

# **Exploring OpenSSH: Hands-On Workshop for Beginners**

William Robinet (Conostix S.A.)

**Confidence 2024 Kraków**

**Exploring OpenSSH: Hands-On Workshop for Beginners**

William Robinet (Conostix S.A.) - 2024-05-27

# About me

- Introduced to Open Source & Free Software around the end of the 90's
- CompSci studies, work in IT at Conostix S.A. - AS197692
- ssldump improvements (build system, bug fixes, JSON output, IPv6 & ja3(s), ...)
- asn1template: painless ASN.1 editing
- 🎸 🏃 🦊 🔭 📶

# Before we begin

## Workshop resources

Workshop repository: <https://github.com/wllm-rbnt/confidence-2024-openssh-workshop/>

- Repository -> <https://tinyurl.com/38ktw8rs>
- HTML -> <https://tinyurl.com/2a35j83v>
- PDF -> <https://tinyurl.com/yht2nmkk>

Slides are written in Markdown

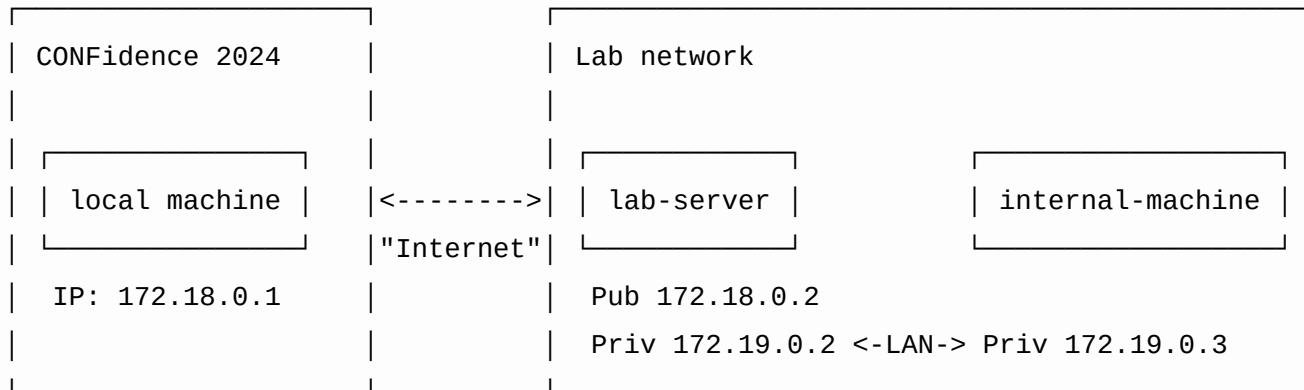
Get the *PDF/HTML* or use *patat* to render the presentation

Go to release page <https://github.com/jaspervdj/patat/releases>

or

```
$ wget https://github.com/jaspervdj/patat/releases/download/v0.12.0.0/patat-v0.12.0.0-linux-x86_64.tar.gz
$ tar xzf patat-v0.12.0.0-linux-x86_64.tar.gz patat-v0.12.0.0-linux-x86_64/patat
$ patat-v0.12.0.0-linux-x86_64/patat confidence-2024-openssh-workshop.md
```

# Labs Network Layout



- *local machine* is your personal laptop or VM. It is located “somewhere on the Internet” It is able to reach *lab-server* on TCP port 22 (on 172.18.0.2)
- *Lab network* is a remote private LAN (172.19.0.0/16 in this case)
- On this remote LAN, *lab-server* is privately known as 172.19.0.2.
- *lab-server* is connected to another machine named *internal-machine* (172.19.0.3)

# Uxnames and Passwords

2 users exist on each VM: *root* and *user*.

Passwords are the same as usernames. *user* has sudo access on each machine.

