An Overview of SSH

Presentation to Linux Users of Victoria

Melbourne, August 26, 2017

http://levlafayette.com

Utilisation and Rationale

- The most common use of SSH (secure shell) is remote login access to computer system. However, any network service can use ssh (e.g,. file transfers, remote mounts, proxy server tunneling etc). It is built on a small mountain of RFCs (4250-56, 4419, 4432, 6668 etc)

- The original reason for SSH is to provide a replacement for insecure remote system applications such as telnet, rlogin, ftp etc. All of these send information (e.g., passwords) in plain-text.

- Practical task: Use a network analyser to capture such information (e.g., https://www.wireshark.org/), and capture sessions and passwords. Short example available on Youtube. https://www.youtube.com/watch?v=xShwyUq-uHk . Example session with accessing (for example) telnet rainmaker.wunderground.com

A High Level Architecture

- SSH is based around public-key cryptography. This key-pair system requires a public key which is distributed so that others can send messages and a private key so the authenticated recipient can read them. On UNIX-like systems, a list of authorised public keys are usually kept in the ~/.ssh/authorized_keys file.

- SSH is built with a client-server architecture. An SSH client is an application that is used to connect via SSH to a remote system. An SSH server is an application which accepts connections from remote system. The most popular client and server is OpenSSH. If you are insane use a proprietary implementation.

- PuTTY is a useful GUI SSH client which can also be used on MS-Windows. The client is already installed on most systems; unless you want to login to your system remotely you don’t *need* the server.

Installation and Management of OpenSSH Server

• OpenSSH is commonly installed via package management (e.g., sudo apt-get install openssh-server openssh-client). This will install the OpenSSH server and start the daemon on port 22 by default.

•MS-Windows has never had an SSH-server. In 2015 Microsoft announced (for the third time) that it would soon offer OpenSSH soon. It is currently (as of last night) up to v0.0.19.0 (pre-release, non-production ready)

•The configuration file is usually located in /etc/ssh/sshd_config. Common options to change; port number for SSH, allow root logins.

• Usual service method to stop/start/restart (e.g., /etc/init.d/ssh [stop start restart] or service ssh [stop start restart]) or systemctl [stop start restart] sshd.service)

Creating and Tracking Keys

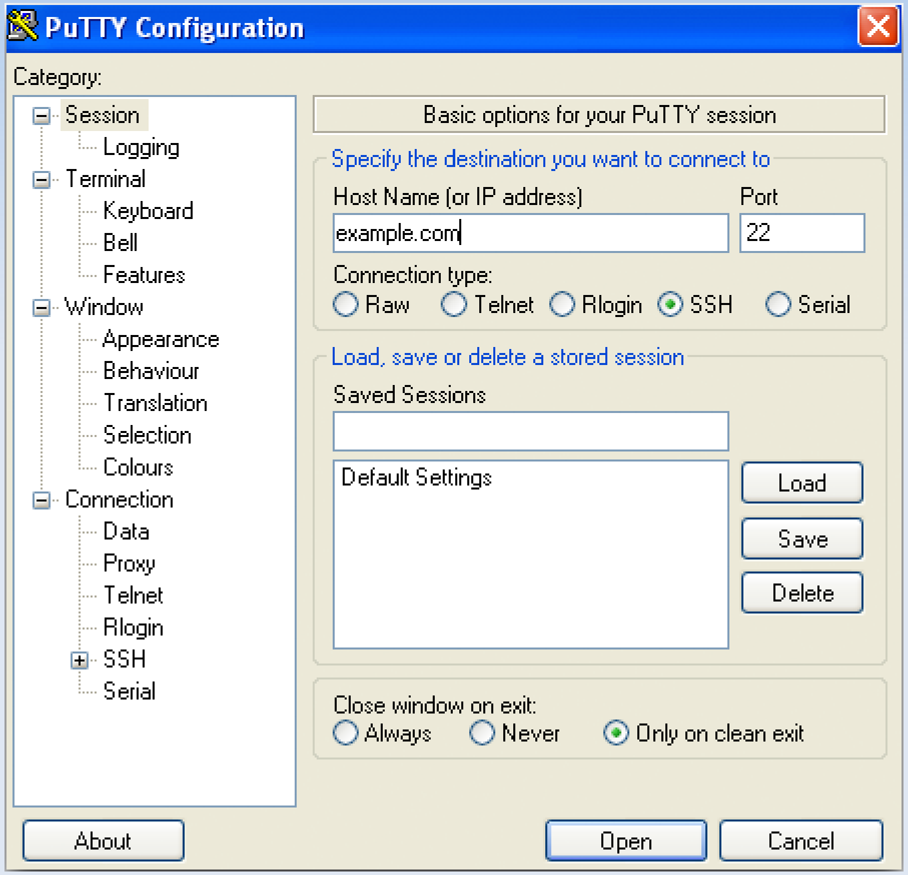
• SSH keys should be generated on the computer you wish to log in from with the key generator; `ssh-keygen`. Common options include -t (type, usually rsa), -b (size in bits, 4096 is a good choice, -f filename for multipe keys, -C to describe the keys) e.g., `ssh-keygen -t rsa -b 4096`. Follow the prompts and choose a secure passphrase. After this the `~/.ssh/` directory will have an `id_rsa` file (private key, don't share) and a `id_rsa.pub` key which can be shared. If the key on a remote system has changed used `ssh-keygen -R remote` to remove.

