SSH - Secure Shell

Miguel Frade & Francisco Santos

POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Introduction

WHAT IS SSH?

Secure Shell (SSH)

- is a cryptographic network protocol
- was designed as a replacement for unsecured remote shell protocols
  - `telnet`, `rsh`, `rlogin` and `rexec`
- typical applications include remote command-line, login, and remote command execution

WHAT IS SSH?

Secure Shell (SSH)

- is a cryptographic network protocol
- was designed as a replacement for unsecured remote shell protocols
  - `telnet`, `rsh`, `rlogin` and `rexec`
- typical applications include remote command-line, login, and remote command execution
- security services provided by SSH
  - confidentiality, integrity and authentication

WHAT IS SSH?

Secure Shell (SSH)

- is a cryptographic network protocol
- was designed as a replacement for unsecured remote shell protocols
  - `telnet`, `rsh`, `rlogin` and `rexec`
- typical applications include remote command-line, login, and remote command execution
- security services provided by SSH
  - confidentiality, integrity and authentication
- there are many implementations of the SSH protocol
  - WinSCP, Putty, OpenSSH, *etc*

WHAT IS NOT SSH?

Secure Shell (SSH) is **not**:

- a product
- a command line shell, like `bash`
- a way to store encrypted data

INSTALLATION

GNU/Linux systems use `OpenSSH`

- install `openssh-server` and `openssh-client`

Terminal

```
user@linux:~$ sudo apt install openssh-server openssh-client
[sudo] password for user:
Reading package lists... Done
Building dependency tree
Reading state information... Done
...
```

Verify server and client installation

```
Terminal
user@linux:~$ sudo service ssh status
[sudo] password for mfrade:
  ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2020-09-27 10:47:41 WEST; 1h 0min ago
  Process: 10271 ExecReload=/bin/kill -HUP MAINPID (code=exited, status=0/SUCCESS)
  Process: 10258 ExecReload=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
  Process: 1204 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 1247 (sshd)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/ssh.service
           └─1247 /usr/sbin/sshd -D
     ...

user@linux:~$ ssh -V
OpenSSH_8.2p1 Ubuntu-4, OpenSSL 1.1.1f  31 Mar 2020
```

OpenSSH Usage

Remote login

- syntax: `ssh [-l login_name] [login_name@]hostname`
- the `login_name` can be omitted if it's the same in both the client and the server
- examples

```
Terminal
user@linux:~$ ssh -l username someserver.com

user@linux:~$ ssh username@someserver.com

user@linux:~$ ssh 192.168.225.3
```

Secure copy

- syntax: `scp [[user@]host1:]source_file ... [[user@]host2:]destination_file`
- like the `cp` command, but the source and/or destination can be a remote computer
- examples

```
Terminal
user@linux:~$ scp user@example.com:remote_file.txt .

user@linux:~$ scp user@example.com:remote_file.txt local_dest_dir

user@linux:~$ scp user@example.com:remote_file.txt /local_dest_dir

user@linux:~$ scp local_file user@example.com:.
```

Remote file system

- syntax: `sshfs [user@]host:[dir] mount_point`
- allows GUI apps to access remote files in a transparent way
- install

```
Terminal
user@linux:~$ sudo apt install sshfs
```

- mount example

```
Terminal
user@linux:~$ sshfs user@192.168.226.3:remote_dir local_empty_dir
```

- verify the mount

```
Terminal
user@linux:~$ mount
```

- unmount

```
Terminal
user@linux:~$ fusermount -u local_empty_dir
```

Remote command execution

- execute commands on servers from a client
- ideal to perform repetitive tasks
- syntax: `ssh [user@]hostname command`
- example script:

```
#!/bin/sh
for pc in server1 server2 server3 server4
do
    ssh $pc uptime
done
```

Note

These exercises might require the usage of RJ45 cables due to wi-fi restritions

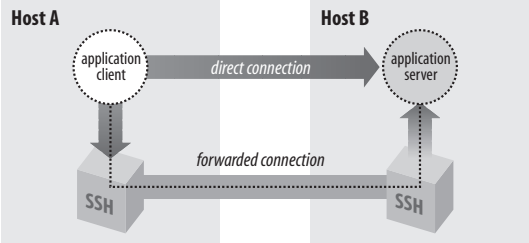
1. Remote login
   - connect to your colleague's PC
   - run some commands such as `ls -l` and `ip a`
   - end the connection with `exit`
2. Remote command execution
   - run the command `df -H` your colleague's PC without creating an interactive section
3. Remote and secure copy of files
   - from your computer copy a local file to your colleague's PC
   - from your computer copy a remote file (from your colleague's PC) to a local folder
4. Remote file system
   - create a new folder `mnt` on your home directory
   - with `sshfs` mount your colleague's home directory into your local `~/mnt` folder
   - verify with the `mount` command
   - use a graphical editor (*e. g.* `kate` or `gedit`) and save a text file into the mounted file system
   - terminate the remote file system created with `sshfs`