## Shell commands

Shell commands are prefixed by a prompt designating the machine on which the command shall be run:

```
(local)$ <local command>
```

```
(lab-server)$ <remote command on lab-server>
```

```
(internal-machine)$ <remote commandi on internal-machine>
```

## IP addresses

- IP addresses are dynamically allocated when you execute `start_containers.sh`
- 3 IP addresses will appear during this workshop
  - lab-server\_pub
  - lab-server\_priv
  - internal-machine\_priv

# Labs Containers

- 2 containers will be used during this workshop, one for *lan-server* and a second for *internal-machine*

- Start containers with:

```
(local)$ cd docker  
(local)$ ./start_containers.sh
```

- Stop containers with:

```
(local)$ cd docker  
(local)$ ./stop_containers.sh
```

- Cleanup the whole docker setup: **WARNING this will remove all containers, images and networks from your local setup**

```
(local)$ cd docker  
(local)$ ./docker_cleanup.sh  
(local)$ sudo systemctl restart docker
```

# Illustration: Telnet is not secure

- A *telnet* server is listening on *lab-server*, TCP port 23
- Start a traffic capture on TCP port 23 in another terminal:

```
(local)$ sudo apt install wireshark
```

```
(local)$ sudo wireshark
```

or

```
(local)$ sudo apt install tcpdump
```

```
(local)$ sudo tcpdump -n -i any -XXX tcp and port 23
```

- Then, in another shell, run the *telnet* client on your local machine:

```
(local)$ sudo apt install telnet
```

```
(local)$ telnet 172.18.0.2
```

- Login, *user* Password, *user*

## Two main issues:

- Cleartext message exchange: vulnerable to traffic sniffing tcpdump/wireshark on traffic path (firewall, router)
- Insecure authentication: vulnerable to Man-In-The-Middle attack Ettercap (another machine on same LAN), proxy software on an intermediate router/firewall

Same goes for FTP, HTTP, SMTP, ...



# SSH History & Implementations

SSH stands for Secure **S**hell

## Protocol Versions

- SSH-1.0 1995, by Tatu Ylönen, a researcher at Helsinki University of Technology
- SSH-2.0 2006, IETF Standardization RFC 4251-4256
- SSH-1.99 Retro-compatibility pseudo-version
- SSH3 Experimental implementation using HTTP/3 (QUIC)

## Implementations

- OpenSSH on Unices, Client & server for GNU/Linux, \*BSD, MacOS, ...
- Dropbear, Lightweight implementation, for embedded-type Linux (or other Unices) systems
- On mobile: ConnectBot for Android, Termius for Apple iOS
- Terminal & File transfer clients for MS Windows: PuTTY, MobaXterm, WinSCP, FileZilla, ...
- Network Appliances, OpenSSH or custom implementation

# Focus on OpenSSH Tool suite

- Focus on the OpenSSH tool suite, a project started in 1999
- Clients & Server software
- This is the reference opensource version for many OSes
- It is based on modern cryptography algorithms and protocols
- It is widely available out-of-the-box
- It contains a wide range of tools (remote shell, file transfer, key management, ...)
- Automation friendly (Ansible, or custom scripts)
- Main tools
  - *ssh* - Remote terminal access
  - *scp* - File transfer
  - *sftp* - FTP-like file transfer
- Helpers
  - *ssh-keygen* - Public/Private keypair generation
  - *ssh-copy-id* - Key deployment script
  - *ssh-agent* - Key management daemon (equivalent to PuTTY's pageant.exe)
  - *ssh-add* - Key/Agent management tool

# Documentation

Online manual pages

- Listing of Command Line man pages:

```
$ man -k ssh
```

- Listing client's configuration options:

```
$ man ssh_config
```

- Listing server's configuration options (the *openssh-server* package must be installed):

```
$ man sshd_config
```

- CLI help, in your terminal, just type
  - `ssh` for the client
  - `/usr/sbin/sshd --help` for the server
  - `ssh-keygen --help` for the key management tool
  - ...