• Keeping track of passwords can be a pain; `ssh-agent` handles these passwords in background. In contrast `ssh-add` adds it to the list maintained by `ssh-agent`. The process is to initiate the agent with `eval $(ssh-agent)`, and then `ssh-add`, entering the private key passphrase. Kill the process prior to logging out, for example by adding `kill $SSH_AGENT_PID` to `.bash_logout`

Client Use of SSH

- **A user connects with a remote system by invoking their SSH client with a variety of options, and the username and address of the remote system. The typical options than are passed are include the -p (port, if nonstandard), -i (to specify a partiuclar identity file), -X or -Y (to enable X-windows forwarding in normal or trusted mode).**

For example: ssh -Y lev@spartan.hpc.unimelb.edu.au



Host Name (or IP address): example.com
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Close window on exit: Always Never [✓ Only on clean exit]

File Transfers and Directory Mounts

- Use of SSH to copy files between systems (scp) is one of the most common uses of the protocol. It uses the same sort of commands options as the cp command (e.g., -r for directories), and is based on the source-destination convention. e.g., `scp testfile spartan:files/`. Synchronisation applications like rsync also run over SSH.

- Another common use is the remote mounting of files. using SSHFS (SSH filesystem). The client interacts with the remote system with SSH File Transfer Protocol (SFTP), which conducts FTP-operations with an SSH channel.

Passwordless SSH

- Remote logins to systems with an SSH client provide encryption, and by themselves uses a password authentication system. However it is common to use public-private key authentication to allow for passwordless SSH connections.

- This is simply a matter of copying the public key to your account on a remote system and appending to the SSH authorized_keys file. For example:

`ssh-copy-id -i .ssh/id_rsa.pub user@remote`

SSH Client Configuration Files

• If you have a range of hosts to connect to, different ports, multiple identity keys etc, keeping track of these could be onerous. One method would be to add them all as an alias in a login file (e.g., `alias spartan='ssh lev@spartan.hpc.unimelb.edu.au').

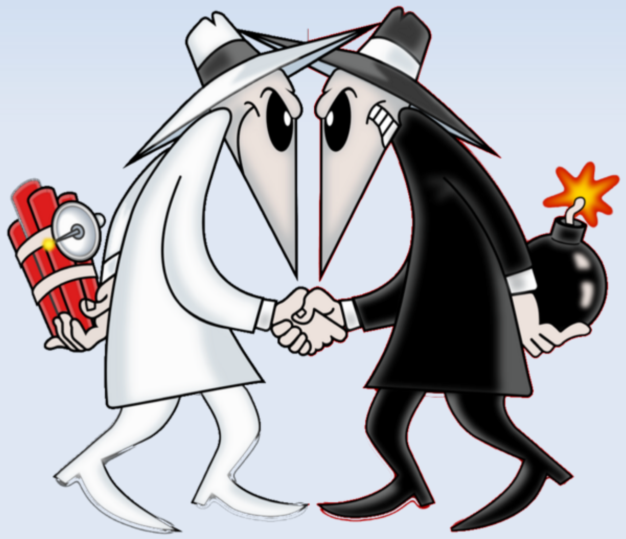
•An even more elegant method however is to use an SSH config file, which provides all the alias features, plus client-side configuration features (such as keepalives). The ssh config file is kept in `~/.ssh/config` or `/etc/ssh/ssh_config` for global settings. Config directives are subject to local commands.

•SSH config files plus passwordless SSH combined are very useful tools that will free up your memory for more important tasks.

