Installing and Using SSH on Linux

Phil Karn

karn@ka9q.ampr.org

Secure Shell - SSH

- Originally designed and written by Tatu Ylonen (ylo@ssh.fi) as a secure, drop-in replacement for the BSD UNIX rsh/rlogin/rcp suite
- Provides automatic, strong, cryptographic authentication and confidentiality
- Remarkably easy to install and use
  - now almost universal on UNIX servers

Where to get SSH

- ftp://ftp.cs.hut.fi/pub/ssh
  - two protocol versions, 1.2 and 2.0
  - 2.0 has more restrictive licensing provisions
  - Most people still use 1.2 - released under GPL
- http://www.cryptography.org/cgi-bin/crypto.cgi/ssh/
  - my version of 1.2.26 with fast x86 DES code
  - export-controlled site (must be US citizen or permanent resident)

Installing SSH

- # tar xzvf ssh-1.2.26.tar.gz
- # cd ssh-1.2.26
- # ./configure
- # make install
- That’s it!
  - you might have to edit a boot script to start sshd

Using SSH: the ssh command

- ssh [-C] remotehost command
  - executes ‘command’ on host ‘remotehost’, with standard input, output and error to the local pipeline
  - will prompt for password or passphrase unless the authentication agent is in use
  - -C option enables compression; advised on most paths beyond a fast local LAN

The scp command

- scp [-C] remotehost:remotefile localfile
- scp [-C] localfile remotehost:remotefile

The slogin command

- slogin [-C] remotehost
  - logs you into ‘remotehost’ just like rlogin
  - Also sets up tunneled X-windows server connection

TCP port forwarding

- A *very* powerful and useful feature!
- Began as clever mechanism to handle remote X windows applications
- Became general-purpose TCP tunneling feature; can tunnel connections in either direction once a SSH session is set up
- Routinely used to websurf, fetch and send mail with Eudora, etc

Remote X windows

- When you log into a remote host conventionally, you set the $DISPLAY variable so that remote X applications can connect back to your local X display, e.g. set DISPLAY=myworkstation:0
- Problems with security and firewalls:
  - firewall may block the inbound connection
  - others may connect to your X server and do nasty things

SSH X forwarding

- With ssh/slogin, the remote sshd posts a listen on local TCP port 6000+n and sets DISPLAY=:n.0
- Remote X applications connect to what they think is a secondary local X display
- SSH intercepts and forwards over the encrypted TCP channel to the local X display

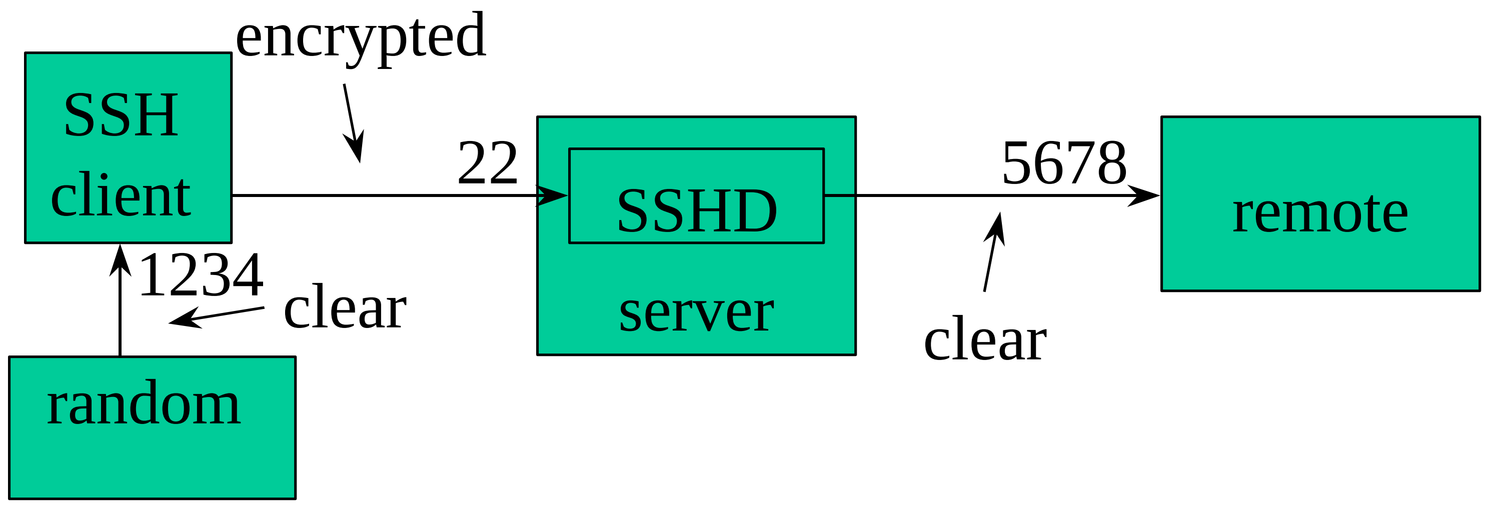
SSH forwarding advantages

- No inbound connections need be allowed by firewall
- Everything is transparently encrypted in transit over the existing outbound TCP connection
- Firewalls only need permit outbound TCP connections to port 22 (ssh)

