Getting the Most From SSH

Brooks Davis
<brooks@one-eyed-alien.net>

Outline

- SSH feature overview
- Basic usage
- Advanced usage
  - Client configuration, Public key authentication and authorization, Port forwarding, ProxyCommand
- Specialized versions

Major features

- Cryptographically secure protocol
- Remote access (telnet, rsh)
- Remote command execution (rsh)
- File transfer (rcp, ftp)
- X11 connection forwarding
- TCP port forwarding

The SSH Protocol

- Connects to a remote host and uses Diffie Hellman or RSA public key cryptography to exchange a secret key.
- Uses secret key for transport encryption with a symmetric cypher, usually using Blowfish, AES or 3DES.
- Also supports PKI authentication and authorization.

The SSH protocol

- There are two versions of the SSH protocol, 1 and 2. Version 1 should not be used.
- SSH operates over a single data stream, usually a TCP session.
- Internally, SSH multiplexes multiple streams over its encrypted channel. These streams are used to support X11 connection forwarding, port forwarding, and agent forwarding.

Remote access

- Connecting to a host
  - ssh <host>
- Connecting as a specific user
  - ssh <user>@<host>
  - ssh -l <user> <host>

Remote execution

- Run a simple command
  - ssh <host> <command>
- Run an interactive command
  - ssh -t <host> <command>
- Run an X11 command
  - ssh -X <host> <Xcommand>

File copy

- Copy a file from localhost to your home directory on remote host
  - scp <src_path> <host>:
- Copy a file to a specific location on a remote host
  - scp <src_path> <host>:<dest_path>

File copy

- Copy a file to the local host
  - scp <host>:<src_path> <dest_path>
- Copy a file from host1 to host2
  - scp <host1>:<src_path> <host2>:<dest_path>
  - ssh <host1> scp <src_path> \
    <host2>:<dest_path>

SFTP

- Provides an FTP-like interface to a remote system secured using SSH
  - sftp <host>

Client configuration

- Evaluated in order (first setting wins):
  - Command line options
  - User configuration in ~/.ssh/config
  - System wide defaults in etc/ssh/ssh_config or etc/ssh_config

Config Example

```
Host *.example.org
	User eouser
	ForwardX11 yes
Host *.sub.example.com
	User suser
Host *.example.com
	User ecuser
Host *
	ForwardX11 no
```

Set the user for each site, defaulting to the local user for unlisted sites.

Do not forward X connections by default, except to hosts at example.org.

Host keys

- Host keys are used to verify the authenticity of a host during the connection process
- The public keys of verified hosts are stored in etc/ssh/known_hosts and/or ~/.ssh/known_hosts
- Either whole keys or key fingerprints may be verified

User keys

- User keys authenticate users to hosts
- Three types are supported:
  - DSA: ~/.ssh/id_dsa, ~/.ssh/id_dsa.pub
  - RSA: ~/.ssh/id_rsa, ~/.ssh/id_rsa.pub
  - RSA1 (obsolete): ~/.ssh/identity, ~/.ssh/identity.pub

User keys

- Keys are generated using ssh-keygen(1)
  - ssh-keygen -t dsa
  - ssh-keygen -t rsa
- The ssh-keygen supplied with OpenSSH can also convert between OpenSSH format key files and “SECSH Public Key File Format” files as used by some commercial implementations.

User keys

- To use keys to authenticate to a host, place your public key in the authorized_keys file on the target host (usually under ~/.ssh/).
- By default ssh will attempt to authenticate using available keys.
- You will have to enter your pass-phrase each time you log in unless you configure an SSH agent.

SSH agent

- An SSH agent stores decrypted copies of keys loaded into it to allow automatic, key based authentication.
- Starting an agent:
    - `` eval `ssh-agent` ``
- Adding your keys:
    - `ssh-add`

SSH agent startup

- While the agent can be started by hand, it is generally better to start it automatically.
  - Usually done in startup/shutdown script scripts.
  - Can be done by PAM to start an agent as part of the login process.
  - Agents may also be forwarded between hosts.