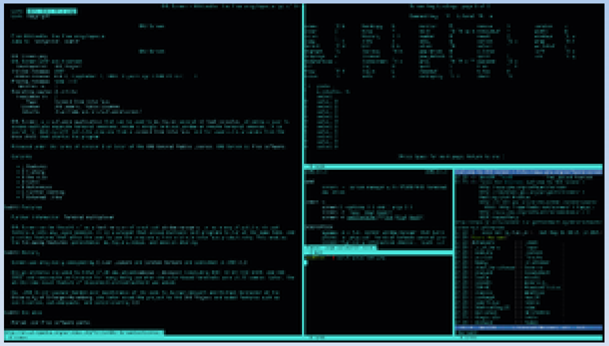
SSH Agent Authenication and Commands

- Authentication agent connection can be forwarded with the -A option. Note that the intermediate could hijack the session (not the keys); nevertheless this is obviously better than storing one's private key on remote servers to access even more remote servers. Example: `ssh ninjadan -A, ssh spartan-m`.

- SSH can be used to run a single command or a set of commands on remote systems. For example `ssh spartan ls > dirlist.txt` will execute remotely and save the results locally, whereas `ssh spartan "$(cat commands.txt)"` will execute commands.txt on the remote system.



SSH and GNU Screen



• SSH works very well with GNU Screen, a terminal multiplexer which allows a user to access multiple separate login sessions inside a single terminal window.

•A screen session can be started up (`ssh -t spartan /usr/bin/screen -xRR`), a second started (Cntrl+a C), detached (Ctrl+a d), then switch (Ctl+a n), then detached. Another alternative is to open it up when connected (`ssh spartan`, `screen`, detach), then connect to it remotely `ssh -t spartan screen -r test`).

SSH Port Forwarding

• Port forwarding with SSH creates a secure relay connection between system. It is very useful for tunneling unencrypted protocol information (e.g., IRC, VNC). The most common form is local port forwarding, where the SSH client connects to an SSH server and then the destination system. There is also remote port forwarding (server to client, then destination) and dynamic (multiple programs via client to server then to several destinations).

• Encrypted webbrowsing can be used by using the SOCKS Proxy with modifications to the webbrowser with (for example the 8080 port), with `ssh -D 8080 -C -N username@remotesite` (bind to port, compress data, non-execution)

• Another example is allocated by the user (local) and the other based on the destination (usually common port numbers). e.g., VNC desktop example (to itself! e.g., `ssh -L 5900:localhost:5900 <host>`).

SSH Verification

• When setting up an initial connection the possibility is raised that you might be facing a man-in-the-middle attack (short example: https://www.youtube.com/watch?v=4abglcnDBcY).

•However this can be circumvented because when new key pairs are created they also create a unique fingerprint and ASCII-art image. When generating a key for a new server (e.g., `ssh-keygen -t rsa -C newserver -f .ssh/newserverkey`) save the key fingerprint and randomart image. Then you can fetch a key's fingerprint and randomart image anytime to compare and make sure they have not changed: (`ssh-keygen -lvf newserverkey` or for all of them, `$ ssh-keygen -lvf ~/.ssh/known_hosts`).

•A modification on a the ssh_config on a local computer (modify `/etc/ssh/ssh_config` to turn on `VisualHostKey yes`) followed by a login to remote systems can compare fingerpints (a key snippet) and the randomimage.

SSH History and Security

• Tatu Ylönen, a researcher at Helsinki University of Technology first developed SSH in 1995 after a password sniffing attack. It was later commoditised and in 1999, forked the older 1.2.12 version which the last released under a open-source license. For a while there was two SSHs in common circulation (OpenSSH and OSSH). A major security flaw was discovered in the proprietary version in 1998, and soon OpenSSH became the single most popular implementation.

•There were some inherent design flaws in SSH-1, and in 2006 it was replaced by SSH-2, which includes more sophisticated key exchange technology and message authentication codes. Some SSH implementations only support SSH-2 as a result. In 2008 a security flaw was discovered which allowed for the plain-text recovery of 32 bits from a block of encrypted text; this was easily circumvented by using a different default encryption mode. In 2017 it was revealed via Wikileaks that the CIA was using methods to hijack user credentials from active SSH sessions.

Future Developments

- There have been several developments in parallel SSH which presumably will become part of the mainstream in the future. The core idea is to read a hostfile launch a command which is executed on the remote systems in the hostfile. Technically these could be implmented as a loop but if the tasks take an extended period of time...

- Four main implementations; Parallel SSH, Cluster SSH, ClusterIT, and Distributed Shell. Parallel SSH has a parallel shell, parallel scp, parallel rsync, and parallel kill. ClusterSSH creates an xterm session on each system subject to to the multiple commands, and has one controller terminal as well. Distributed Shell (dsh) is the most popular, also works on a machine list (e.g., `/usr/local/etc/machines.list`), passwordless ssh, and can return results to the local terminal.

- Why aren’t we using SSH for everything?

THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org