# First Login (1/2) - Commands, tcpdump & fingerprints

Syntax is: `ssh <username>@<host>`, where can be a hostname or an IP address

Username and password are the same as the one from the telnet example: - Username: *user* / Password: *user*

- Start a traffic capture on TCP port 22 in another terminal, traffic is **encrypted**:

```
(local)$ sudo tcpdump -n -i docker0 -XXX tcp and port 22
```

- Retrieve the server keys fingerprints through a secure channel:

<https://github.com/wllm-rbnt/confidence-2024-openssh-workshop/blob/main/fingerprints.txt>

# First Login (2/2) - Connection & host authentication

Type the following in a local terminal on your machine:

```
(local)$ ssh user@<lab-server_pub>
```

or

```
(local)$ ssh -o VisualHostKey=true user@<lab-server_pub>
```

The authenticity of host '172.18.0.2 (172.18.0.2)' can't be established.

ED25519 key fingerprint is SHA256:HFofTLfh2W/1IR3+g0sXGAcRs4ZnVsWwGKmb0zeMefk.

```
+--[ED25519 256]--+
```

```
|      . +B=*o |
|      o ooBX.o |
|      o oo=0o=. |
|      + o..= o.* |
|      . S .o o o=|
|      o . o.. |
|      = o   |
|      = *   |
|      + oE  |
```

```
+----[SHA256]-----+
```

This key is not known by any other names.

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

- Type *yes* to accept and go on with user authentication, or *no* to refuse and disconnect immediately
- or type the *fingerprint* you received from the secure channel If the fingerprint you entered matches the one that is printed, the system will proceed with user authentication

# Known hosts fingerprint databases

Remote Host Authentication is performed only on first connection

`~/.ssh/known_hosts` is then populated with host reference and corresponding key fingerprint

`/etc/ssh/ssh_known_hosts` can be used as a system-wide database of known hosts

Hosts references can be stored as clear text (IP or hostname) or the corresponding hash (see *HashKnownHosts* option)

## Host keys location on OpenSSH server

```
(lab-server)$ ls -l /etc/ssh/ssh_host\*.pub
-rw----- 1 root root 513 May 23 12:39 /etc/ssh/ssh_host_ecdsa_key
-rw-r--r-- 1 root root 179 May 23 12:39 /etc/ssh/ssh_host_ecdsa_key.pub
-rw----- 1 root root 411 May 23 12:39 /etc/ssh/ssh_host_ed25519_key
-rw-r--r-- 1 root root 99 May 23 12:39 /etc/ssh/ssh_host_ed25519_key.pub
-rw----- 1 root root 2602 May 23 12:39 /etc/ssh/ssh_host_rsa_key
-rw-r--r-- 1 root root 571 May 23 12:39 /etc/ssh/ssh_host_rsa_key.pub
```

## Computing fingerprints of host keys

```
(lab-server)$ for i in $(ls -l /etc/ssh/ssh_host\*.pub); do ssh-keygen -lf $i; done
256 SHA256:gbF30TEqv4ucpI3VFIEjq0dnrji5woxacnPe+N9mFX8 root@460a6cac3a3c (ECDSA)
256 SHA256:/hUA0roJsQzhM4f9qSZxcBLqEYqmoPi03pVX2fQUxrg root@460a6cac3a3c (ED25519)
3072 SHA256:D0gvg+2kFzvrljqi00EZ23tnQN3H/+oB3cqm0VZHWiQ root@460a6cac3a3c (RSA)
```

Note: use `ssh-keygen -lvf <public_key>` to generate the visual ASCII art representation of a key

# Configuration (1/2)

## Configuration files

Client:

- Per-user client configuration: `~/.ssh/config`
- System-wide client configuration: `/etc/ssh/ssh_config`
- System-wide local configuration: `/etc/ssh/ssh_config.d/*`

Server:

- Server configuration: `/etc/ssh/sshd_config`
- Server local configuration: `/etc/ssh/sshd_config.d/*`

## Configuration options

- Client configuration options: `$ man ssh_config`
- Server configuration options: `$ man sshd_config`

# Configuration (2/2) - Per host client configuration

Client configuration options can be specified per host

Example:

Type following in your local `~/.ssh/config`:

```
Host lab-server
    Hostname <lab-server_pub>
    User user
```

Tips: Printing the “would be applied” configuration

The `-G` parameter cause ssh to print the configuration that would be applied for a given connection (without actually connecting)

```
(local)$ ssh -G lab-server
```