SSH agent startup: csh/tcsh

```
# ~/.login
if( ! ${?SSH_AUTH_SOCK} && -f `which ssh-agent` ) then
	eval `ssh-agent -c`
endif

# ~/.logout
if ( ${?SSH_AGENT_PID} ) then
	echo killing agent ${SSH_AGENT_PID}
	kill ${SSH_AGENT_PID}
endif
```

SSH agent startup: bash

```
# ~/.bash_login
if [ -x `which ssh-agent` -a -z "${SSH_AUTH_SOCK-}" ];
then
      eval `ssh-agent -s`
fi

# ~/.bash_logout
if [ -n "${SSH_AGENT_PID-}" ]; then
      kill ${SSH_AGENT_PID}
fi
```

SSH agent startup: .xinitrc

```
if [ -f `which ssh-agent` -a-z "${SSH_AUTH_SOCK-}" ]; then
        KILL_SSH_AGENT=1
        eval `ssh-agent -s`
        ssh-add &
fi

# XXX: Start your window manager here

if [ -n "${KILL_SSH_AGENT}" ]; then
        echo "killing ssh agent ${SSH_AGENT_PID}"
        kill $SSH_AGENT_PID
fi
```

Dedicated keys

- In addition to normal user keys, dedicated keys (typically stored unencrypted) may be used to automate tasks.
- Extended options in the authorized_keys file allow restrictions to be placed on a key's use to limit damage if the key is compromised.

Key restrictions

```
# Normal key
1024 33 12121...312314325 user@example.com
#
# Only from example.org and not from bad.example.org
from="*.example.org,!bad.example.org" 1024 35 23...2334 user@example.net
#
# Automatically run "dump /home", do not allow allocation of a pseudo terminal or
# port forwarding
command="dump /home",no-pty,no-port-forwarding 1024 33 23...2323 backup.example.net
#
# only allow limited forwarding of ports
permitopen="10.2.1.55:80",permitopen="10.2.1.56:25" 1024 33 23...2323
```

Key restrictions

- Forcing the command in the authorized_keys file is less of a restriction than it appears.
- The submitted command is passed to the forced command via the SSH_ORIGINAL_COMMAND environmental variable where it can be executed after appropriate filtering.
- Writing a command filter is non-trivial, but may be worth while in some cases.

Key restrictions

```
#!/bin/sh
# Simple ssh command script.
# From "Using Rsync and SSH" http://www.jdmz.net/ssh/
case "$SSH_ORIGINAL_COMMAND" in
    *\&*)
        echo "Rejected"
        ;;
    *\;*)
        echo "Rejected"
        ;;
    rsync\ --server*)
        $SSH_ORIGINAL_COMMAND
        ;;
    *)
        echo "Rejected"
        ;;
esac
```

Port forwarding

- Port forwarding allows you to make a TCP port on the local or remote host work like a connection to another port reachable from the remote or local host respectively.
- Port forwarding can be used to support secure an insecure application or to access a service that is inaccessible from the local or remote host.

Port forwarding

- By default, forwarded ports are bound to localhost and only allow connections from localhost. This may be changed with the -g option.
- Basic local forwarding:
  - ssh -L<localport>:<targethost>:<targetport> <host>

HTTP over SSH

- Using SSH and a proxy server to access restricted websites
  - ssh -L 8080:proxy:3128 gateway.restricted.example.com

```
/* proxy auto-configuration script */
function FindProxyForURL(url, host)
{
    if (dnsDomainIs(host, ".restricted.example.com"))
        return "PROXY localhost:8080;";
    return "DIRECT";
}
```

Securing VNC

- VNC lacks any sort of useful transport security.
- If VNC servers are placed on a private network, SSH can provide that security.
  - `ssh -L 5900:<vnchost>:5900 <gateway>`
  - `vncviewer localhost`

Securing VNC

- If using tightvnc client:
  - vncviewer -via <gateway> <vnchost>
- Hint: use vncreflector to add tight encoding support to old vnc servers

ProxyCommand

- In addition to the standard mode of operation where the ssh client makes a TCP connection to the remote host, an external program can be used to make the connection another way.
- This program is specified by the ProxyCommand configuration option.
- The command should take two arguments, target host and target port.

ProxyCommand

- Behave normally, except use netcat to make the connection
    - ssh -o "ProxyCommand nc %h %p" <host>
- Connect through a gateway host
    - ProxyCommand ssh <gateway> nc %h %p

ProxyCommand

- Use an SSH server on the POP3 port of your home server to work from an internet cafe with a stupid firewall
  - ProxyCommand ssh -p 111 <home_server> nc %h %p
- SSH through an HTTP proxy using corkscrew
  - ProxyCommand corkscrew proxy.example.com 8080 %h %p

High performance file transfer

- Myth: encryption makes scp slow!
- Over long, fat pipes, scp is slow due to the use of a 64K hardwired window!
- High Performance Enabled SSH/SCP sets the maximum window using getsockopt(): 195+Mbps
    - http://www.psc.edu/networking/projects/hpn-ssh/

Questions? Comments?

CACert Assurance

- Requirements
  - CACert Identity Verification Form
  - Two forms of government issued photo ID
  - $20+ donation to The FreeBSD Foundation with Name, Address