Port Forwarding

SSH can encrypt another application's data stream

This is called **port forwarding**

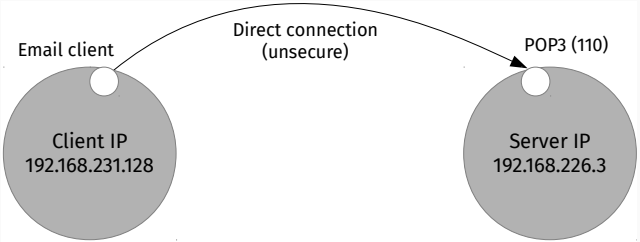
- a secure tunnel is created with SSH
- transparent at the application level
  - the application service must use TCP protocol
  - must run on a single port (FTP uses 2 ports)
  - the application server must run also SSH
  - change the configuration of the client application



Unsecure direct connection of a POP3 client

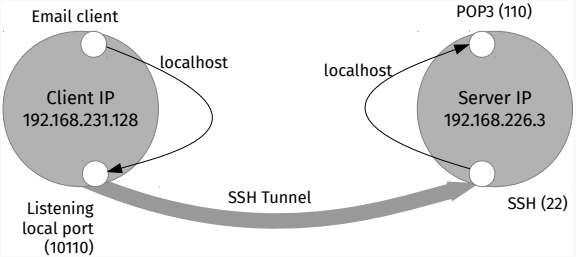
Client configuration

- `IP` = `192.168.226.3`
- `Port` = `110`



Local port forwarding

```
Terminal
user@linux:~$ ssh -N -L10110:localhost:110 user@192.168.226.3
```



Steps

1. create the SSH tunnel
2. change client configuration to
   - `IP` = `localhost`
   - `Port` = `10110`
3. use the client transparently

Explanation of the parameters

```
Terminal
user@linux:~$ ssh -N -L10110:localhost:110 user@192.168.226.3
```

- `-N` non-interactive session
- `-L` setup a local listening port
- `10110` local listening port number
- `localhost:110` where to forward the connection after arriving to the server
- `user@192.168.226.3` SSH authentication on the server

Reverse port forwarding

```
Terminal
user@linux:~$ ssh -N -R10110:localhost:110 client@192.168.231.128
```

- creates the same tunnel, but the SSH command is entered on the server
- `-N` non-interactive session
- `-R` setup a remote listening port (on the client)
- `10110` local listening port number on the client
- `localhost:110` where to forward the connection after arriving to the server
- `client@192.168.231.128` SSH authentication on the **client**

SOCKS proxy protocol port forwarding

```
Terminal
user@linux:~$ ssh -N -D8123 user@192.168.226.3
```

1. create a tunnel to forward SOCKS 4/5 protocol
   - `-N` non-interactive session
   - `-D` setup a local listening port
   - `8123` local listening port number
   - `user@192.168.226.3` SSH authentication on the server

SOCKS proxy protocol port forwarding

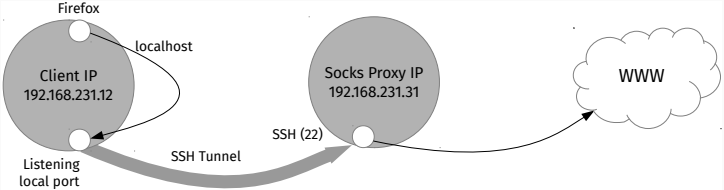
```
Terminal
user@linux:~$ ssh -N -D8123 user@192.168.226.3
```

1. create a tunnel to forward SOCKS 4/5 protocol
   - `-N` non-interactive session
   - `-D` setup a local listening port
   - `8123` local listening port number
   - `user@192.168.226.3` SSH authentication on the server
2. setup Firefox browser to use SOCKS proxy
   - `SOCKS server` = `localhost`
   - `Port` = `8123`

Note

This exercise might require the usage of RJ45 cables due to wi-fi restritions

Setup a SOCKS proxy with SSH and test it



1. create a SSH tunnel for SOCKS with your colleague's PC
2. configure the Firefox browser to use the tunnel
3. visit any webpage
4. kill the SSH tunnel
5. try to visit any webpage (it should give an error)

Authentication

Servers must be authenticated, on the first connection attempt this question shows up:

```
Terminal
user@linux:~$ ssh user@192.168.226.3
The authenticity of host '192.168.226.3' can't be established.
ECDSA key fingerprint is
80:37:7c:fd:40:5f:8f:fa:c8:99:9b:55:db:ac:65:7a.
Are you sure you want to continue connecting (yes/no)?
```

- if you answer `yes` the SSH server key is added to the file `~/.ssh/known_hosts`
- the goal is to prevent man-in-the-middle attacks

Management of servers' keys

- to list all stored keys

```
Terminal
user@linux:~$ ssh-keygen -l -f ~/.ssh/known_hosts
```

- to list a single key

```
Terminal
user@linux:~$ ssh-keygen -H -F name_or_IPaddress
```