The following command should output your username:

```
(local)$ ssh -G lab-server | grep user
user user
```



# Tips

## Increase verbosity

Launch ssh commands with -v parameter in order to increase verbosity, and help with debugging

Example:

```
(local)$ ssh -v user@<lab-server_pub>  
OpenSSH_8.4p1 Debian-5+deb11u2, OpenSSL 1.1.1w 11 Sep 2023  
debug1: Reading configuration data /home/user/.ssh/config  
debug1: Reading configuration data /etc/ssh/ssh_config  
[...]
```

## Escape character

The escape character can be used to pass out-of-band commands to ssh client

- By default ~, must be at beginning of a line
- Repeat char in order to type it ( ~~ )
- Commands:
  - Quitting current session ~.
  - List Forwarded connections ~#

# Public Key Authentication

## Main Authentication Methods

- *Password* authentication
- *Public/Private key* authentication
  - Used for password-less authentication (passphrase may be required to unlock private key)

## Lab

- Generate a new key pair on your local system (with or without a passphrase):

```
(local)$ ssh-keygen -f ~/.ssh/my-ssh-key
```

- Install your public key on the remote server:

```
(local)$ ssh-copy-id -i ~/.ssh/my-ssh-key.pub user@<lab-server_pub>
```

**Note:** `ssh-copy-id` copies the public key from `~/.ssh/authorized_keys` to the remote machine

- Login again with your new key pair:

```
(local)$ ssh -i ~/.ssh/my-ssh-key user@<lab-server_pub>
```

- Reference your key pair in your personal local configuration file (`~/.ssh/config`):

```
Host lab-server Hostname User user IdentityFile ~/.ssh/my-ssh-key
```

# Authentication Agent

The Authentication Agent can hold access to private keys, thus eliminating the need to enter passphrase at each use

Start the agent:

```
(local)$ ssh-agent | tee ssh-agent-env.sh
SSH_AUTH_SOCK=/tmp/ssh-KwTcl7ZieUKD/agent.1193973; export SSH_AUTH_SOCK;
SSH_AGENT_PID=1193974; export SSH_AGENT_PID;
echo Agent pid 1193974;
(local)$ source ssh-agent-env.sh
Agent pid 1193974
```

Load private key to the agent:

```
(local)$ ssh-add ~/.ssh/my-ssh-key
Enter passphrase for /home/user/.ssh/my-ssh-key: *****
Identity added: my-ssh-key (user@local)
```

Connect to remote machine:

```
(local)$ ssh user@<lab-server_pub>
```

Going further, [keychain](#) can be used to manage ssh-agent & keys across logins sessions

# Remote Command Execution

Simple command execution:

```
(local)$ ssh user@<lab-server_pub> hostname
```

With redirection to local file:

```
(local)$ ssh user@<lab-server_pub> hostname > hostname.txt
```

With redirection to remote file:

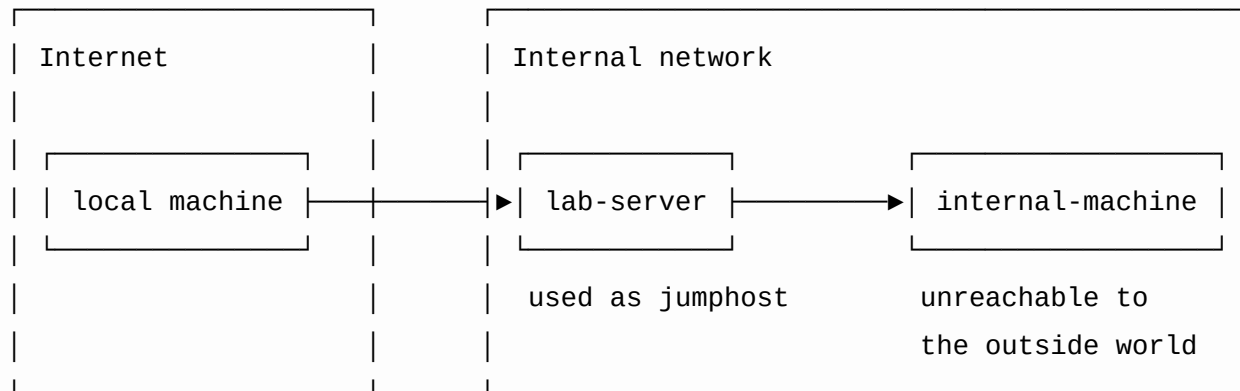
```
(local)$ ssh user@<lab-server_pub> "hostname > hostname.txt"
```

