
3 Using SSH	4
3.1 SSH as Telnet	4
3.2 SSH and Keys - w00t!	4
3.3 SSH as FTP/rcp	5
3.4 SSH as rexec	6
3.5 SSH as encrypted pipe	8
4 Conclusion	10
5 About This Document	11

1 License

Attribution-NonCommercial-ShareAlike 2.5

You are free:

1. to copy, distribute, display, and perform the work
2. to make derivative works

Under the following conditions:

Attribution - You must attribute the work in the manner specified by the author or licensor.

Noncommercial - You may not use this work for commercial purposes.

Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

For any reuse or distribution, you must make clear to others the license terms of this work.

Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

2 About SSH

2.1 What is SSH? What is OpenSSH?

SSH stands for (S)ecure (SH)ell. It is a suite of protocols for remote authorization and information transfer. It is encryption-independent.

OpenSSH is a BSD product, from the same codebase as the closed-source SSH 1.2 product. SSH 1.3+ went to a more restrictive license, so it was forked and the rest is history.

Rather than ask what we can use it for, let's examine what it can replace:

- Telnet, Rlogin Remote command line access.
- FTP, rcp Authenticated file transfers. ¹
- Rsh Remote shell command execution
- SOCKS Connection forwarding-relaying software.

2.2 Why do we need SSH

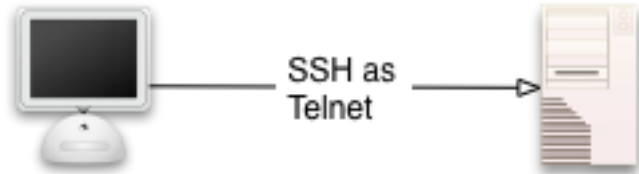
Considering the products that it is replacing, there are two reasons:

- Security The older protocols use cleartext communication, meaning that anyone could listen in to your sessions and steal whatever they'd like. What's more, some of the older tools were complete crap, filled with security holes galore. (FTP, I'm looking at you...)
- Bad Ideas FTP (sorry to pick on FTP, but it's a valid point) did stupid crap like 'ascii/binary' transfer methods. The ASCII transfer would attempt to convert the linebreaks in a document between DOS/UNIX formats. If you accidentally sent a binary file via ASCII transfer, you were hosed.

¹FTP is complete CRAP. Good riddance.

3 Using SSH

3.1 SSH as Telnet



This is the SSH usage that we're all familiar with. You can connect to a remote machine and open a shell.

The simplest way to connect to a box is just...

```
ssh hostname.tld
```

That will attempt to connect to a remote box using your current username. What's that? Your username is different on the other box? No problem...

```
ssh -l lusername hostname.tld
```

-or-

```
ssh lusername@hostname.tld
```

There! Now you go through the song and dance of passwords. We'll tell you how to avoid that, but first let's look at other useful ssh bits for this type of connection.

- C - Compress the traffic flowing through. Very useful when doing things like X connections and slow pipes.
- v - Verbose. Please dump 82 brazilian megs of information on my screen about the 1337 handshaking going on.
- q - Shut up. I don't care what's going on.
- p # - Connect to port X.
- V - Display version information and quit.
- l - You haven't forgotten already, have you?

3.2 SSH and Keys - w00t!

Since the dawn of man, Unix has been a machine of username/password queries. Beelyuns and beelyuns of `password:` prompts have been doled out and beelyuns and beelyuns of man-hours have been wasted while people go "Uuuhhhh..."

No more. NO MORE, I SAY.

Enter keys. You can generate a 'public key' that will allow you to enter a system nearly untouched. Never again be faced with a password question, which means that you will quickly forget what your password is and be completely stumped the next time you sudo.

Anyways, on your ORIGINATION (not destination) machine, do this:

```
ssh-keygen -t rsa
```

...and then hit enter a bunch of times. Don't bother with a key password. (For more on key passwords, see "SSH Public Key Passwords: Does my tinfoil hat make my ass look big?")

You're done! Ha, not really. This has now created an `id_rsa.pub` file (along with a few others) inside your `/.ssh/` dir. You need to move this file to every DESTINATION machine that you wish to connect to. ²

So, now that you have moved the `id_rsa.pub` file over, you need to cat it on the end of your "authorized_keys" file. ³

```
luser@destination ~/      $ cd ~/.ssh/  
luser@destination ~/.ssh/ $ cat id_rsa.pub >> authorized_keys  
luser@destination ~/.ssh/ $ chmod 600 authorized_keys
```

You're done! Unless you're working with connection from SSH to OpenSSH or version 1 or version 2 issues, in which case you've just wasted 5 minutes.

Still having problems? Use the `-v` key!

3.3 SSH as FTP/rcp



So, you need to move a file from point A to point B, but it's sheep porn and you'd rather not have everyone looking at it.

Enter `scp`, you pervert.

`scp` takes almost all of the same options that `ssh` does, with a few differences.

-C - Compress the traffic flowing through. Always do this.

-v - Verbose. You know what those means.

-q - No progress meter for you.

²See the section on `scp`

³If you don't have that file yet, it doesn't matter

-P # - Through port X. Little p vs. Big p? I always get it wrong.

-p - Preserve modes, modification times and the like.

-l - Was luser, now it means 'limit' as in kb/sec.

-r - Recursively. You want everything in /sheepporn to go over.

Notice that there's no flag for username? I guess that they figure if you're using scp, you're big enough to handle the user@host syntax.

```
scp -rCv ~/sheepporn/ luser@remotehost:
```

The colon at the end always gets me. It's really a separator between host and target file location, but I always forget.

```
scp -rCv ~/sheepporn/ luser@remotehost:pics/
```

See the target file in that example? Rather than just dumping /sheepporn/ into your home dir, you've put it inside of /pics/sheepporn/. While your perversion still gives me the willies, I admire your organizational skills.