General Purpose TCP Forwarding

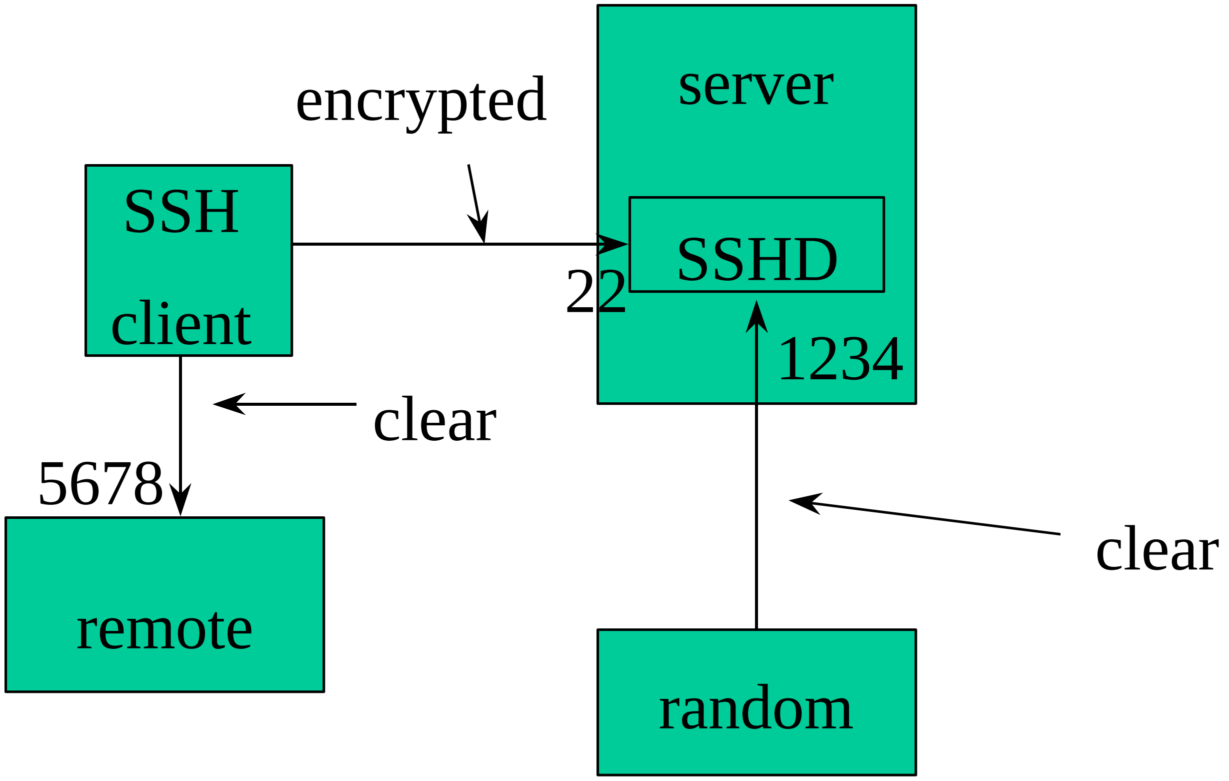
- ssh -L1234:remotehost:5678 serverhost
  - Local ssh client listens to port 1234
  - Connections to local port 1234 are automatically patched to connections from serverhost to TCP port 5678 on remotehost
- ssh -R1234:remotehost:5678 serverhost
  - Ssh on serverhost listens to port 1234
  - Connections are forwarded to port 5678 on remotehost

SSH Local Tunneling



```
client$ ssh -L1234:remote:5678 server
random$ telnet client 1234
```

SSH Remote Tunneling



```
client$ ssh -R1234:remote:5678 server
random$ telnet server 1234
```

Example: Secure Websurfing

- ssh -L3128:oceana.nlanr.net:3128 oceana.nlanr.net
- Configure netscape to use 127.0.0.1:3128 as http/ftp proxy
- HTTP requests are transparently tunneled across the encrypted SSH session to web proxy on oceana.nlanr.net

Example: Secure Email

- ssh -C -L110:popserver:110 -L25:smtpserver:25 sshserver
- A local client (e.g., Eudora) may now connect to ports 110 and 25 on ssh client system to fetch and send mail
- Mail will be compressed and encrypted over the SSH connection to sshserver
  - but will be in clear on both ends

Inside SSH

- Session encrypted with IDEA, 3DES, Blowfish, Arcfour, DES (deprecated)
- Session key generated by client, doubly encrypted with RSA and sent to server
- Two RSA keys:
  - "host key" - fixed 1024 bit RSA key
  - "server key" - 768 bit RSA key, changes every hour

Why two RSA keys?

- The static host key authenticates the server to the client, which caches these public keys
- When the server key is changed, the old key is deliberately destroyed
- This prevents recorded traffic from being decrypted even if all long-term secret keys are subsequently seized
- This is *perfect forward secrecy*

What about the host key?

- On the first connection to a ssh server, the client fetches the server's host key and adds it to a local list, unverified
- This raises the possibility of a man-in-the-middle masquerading as the server
- If this is a concern, you can pre-load your host key database, or distribute it signed with PGP, etc

User Authentication

- User passwords, typed down the encrypted channel
  - can be disabled for increased security
- RSA challenge/response with a personal user key
  - secret key kept encrypted on client machine
- Various other options, including Kerberos

User Authentication Agent

- Rather than type a password or pass phrase for every connection, SSH provides an optional authentication agent that can automatically answer RSA challenges
- Clients talk to agent in two ways:
  - shared UNIX file descriptor inherited by child processes
  - UNIX domain sockets in protected directory

Using the authentication agent

- ssh-agent bash
- ssh-agent startx
  - for X windows users
- ssh-add
  - prompts for passphrase, adds to agent
- ssh-add -d
  - destroys previously entered passphrase

Config files in ~/.ssh

- identity
  - user’s private RSA key, encrypted with passphrase
- identity.pub
  - user’s public RSA key
- known_hosts
  - list of known host public keys
- authorized_keys
  - list of public keys that can authenticate this user

To learn more

- Read the documentation!
- SSH has *many* configuration options and optional features; fortunately, the defaults are pretty reasonable