With pipes:

```
(local)$ echo blabla | ssh user@<lab-server_pub> "cat - | tr 'a-z' 'A-Z'"
```

# Jumphost

A Jump Host is a machine used as a relay to reach another, otherwise possibly unreachable, machine. This unreachable machine is named internal-machine



*Lab objective:* Connect to internal-machine from your local machine via SSH with a single command

Setup:

- First, copy your public key to the remote server (lab-server):

```
(local)$ scp .ssh/my-ssh-key.pub user@<lab-server_pub>:
```

- Login to the remote server then copy your public key to the destination machine:

```
(local)$ ssh user@<lab-server_pub> (lab-server)$ ssh-copy-id -f -i my-ssh-key.pub <internal-machine_priv>
```

- Connect to the remote machine with a single command:

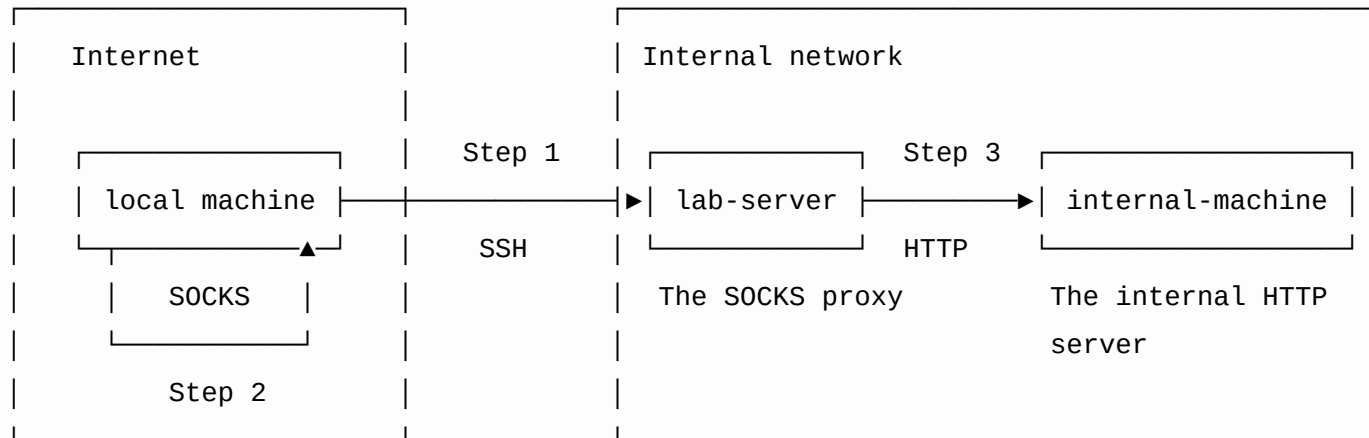
```
(local)$ ssh -J user@<lab-server_pub> user@<internal-machine_priv>
```

**Note:** *internal-machine* host key fingerprints available at <https://github.com/wllm-rbnt/confidence-2024-openssh-workshop/blob/main/fingerprints.txt>

