

An introduction to SSH

Lucas Nussbaum

lucas.nussbaum@univ-lorraine.fr

Licence professionnelle ASRALL

Administration de systèmes, réseaux et applications à base de logiciels libres



UNIVERSITÉ
DE LORRAINE



nancy Charlemagne
Département Informatique

License: GNU General Public License version 3 or later
or Creative Commons BY-SA 3.0 Unported
(see README.md)

Plan

- 1 SSH basics
 - SSH 101
 - Public-key authentication
 - Checking the server's identity
 - Configuring SSH

- 2 Advanced usage
 - SSH as a communication layer for applications
 - Access remote filesystems over SSH: sshfs
 - SSH tunnels, X11 forwarding, and SOCKS proxy
 - Jumping through hosts with ProxyCommand
 - Triggering remote command execution securely
 - Escape sequences

- 3 Conclusions

Introduction

- ▶ SSH = Secure SHell
- ▶ Standard network protocol and service (TCP port 22)
- ▶ Many implementations, including:
 - ◆ OpenSSH: Linux/Unix, Mac OS X ← this talk, mostly
 - ◆ Putty: Windows, client only
 - ◆ Dropbear: small systems (routers, embedded)
- ▶ Unix command (ssh); server-side: sshd
- ▶ Establish a **secure communication channel** between two machines
- ▶ Relies on cryptography
- ▶ Most basic usage: **get shell access** on a remote machine
- ▶ Many advanced usages:
 - ◆ Data transfer (scp, sftp, rsync)
 - ◆ Connect to specific services (such as Git or SVN servers)
 - ◆ Dig secure tunnels through the public Internet
- ▶ Several authentication schemes: password, public key

Basic usage

- ▶ Connecting to a remote server:

```
$ ssh login@remote-server
```

↪ Provides a shell on *remote-server*

- ▶ Executing a command on a remote server:

```
$ ssh login@remote-server ls /etc
```

- ▶ Copying data (with scp, similar to cp):

```
$ scp local-file login@remote-serv:remote-directory/
```

```
$ scp login@remote-serv:remote-dir/file local-dir/
```

Usual cp options work, e.g. -r (recursive)

- ▶ Copying data (with rsync, more efficient than scp with many files):

```
$ rsync -avzP localdir login@server:path-to-rem-dir/
```

Note: trailing slash on source matters with rsync (not with cp)

◆ `rsync -a dir1 u@h:dir2` ↪ dir1 copied inside dir2

◆ `rsync -a dir1/ u@h:dir2` ↪ content of dir1 copied to dir2

Public-key authentication

- ▶ General idea:
 - ◆ Asymmetric cryptography (or public-key cryptography):
 - ★ The public key is used to encrypt something
 - ★ Only the private key can decrypt it
 - ◆ User owns a private (secret) key, stored on the local machine
 - ◆ The server has the public key corresponding to the private key
 - ◆ Authentication = *<server> prove that you own that private key!*
- ▶ Implementation (*challenge-response authentication*):
 - 1 Server generates a nonce (random value)
 - 2 Server encrypts the nonce with the Client's public key
 - 3 Server sends the encrypted nonce (= the challenge) to client
 - 4 Client uses the private key to decrypt the challenge
 - 5 Client sends the nonce (= the response) to the Server
 - 6 Server compares the nonce with the response

Public-key authentication (2)

- ▶ Advantages:
 - ◆ The password does not need to be sent over the network
 - ◆ The private key never leaves the client
 - ◆ The process can be automated
- ▶ However, the private key should be protected (what if your laptop gets stolen?)
 - ◆ Usually with a passphrase

Key-pair generation

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): [ENTER]
Enter passphrase (empty for no passphrase): passphrase
Enter same passphrase again: passphrase
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
f6:35:53:71:2f:ff:00:73:59:78:ca:2c:7c:ff:89:7b user@my.hostname.net
The key's randomart image is:
+--[ RSA 2048 ]-----+
  ..o
  (...)
  .o
+-----+
$
```

- ▶ Creates the key-pair:
 - ◆ ~/.ssh/id_rsa (private key)
 - ◆ ~/.ssh/id_rsa.pub (public key)

Copying the public key to the server

- ▶ Example public key:

```
ssh-rsa AAAAB3NX[. . . ]hpoR3/PLlXgGcZS4oR user@my.hostname.net
```

- ▶ On the server, **~user/.ssh/authorized_keys** contains the list of public keys authorized to connect to the user account
- ▶ The key can be copied manually there
- ▶ Or use `ssh-copy-id` to automatically copy the key:
`client$ ssh-copy-id user@server`
- ▶ Sometimes the public key needs to be provided using a web interface (e.g. on GitHub, FusionForge, Redmine, etc.)

Remembering the passphrase

- ▶ If the private key is not protected with a passphrase, the connection is established immediately:

