SSH for good, not evil

A little about the power - and the responsibility

A taste

- Forget your password: publickey authentication
- Files at your fingertips: sshfs
- Reach the inaccessible: ProxyJump
- Wormholes: port forwarding
- There must be 50 ways to copy your file
- Kinda famous: publish a private key

But first...

- SSH, like SSL (the ‘s’ in https) is asymmetric encryption:
  - One key can only encrypt: you publish that part
  - The other key is private, and lets you decrypt

  So if I know your Pubkey, I can send something that only you can read.

- The first thing we use PK for is to agree on symmetric keys, because the math of PK is horrible and slow.
- But a side-effect of known public keys is mutual authentication. (Dirty secret: browsers preload tons of pubkeys, otherwise you wouldn’t be able to trust any website and surfing from a cafe would be a total bummer!)

OK, again

- “Keypair”: public part and private part.
- Everything depends on keeping privates private.
- Basic technique is used by HTTPS as well as SSH.
- Provides both encryption (non-snoopability) and mutual authentication (no imposters).

I can’t repeat this enough: still have to keep your private key private.

SSH, how to get it

SSH is a classic Unix thing: a versatile building block that has a million uses - but it’s also not very prescriptive: it doesn’t tell you want to use it for.

- Linux: OpenSSH ubiquitous
- MacOS: OpenSSH built-in but not obvious
- Windows: used to be somewhat hostile (PuTTY, ssh.com, etc)
  - You should probably just install MobaXterm
  - OpenSSH included with very recent Windows 10 (!!!)

Yes, I’m assuming commandline

Remember: SSH is a building block, a tool, not really an application.

Commandline gets you all the basics you need: connect to a cluster, submit jobs.

Commandline builds character - all the greats used commandlines!

Yes, there’s more than commandline...

Still basic Unix: windows and GUI

Unix/Linux GUI applications use X.

X does GUI over a network connection.

SSH can “tunnel” X from a remote Linux machine to your device.

It’s not great, but it works.

There are tweaks: X compression, VNC, etc

It’s still just an SSH tunnel. Wait, what’s an SSH tunnel!?!

SSH is just a tunneling machine

When you “connect” to a remote server, what actually happens?

- You login
- You see text and type stuff
- You logout or get disconnected

Couple bits more detail

There’s more to logging in than typing your password.

- Host authentication
- Negotiation of session key
- Optional host-based trust (no auth, really)
- Optional publickey auth
- Optional certificate auth
- Optional password auth

Notice that password is the last option...

I thought we already did public keys

SSH first creates an authentic, encrypted connection between the hosts, then waits for the user to do something

Usually, that means opening a login session: establishing who you are and connecting to a shell.

Again, password is the last way to do this.

Publickey authentication cannot be recommended too strongly!

Forget your password: publickey authentication

1. ssh-keygen
2. ssh-copy-id -i .ssh/id_rsa me@remotehost
3. ssh me@remotehost
4. profit passwordlessly

YOU CAN ONLY DO THIS ON A TRUSTED MACHINE.

Remember that the private key is the moral equivalent of your password: don’t let it escape.

Encrypt your private key

Use a password to unlock a thing that kinda sorta acts like a password? Think of it as a master password for a password-manager app. We call the key to your private key a “passphrase”. You could type this passphrase each time you need to login - pointless!

- Key agent holds the decrypted keys, uses them automatically when you need them to SSH.
- Run the agent only on secure machines, preferably only when you’re sitting in front of them.
- Agent can timeout (forget keys after N hours, etc)

keytypes

SSH has been around a long time - there are lots of deprecations.

- No one even mentions the ssh-1 protocol so forget I did.
- RSA keys used to be under a patent cloud.
- DSA keys were unclouded, but there are some obscure issues that probably make them unsafe to use. Same for ECDSA.
- All keys become more crackable, so to be conservative is to choose longer keys (doesn’t affect usability)
- Ed25519 probably the best choice, but RSA is pretty OK.

What keys look like

```
[hahn@hahn ~]$ ssh-keygen -P '' -C sample -t ssh-ed25519 -f sample
Generating public/private ssh-ed25519 key pair.
Your identification has been saved in sample.
Your public key has been saved in sample.pub.
[hahn@hahn ~]$ cat sample.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDOcuGunxD0DOvJpY46zDy+zx4ElSBDCfYHKGMAFAol7 sample
[hahn@hahn ~]$ cat sample
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACAznLhrp8Q9AzryaWOOsw8vs8eBJUgQwn2ByhjABQKJewAAAJDMjmFDzI5h
QwAAAAtzc2gtZWQyNTUxOQAAACAznLhrp8Q9AzryaWOOsw8vs8eBJUgQwn2ByhjABQKJew
AAAEDSvgV2Y7OVRGgR1NMTD9a7qD/41KG8CrDAKHOQlLmc8DOcuGunxD0DOvJpY46zDy+z
x4ElSBDCfYHKGMAFAol7AAAACWhhaG5AaGFobgECAwQ=
-----END OPENSSH PRIVATE KEY-----
```

Config files, always config files

- ~/.ssh - keep everything here
- ~/.ssh/id_rsa - an RSA private key
- ~/.ssh/id_rsa.pub - public part
- ~/.ssh/config - put stuff here to avoid typing
- ~/.ssh/authorized_keys - public keys go here

“Id_rsa” is just the default that ssh-keygen uses. You can, and should, give your keys mnemonic names.

I typically put my name, a purpose and/or year into the name/comment.