# SOCKS proxy (1/2)

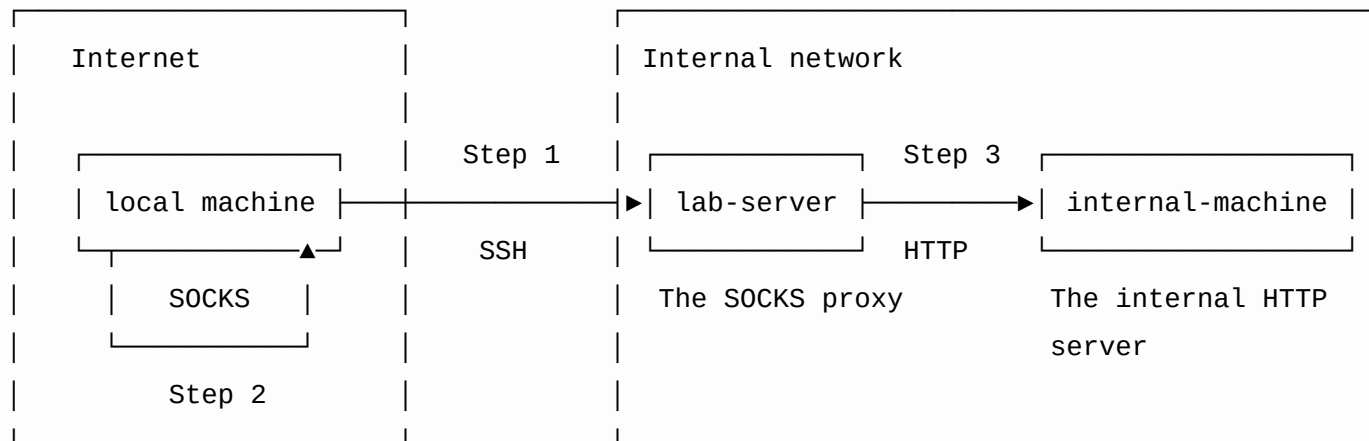
A *SOCKS* server proxies TCP connections to arbitrary IP addresses and ports

With SOCKS 5, DNS queries can be performed by the proxy on behalf of the client



*Lab objective:* Reach the internal HTTP server at `http://secret-intranet` (running on internal-machine) through a SOCKS proxy running on lab-server

## SOCKS proxy (2/2)



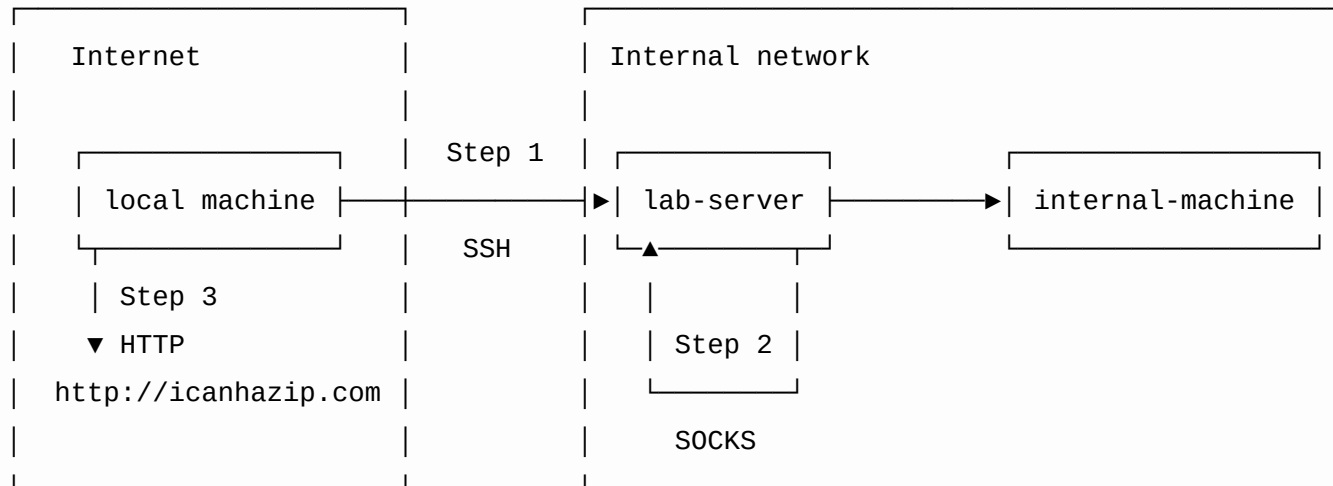
- Start a local SOCKS Proxy: (local)\$ `ssh -D 1234 user@<lab-server_pub>` by establishing an SSH connection to lab-server with parameter -D
- Check, locally, for listening TCP port with (local)\$ `ss -tpln | grep :1234`
- Configure your local browser to use local TCP port 1234 as a SOCKS proxy
- Configure your local browser to send DNS queries though the SOCKS proxy (tick the option in configuration)
- Point your browser to `http://secret-intranet` or Try it with curl:  

```
(local)$ http_proxy=socks5h://127.0.0.1:1234 curl http://secret-intranet
```