```
*** login@laptop:~$ ssh rlogin@rhost [ENTER]
*** rlogin@rhost:~$
```

- ▶ Otherwise, ssh asks for the passphrase:

```
*** login@laptop:~$ ssh rlogin@rhost [ENTER]
Enter passphrase for key '/home/login/id_rsa': [passphrase+ENTER]
*** rlogin@rhost:~$
```

- ▶ An **SSH agent** can be used to remember the passphrase
 - ◆ Most desktop environments act as SSH agents automatically
 - ◆ One can be started with `ssh-agent` if needed
 - ◆ Add keys manually with `ssh-add`

Checking the server identity: known_hosts

- ▶ Goal: detect hijacked servers
What if someone replaced the server to steal passwords?
- ▶ When you connect to a server for the first time, ssh stores the server's public key in `~/.ssh/known_hosts`

```
*** login@laptop:~$ ssh rlogin@server [ENTER]
```

```
The authenticity of host 'server (10.1.6.2)' can't be established.
```

```
RSA key fingerprint is
```

```
94:48:62:18:4b:37:d2:96:67:c9:7f:2f:af:2e:54:a5.
```

```
Are you sure you want to continue connecting (yes/no)? yes [ENTER]
```

```
Warning: Permanently added 'server,10.1.6.2'(RSA) to the list of  
known hosts.
```

```
rlogin@server's password:
```

Checking the server identity: known_hosts (2)

- ▶ During each following connection, ssh ensures that the key still matches, and warns the user otherwise

```
*** login@laptop:~$ ssh rlogin@server [ENTER]
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
e3:94:03:90:5d:81:ed:bb:d5:d2:f2:de:ba:31:18:d8.
Please contact your system administrator.
Add correct host key in /home/login/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/login/.ssh/known_hosts:12
RSA host key for server has changed and you have requested strict checking.
Host key verification failed.
*** login@laptop:~$
```

- ▶ Remove a truly outdated key can be removed with
ssh-keygen -R server

Configuring SSH

- ▶ SSH gets configuration data from:
 - 1 command-line options (`-o ...`)
 - 2 the user's configuration file: `~/.ssh/config`
 - 3 the system-wide configuration file: `/etc/ssh/ssh_config`
- ▶ Options are documented in the `ssh_config(5)` man page
- ▶ `~/.ssh/config` contains a list of hosts (with wildcards)
- ▶ For each parameter, the first obtained value is used
 - ◆ Host-specific declarations are given near the beginning
 - ◆ General defaults at the end

Example: ~/.ssh/config

```
Host mail.acme.com
    User root
```

```
Host foo # alias/shortcut. 'ssh foo' works
    Hostname very-long-hostname.acme.net
    Port 2222
```

```
Host *.acme.com
    User jdoe
    Compression yes # default is no
    PasswordAuthentication no # only use public key
    ServerAliveInterval 60 # keep-alives for bad firewall
```