- to delete a key

```
Terminal
user@linux:~$ ssh-keygen -R name_or_IPaddress
```

PUBLIC KEY AUTHENTICATION

User authentication can be done through public keys

- it's the most secure way of authentication
- it'll stop bots from bruteforcing on users' passwords

PUBLIC KEY AUTHENTICATION

User authentication can be done through public keys

- it's the most secure way of authentication
- it'll stop bots from bruteforcing on users' passwords

1. generate a key-pair (replace `ed25519` with `rsa` if you have compatibility problems)

```
Terminal
user@linux:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_ed25519.
Your public key has been saved in ~/.ssh/id_ed25519.pub.
...
```

2. copy public key to the server

```
Terminal
user@linux:~$ ssh-copy-id user@192.168.226.3
```

Load the private key to memory

- this step avoids the constant request to enter the `passphrase`
- must be repeated each time the computer reboots

Terminal

```
user@linux:~$ ssh-add
Enter passphrase for id_ed25519:
Identity added: /home/user/.ssh/id_ed25519 (user@linux)
```

Load the private key to memory

- this step avoids the constant request to enter the `passphrase`
- must be repeated each time the computer reboots

Terminal

```
user@linux:~$ ssh-add
Enter passphrase for id_ed25519:
Identity added: /home/user/.ssh/id_ed25519 (user@linux)
```

Login with OpenSSH keys from Windows computers

The Putty Key Generator can convert OpenSSH keys and then you can configure them on the Putty SSH client

Note

This exercise might require the usage of RJ45 cables due to wi-fi restritions

1. setup public key authentication with your colleague's PC and add the private key to your computer memory
2. test the configuration by doing a remote login and exit
3. ask your colleague to edit the file `~/.ssh/authorized_keys` and delete the line that corresponds to your public key
4. test your SSH connection again
   - did you notice any difference?

Server Configuration

Recommended server configurations

- edit the file `/etc/ssh/sshd_config` and check these parameters

```
# Make shure this option is set to "no"
PermitRootLogin no

# To disable password authentication, change to "no" here!
PasswordAuthentication no
# Then enable the public key authentication
PubkeyAuthentication yes

# To limit the users who can login through SSH
AllowUsers user1 user2 user3
```

For the configuration changes to take effect

```
Terminal
user@linux:~$ sudo service ssh reload
```

If the previous command doesn't work for some reason try

```
Terminal
user@linux:~$ sudo service ssh restart
```

Fail2ban

- it's a simple intrusion detection system
- monitors specific logs files for failed login attempts or automated attacks on a server
- when an attempt is discovered `fail2ban` blocks the IP address by adding an iptables rule

Fail2ban

- it's a simple intrusion detection system
- monitors specific logs files for failed login attempts or automated attacks on a server
- when an attempt is discovered `fail2ban` blocks the IP address by adding an iptables rule
- installation

```
Terminal
user@linux:~$ sudo apt install fail2ban
```

- start and enable the service

```
Terminal
user@linux:~$ sudo systemctl start fail2ban
user@linux:~$ sudo systemctl enable fail2ban
```

Configure Fail2ban

- edit the file `/etc/fail2ban/jail.local` and type the following:

```
[sshd]
enabled = true
port = 22
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
# "bantime" is the number of seconds that a host is banned.
bantime  = 600
```

- then restrat the service

```
Terminal
user@linux:~$ sudo systemctl restart fail2ban
```

Check Fail2ban status

Terminal

```
user@linux:~$ sudo fail2ban-client status ssh
[sudo] password for user:
Status for the jail: ssh
|- filter
|   |- File list:        /var/log/auth.log
|   |- Currently failed: 3
|   `- Total failed:     39124
`- action
    |- Currently banned: 0
    |   `- IP list:
    `- Total banned:     426
```

Note

This exercise might require the usage of RJ45 cables due to wi-fi restritions

1. configure `fail2ban` service
2. configure `sshd` to allow password authentication
3. connect to your colleague's PC with a wrong password 4 times
4. connect to your colleague's PC with the correct password
   - where you blocked by `fail2ban`?
   - if so check the new `iptables` rules

```
Terminal
user@linux:~$ sudo iptables -L
```

   - then try to connect again after running this command (replace IP_ADDRESS with the real IP):

```
Terminal
user@linux:~$ sudo fail2ban-client set sshd unbanip IP_ADDRESS
```

Questions?

BIBLIOGRAPHY

- Daniel J. Barret e Richard E. Silverman, “SSH, The Secure Shell – the definitive guide”, Fev. 2005, O’Reilly
- these `man` pages:

```
Terminal
user@linux:~$ man ssh
user@linux:~$ man scp
user@linux:~$ man sshfs
user@linux:~$ man ssh-keygen
user@linux:~$ man ssh-add
user@linux:~$ man sshd
user@linux:~$ man fail2ban
user@linux:~$ man fail2ban-client
```