Secret intranet server !  
This is lab-server listening on 127.0.0.1 port 80.
- Bonus: look at your local traffic with *tcpdump*, you shouldn't see any DNS exchanges

# Reverse SOCKS proxy

A reverse SOCKS proxy setup allows a remote machine to use your local machine as a SOCKS proxy



*Lab objective:* Reach the external HTTP server at <http://icanhazip.com> from lab-server through a SOCKS proxy running on your local machine

Setup:

- Start a remote SOCKS Proxy: `(local)$ ssh -R 1234 user@<lab-server_pub>` by establishing an SSH connection to lab-server with parameter -R
- Check, on lab-server, for listening TCP port with `(lab-server)$ ss -tpln | grep :1234`
- Point your curl on lab-server to <http://icanhazip.com> though the SOCKS proxy listening on 127.0.0.1<sup>1234</sup>

```
(lab-server_pub)$ http_proxy=socks5h://127.0.0.1:1234 curl http://icanhazip.com  
<Confidence conference Internet access public IP address>
```



# LocalForward

A *LocalForward* creates a locally listening TCP socket that is connected over SSH to a TCP port reachable in the network scope of a remote machine

*Lab objective:* Create and connect local listening TCP socket on port 8888 to TCP port 80 on 127.0.0.1 in the context of *lab-server*

Setup:

- Configure the forwarding while connecting to *lab-server* through SSH with -L parameter: (local)\$ ssh -L 8888:127.0.0.1:80 user@
- -L parameter syntax:

<local\_port>:<remote\_IP>:<remote\_port>

can be extended to

<local\_IP>:<local\_port>:<remote\_IP>:<remote\_port>

- SSH is now listening on TCP port 8888 on your local machine, check with: (local)\$ ss -tpln
- Point your browser to http://127.0.0.1:8888 You should see something like:

Hello world ! This is lab-server listening on 127.0.0.1 port 80.

# RemoteForward

A *RemoteForward* creates a listening TCP socket on a remote machine that is connected over SSH to a TCP port reachable in the network scope of the local machine

*Lab objective:* Create a TCP socket on *lab-server* on port 8123 and connect it to a locally listening netcat on TCP port 1234

Setup:

- Start a listening service on localhost on your local machine on TCP port 1234: `(local)$ nc -l -p 1234 127.0.0.1`
- Check that it's listening with ss: `(local)$ ss -tpln | grep 1234`
- Configure the forwarding on TCP port 8123 while connecting to *lab-server* with -R parameter:

```
ssh -R 8123:127.0.0.1:1234 user@<lab-server_pub>
```

- ssh is now listening on TCP port 8123 on *lab-server*

-R parameter syntax:

`<remote_port>:<local_IP>:<local_port>`

can be extended to

`<remote_IP>:<remote_port>:<local_IP>:<local_port>`

- Check its listening status on lab-server: `(lab-server)$ netstat -tpln | grep 8123`
- Connect to the forwarded service on remote machine on port 8123 with netcat: `(lab-server)$ nc 127.0.0.1 8123`
- Both netcat instances, local & remote, should be able to communicate with each other

**Note:** reverse proxy SOCKS is a special use case of -R

# X11 Forwarding

*Lab objective:* Start a graphical application on *lab-server*, and get the visual feedback locally

*Setup:*

- Connect to lab-server with -X parameter: `(local)$ ssh -X user@<lab-server_pub>`
- Then, start a graphical application on the remote machine:

```
(lab-server)$ xmessage "This is a test !" &!
```

or

```
(lab-server)$ xcalc &!
```

- Check processes with `(lab-server|local)$ ps auxf` on *lab-server* and *local machine*