```
Host *
    User john
```

- Note: bash-completion can auto-complete using ssh_config hosts

Plan

- 1 SSH basics
 - SSH 101
 - Public-key authentication
 - Checking the server's identity
 - Configuring SSH

- 2 Advanced usage
 - SSH as a communication layer for applications
 - Access remote filesystems over SSH: sshfs
 - SSH tunnels, X11 forwarding, and SOCKS proxy
 - Jumping through hosts with ProxyCommand
 - Triggering remote command execution securely
 - Escape sequences

- 3 Conclusions

SSH as a communication layer for applications

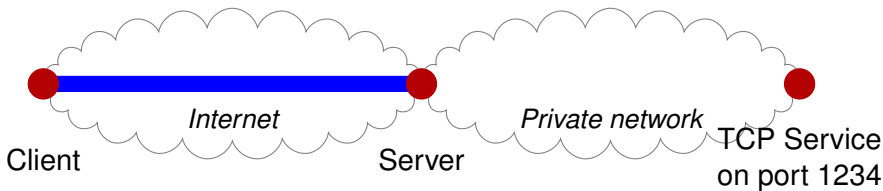
- ▶ Several applications use SSH as their communication layer
 - ◆ Sometimes also authentication layer
- ▶ scp, sftp, rsync (data transfer)
- ▶ unison (synchronization)
- ▶ **Subversion**: `svn checkout svn+ssh://user@rhost/path/to/repo`
- ▶ **Git**: `git clone ssh://git@github.com/path-to/repository.git`
Or: `git clone git@github.com:path-to/repository.git`

Access remote filesystems over SSH: sshfs

- ▶ sshfs: FUSE-based solution to access remote machines
- ▶ Ideal for remote file editing with a GUI, copying small amounts of data, etc.
- ▶ Mount a remote directory:
`sshfs root@server:/etc /tmp/local-mountpoint`
`Unmount: fusermount -u /tmp/local-mountpoint`
- ▶ Combine with afuse to auto-mount any machine:
`afuse -o mount_template="sshfs %r:/ %m" -o \`
`unmount_template="fusermount -u -z %m" ~/.sshfs/`
`~> cd ~/.sshfs/rhost/etc/ssh`

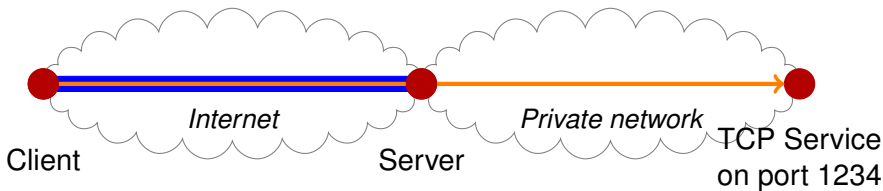
SSH tunnels with -L and -R

- ▶ Goal: transport traffic through a secure connection
 - ◆ Work-around network filtering (firewalls)
 - ◆ Avoid sending unencrypted data on the Internet
 - ◆ But only works for TCP connections
- ▶ **-L**: access a remote service behind a firewall (Intranet server)
 - ◆ `ssh -L 12345:service:1234 server`
 - ◆ Still on *Client*: `telnet localhost 12345`
 - ◆ *Server* establishes a TCP connection to *Service*, port 1234
 - ◆ The traffic is tunnelled inside the SSH connection to *Server*



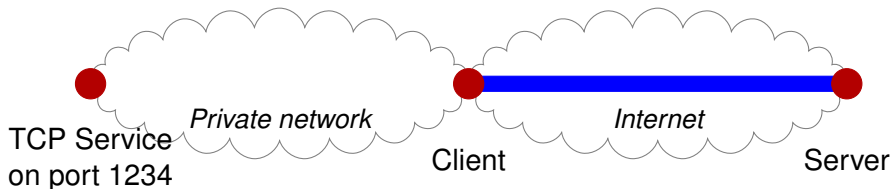
SSH tunnels with -L and -R

- ▶ Goal: transport traffic through a secure connection
 - ◆ Work-around network filtering (firewalls)
 - ◆ Avoid sending unencrypted data on the Internet
 - ◆ But only works for TCP connections
- ▶ **-L**: access a remote service behind a firewall (Intranet server)
 - ◆ `ssh -L 12345:service:1234 server`
 - ◆ Still on *Client*: `telnet localhost 12345`
 - ◆ *Server* establishes a TCP connection to *Service*, port 1234
 - ◆ The traffic is tunnelled inside the SSH connection to *Server*



SSH tunnels with -L and -R

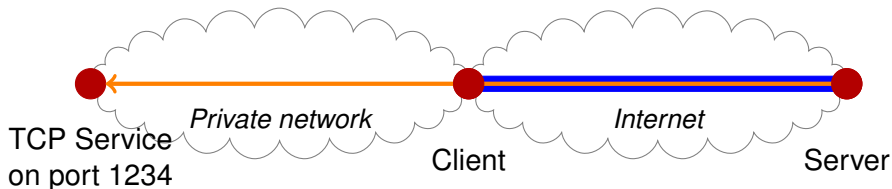
- ▶ **-R**: provide remote access to a local private service
 - ◆ `ssh -R 12345:service:1234 server`
 - ◆ On *Server*: `telnet localhost 12345`
 - ◆ *Client* establishes a TCP connection to *Service*, port 1234
 - ◆ The traffic is tunnelled inside the SSH connection to *Client*



- ▶ Note: SSH tunnels don't work very well for HTTP, because IP+port are not enough to identify a website (`Host: HTTP header`)

SSH tunnels with -L and -R

- ▶ **-R**: provide remote access to a local private service
 - ◆ `ssh -R 12345:service:1234 server`
 - ◆ On *Server*: `telnet localhost 12345`
 - ◆ *Client* establishes a TCP connection to *Service*, port 1234
 - ◆ The traffic is tunnelled inside the SSH connection to *Client*



- ▶ Note: SSH tunnels don't work very well for HTTP, because IP+port are not enough to identify a website (`Host: HTTP header`)

X11 forwarding with -X: GUI apps over SSH

- ▶ Run a graphical application on a remote machine, display locally
- ▶ Similar to VNC, but on a per-application basis
- ▶ `ssh -X server`
- ▶ `$DISPLAY` will be set by SSH on the server:

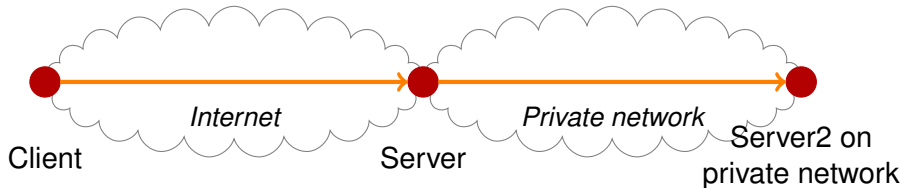
```
$ echo $DISPLAY  
localhost:10.0
```
- ▶ Then start GUI applications on server (e.g. `xeyes`)
- ▶ Troubleshooting:
 - ◆ `xauth` must be installed on the remote machine
 - ◆ The local Xorg server must allow TCP connections
 - ★ `pgrep -a Xorg ~ -nolisten` must not be included
 - ★ Can be configured in your login manager
 - ◆ Does not work very well over slow or high-latency connections

SOCKS proxy with -D

- ▶ SOCKS: protocol to proxy TCP connections via a remote machine
- ▶ SSH can act as a SOCKS server: `ssh -D 1080 server`
- ▶ Use case similar to tunnelling with -L, but more flexible
 - ◆ Set up the proxy once, use for multiple connections
- ▶ Usage:
 - ◆ Manual: configure applications to use the SOCKS proxy
 - ◆ Transparent: use `tsocks` to re-route connections via SOCKS