It works the other way too:

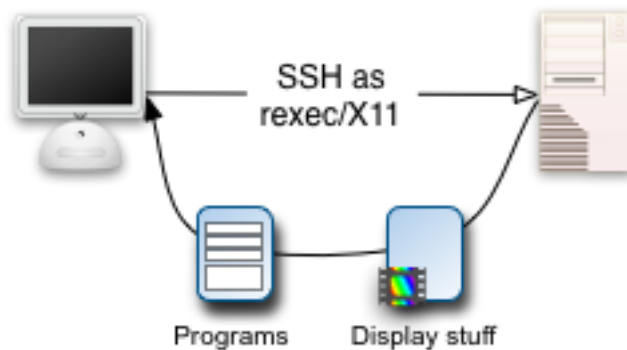
```
scp luser@remotehost:pics/ new-stuff-from-there/
```

You can also add wildcards:

```
scp -l32 luser@remotehost:music/*.mp3 toonz/new/
```

Enough of that.

3.4 SSH as rexec



Now it's getting fun. Nowadays, you can extend your desktop via remotely executed, graphical commands that are compressed and encrypted. For that, let's all say 'W00t.'

W00t.

These go through the standard ssh command, though you won't be going interactive, like the shell. Well, mostly.

- C - Compress the traffic flowing through. Always do this.
- f - Drop to background. Use this when issuing remote commands one at a time. See the examples for when to use this.
- v - Verbose. You know what those means.
- q - No progress meter for you.
- p - port again.
- l - Was luser, then wasn't, now is again.
- x - No X11 forwarding. Uh... ok.
- X - X11 forwarding.
- Y - Trusted X11 forwarding. Cygwin users need this.

You've taken the time to do the public key, right? 'Cause the really cool stuff works best that way.

```
ssh -CXf remotehost firefox
```

The important thing to notice is that we've added a command to the end of the standard ssh call. This means that the ssh program will run that remote program upon login and exit when the program quits. It's a guaranteed, single-use pipe.

So, the previous command does what? It opens X11 forwarding (necessary), it turns on the compression (smart), connects to the remotehost (using your 1337 public key authentication), executes firefox, and then sinks to the background.

Background? Yup. If you issue this from a command line, your commandline won't be affected. The firefox window will popup from the remote box while you continue working. You want more? Consider launching a remote program launcher, like the gnome-panel or the FVWM panel or one of the myriad of program bars. That way you can continue to launch programs through you X connection and you don't have to enter a single extra command.

Cooooool.

Sure, you could get lame:

```
ssh -CX remotehost
```

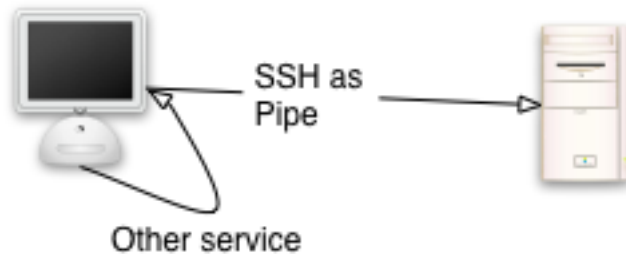
...which would give you a remote shell that you could launch program though. But what fun is that? Better than that, the same thing is:

```
ssh -CXf remotehost xterm
```

Or gnome-term or whatever you kids are using nowadays. Get off my lawn!

For my Cygwin homies: Wazzup? So, yous be chillin' on the Winders box, a'ight? And you be wantin' to do the remote shizzle a'ight? You gots da Cygwin hookup? Word. Y'all be needing the -Y dizzle, 'cause Gates is The Man, yo, and it ain't all that, and -X be all in yo face going "Whachoo gonna do, bitch?" I be pourin forties all day long for da number of times hos axe me that.

3.5 SSH as encrypted pipe



Okay, this is some Area 51 type stuff that I admit I don't completely understand.

All the standard ssh flag, and now this:

-L - Area 51 pipes

Example:

```
ssh -C username@remotehost -L localport#:remotehost:remoteport#
```

What this does is create a connection between your port and the remote port using SSH. This allow you to run things like VNC over an encrypted connection. Vanilla VNC doesn't know anything about encryption and not much about compression, so this is a really good idea.

Example:

```
ssh norlug.org -L 6666:norlug.org:5900
```

What this does is create a local port 6666 that, when you connect to it (localhost:6666), should act like port 5800 on norlug.org. So, if you'd like to VNC over to norlug, you'd actually connect to localhost, port 6666 and let SSH handle the back and forth.

So, you're thinking to yourself "I never run VNC, why do I care?" The beauty of this is that it works for **any** port, for **any** service that you'd like

to encrypt. Gopher, HTTP, XML passing, you name it. Any protocol can be sent through an encrypted pipe, easily.

Tip: When choosing a port number on your local machine, choose a port over 1024. Anything under that is considered privileged and reserved for root and system processes. Remote machine port numbers can be any port number that someone is listening on.

4 Conclusion

SSH is awesome. It will protect your documents, handle them correctly, allow you to become more productive, and to top it off, it's really easy to use.

May you never have to type telnet again.

5 About This Document

This article was written in \LaTeX using Vim and Elvis. This article was written for a presentation given to NORLUG - The Northfield Minnesota Linux Users Group. This article can be found online at <http://gurno.com/adam/norlug/ssh.pdf>.

Copyright 2005, Adam Gurno. Please see the LICENSE section for more details on your rights with this document.