**Note:**

- On a Linux local client, the XOrg graphical server is used
- On a Windows machine use:
  - VcXsrv: <https://sourceforge.net/~vcxsrv/>
  - or Xming: <https://sourceforge.net/~xming/>

# Connection to Legacy Systems (1/2)

## Host key algorithm mismatch

“Unable to negotiate with 10.11.12.13 port 22: no matching host key type found. Their offer: ssh-rsa”

```
(local)$ ssh -o HostKeyAlgorithms=ssh-rsa <user>@<machine>
```

- Listing known host key algorithms: (local)\$ ssh -Q key

## Wrong key exchange algorithm

“Unable to negotiate with 10.11.12.13 port 22: no matching key exchange method found. Their offer: diffie-hellman-group-exchange-sha1”

```
(local)$ ssh -o KexAlgorithms=diffie-hellman-group1-sha1 <user>@<machine>
```

- Listing known key exchange algorithms: (local)\$ ssh -Q kex

# Connection to Legacy Systems (2/2)

## Wrong cipher

“Unable to negotiate with 10.11.12.13 port 22: no matching cipher found. Their offer: aes128-cbc,3des-cbc,aes192-cbc,aes256-cbc”

```
(local)$ ssh -o Ciphers=aes256-cbc <user>@<machine>
```

- Listing known ciphers: (local)\$ ssh -Q cipher

## Wrong public key signature algorithm

“debug1: send\_pubkey\_test: no mutual signature algorithm” (with ssh -v)

```
$ ssh -o PubkeyAcceptedAlgorithms=ssh-rsa <user>@<machine>
```

- Listing known public key sig algorithm: (local)\$ ssh -Q key-sig or (local)\$ ssh -Q PubkeyAcceptedAlgorithms

# SSH Tarpit

- The legitimate SSH server is running on port 22 on the remote server
- `endlessh`, a simple honeypot, is running on port 2222 on the remote server for demonstration purpose
- Try to connect to port 2222 with `ssh user@<lab-server_pub> -p 2222`
- Check both ports with netcat:

```
(local)$ nc -nv 31.22.124.187 22
(UNKNOWN) [31.22.124.187] 22 (ssh) open
SSH-2.0-OpenSSH_9.2p1 Debian-2
```

```
(local)$ nc -nv 31.22.124.187 2222
(UNKNOWN) [31.22.124.187] 2222 (?) open
XkZ?NK>-h5xs#/OSF
SU6Jv
6%n[;
M5I'R8.W}wgE?"DhADl"jp"$x#4;Z
wT%mJK_l5(Nf]Iw_
$2'ZUmQ2YgdyXnI,
\7_c.f4@bQHcY>N'y
[...]
```

# tmux - terminal multiplexer

tmux can be used to keep interactive shell tasks running while you're disconnected

- Installation: `$ sudo apt install tmux`
- Create a tmux session: `$ tmux`
- List tmux sessions: `$ tmux ls`
- Attach to first session: `$ tmux a`
- Attach to session by index #: `$ tmux a -t 1`
- Commands inside a session:
  - `Ctrl-b d`: detach from session
  - `Ctrl-b c`: create new window
  - `Ctrl-b n` / `Ctrl-b p`: switch to next/previous window
  - `Ctrl-b %` / `Ctrl-b "`: split window vertically/horizontally
  - `Ctrl-b <arrow keys>`: move cursor across window panes
  - `Ctrl-[ + <arrow keys>`: browse current pane backlog, press return to quit
- Documentation: `$ man tmux`

# References

- [OpenSSH](#)
- [SSH History \(Wikipedia\)](#)
- [SSH Mastery by Michael W. Lucas](#)
- [SSH Mastery @BSDCAN 2012](#)
- [A Visual Guide to SSH Tunnels](#)
- [SSH Kung Fu](#)
- [The Hacker's Choice SSH Tips & Tricks](#)



---

**Thanks for your attention**