```
$ cat /etc/tsocks.conf
server = 127.0.0.1
server_type = 5
server_port = 1080 # then start ssh with -D 1080
$ tsocks pidgin # tunnel application through socks
```

Jumping through hosts with ProxyCommand



- ▶ Problem: to connect to Server2, you need to connect to Server
 - ◆ Can you do that in a single step? (required for data transfer, tunnels, X11 forwarding)
- ▶ Combines two SSH features:
 - ◆ ProxyCommand option: command used to connect to host; connection available on standard input & output
 - ◆ `ssh -W host:port` \leadsto establish a TCP connection, provide it on standard input & output (suitable for ProxyCommand)

Jumping through hosts with ProxyCommand (2)

- ▶ Example configuration:

```
Host server2 # ssh server2 works
  ProxyCommand ssh -W server2:22 server
```

- ▶ Also works with wildcards

```
Host *.priv # ssh host1.priv works
  ProxyCommand ssh -W $(basename %h .priv):%p server
```

- ▶ -W only available since OpenSSH 5.4 (circa 2010), but the same can be achieved with netcat:

```
Host *.priv
  ProxyCommand ssh serv nc -q 0 $(basename %h .priv) %p
```

- ▶ Similar solution to connect via a proxy:
 - ◆ SOCKS: `connect-proxy -4 -S myproxy:1080 rhost 22`
 - ◆ HTTP (with CONNECT): `corkscrew myproxy 1080 rhost 22`
 - ◆ When CONNECT requests are forbidden, set up `httptunnel` on a remote server, and use `htc` and `hts`

Triggering remote command execution securely

- ▶ Goal: notify Server2 that something finished on Server1
 - ◆ But Server1 must not have full shell access on Server2
- ▶ Method: limit to a single command in `authorized_keys`
 - ◆ Also known as SSH triggers
- ▶ Example `authorized_keys` on Server2:

```
from="server1.acme.com",command="tar czf - /home",no-pty,  
no-port-forwarding ssh-rsa AAAA[...]oR user@my.host.net
```

Escape sequences

- ▶ Goal: interact with an already established SSH connection
 - ◆ Add tunnels or SOCKS proxy, kill unresponsive connection
- ▶ Escape sequences start with '~', at the beginning of a line
 - ◆ So press [enter], then ~, then e.g. '?'
- ▶ Main sequences (others documented in `ssh(1)`):
 - ◆ ~. – disconnect (for unresponsive connections)
 - ◆ ~? – show the list of escape sequences
 - ◆ ~C – open SSH command-line. e.g. ~C -D 1080
 - ◆ ~& – logout and background SSH while waiting for forwarded connections or X11 sessions to terminate

Plan

- 1 SSH basics
 - SSH 101
 - Public-key authentication
 - Checking the server's identity
 - Configuring SSH

- 2 Advanced usage
 - SSH as a communication layer for applications
 - Access remote filesystems over SSH: sshfs
 - SSH tunnels, X11 forwarding, and SOCKS proxy
 - Jumping through hosts with ProxyCommand
 - Triggering remote command execution securely
 - Escape sequences

- 3 Conclusions

Conclusions

- ▶ The Swiss-army knife of remote administration
- ▶ Very powerful tool, many useful features
- ▶ Practical session: test everything mentioned in this presentation
- ▶ Other topics not covered in this presentation:
 - ◆ Built-in support for VPN
 - ◆ Other authentication methods (certificates)
 - ◆ Managing long executions on a remote machine with screen or tmux
 - ◆ Mosh, an SSH alternative suited for Wi-Fi, cellular and long distance links