OK, back to tunneling

When you connect to a machine:

1. Host-host authentication
2. User authorization (pubkey or password)
3. Tunnel is created to a shell (command interpreter)

But notice - we've got a secure connection useful for other things, not just a (single) shell connection.

SSHFS

Basically: `sshfs remotehost:/some/dir local-mount-point`

- Sshfs uses ssh to connect to remotehost
- It runs sftp there, tunneling the connection
- It uses that access to make remote files appear local

(oh, yeah: sftp is a useful commandline tool on its own - instead of connecting to a remote shell, it connects to a really simple file get/put command. Also: scp)

Sshfs looks like this:

```
[hahn@hahn1 ~]$ df
Filesystem           1K-blocks        Used    Available Use% Mounted on
/dev/sda2             51474912    17238756     31598332  36% /
/dev/sda1               999320      182684       747824  20% /boot
/dev/sda3            177980276   164953804      3962488  98% /home
cedar:           247611681392 12765039384 222364289328   6% /home/hahn/mnt/cdr
graham:           68719476736  9148746752  59570729984  14% /home/hahn/mnt/gra
```

In other words, I have my Cedar and Graham home directory tries mounted as local filesystems on my workstation. I can edit files directly, copy between them, etc.

ProxyJump

```
[hahn@hahn1 ~]$ ssh -o proxyjump=graham.computecanada.ca gra1
Last login: Wed Mar 14 10:31:10 2018 from gra-login3.graham.sharcnet
***
Welcome to the ComputeCanada/SHARCNET cluster Graham.
...
[hahn@gra1 ~]$
```

What just happened?

I used SSH to connect to Graham to open a tunnel to the SSH server on gra1, which is not exposed to the internet. This applies to many internal hosts accessible through an exposed host...

ProxyJump is proxying SSH

- Authenticate, authorize, create session on graham
- Create tunnel connecting local port 22 to port 22 on gra1
- Authenticate, authorize, create session on gra1

Note that this can be chained - jump-to-jump, and that gra1 is authenticating with the **local** host, not the jump host.

Here's our first "great power, great responsibility" situation: don't try to use this to abuse compute nodes of our cluster.

Port forwarding is like a cosmic wormhole

```
ssh -R 10.53.0.21:28000:license-matlab.mcmaster.ca:27000 \
  -R 10.53.0.21:27001:license-matlab.mcmaster.ca:27001 \
  sharcnet@iqaluk.sharcnet.ca
```

I use this actual command (with some extras) to tunnel the McMaster campus matlab license. It creates a wormhole on Iqaluk whose remote is license-matlab.

Again, ssh is doing authentication, authorization first, then setting up a session. That session simply does two port forwards - it listens on Iqaluk for connections to the given ports, shoves the traffic through the tunnel to the remote side, where it has a socket connecting to the license server

Lots of forwarding

- Previously mentioned X forwarding
- ProxyJump is forwarding of SSH’s port 22
- There’s also -L for forwarding when the remote end of the wormhole is on the ssh destination.
- There’s even SOCKS dynamic port forwarding that can be used as a general proxy for, eg, web surfing

Ways to copy files

- Previously mentioned SSHFS, SFTP, SCP
- Rsync synchronizes two filesystem trees:
  - `rsync -a cedar.computecanada.ca:project/something .`
  - This ensures that there is a local file/tree called “something” whose contents/size/permissions match the remote version
  - Does this lazily! Doesn’t copy files it doesn’t need to - doesn’t even copy sections of files that are already identical!
  - Both source and destination can be remote! (Bit tricky though, since source has to be able to connect to destination directly…)
  - Rsync is great if you want to sync - to periodically update a copy of a tree
  - Warning: Rsync cares a lot about whether there’s a trailing ‘/’ on the source
  - Since it’s lazy, it’s also incremental! IE: if interrupted, will continue where it stopped.

Publish a private key!?!

Remember ~/.ssh/authorized_keys?

An interesting extension is that you can prefix a key with some assertions about how the key can be used:

```
command="/home/hahn/bin/something -fancy" ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC/B3j87w/HVgl+t5G6Fb1QWmX010Ij7PPdm8gTJ6xb83sH2kTJHUWZ+ZV946t
J75qaDPOTeNVf7OFmtrgnDBldIHK6Jty2y6rjX/uJZBCBsaZHMXs6HlG7P/KOO2K+M6my19tJtIV1+lqAnoUUJwt8D2
fgbpcSJZvu8i0Rvkr9QRrIR/U9/yyTlDjVvpFUhq7fgPTxf7QmUDgsstsgAUReokOScVyzF9CC6Ejxxwu0UJIxrglIk
cU/jmHqxu7/JXY3ex4Im0lrgQSOcwvwiWybPkonr9zXCT+VTFagavIfoVLQz13z4U6ny3OA6qK73eaSmzevrgiY0pTE
eLMxLqPP fancy-key
```

SSH calls this “forced command” key

- Key can only run that command
- If you “ssh -i fancy-key user@remote somethingelse”, the “somethingelse” is NOT what gets run.
- “Somethingelse” is available to the forced command as $SSH_ORIGINAL_COMMAND
- Other important key constraints: from=”my-host.com”, no-pty, no-X11-forwarding, no-port-forwarding, no-agent-forwarding

Private key as capability

This means we have a private key which can only do a specific thing, on a specific host (where it’s installed in authorized_keys), potentially only from specific places.

You can publish this private key!

Or at least use it to give a specific, limited capability to someone else.

For instance, ability to access a specific file, command, etc.

This is also the ONLY situation where it’s justified to use no passphrase!