Exploring OpenSSH: Hands-On Workshop for Beginners

William Robinet (Conostix S.A.)

LGK - Esch-sur-Alzette (2025)

Exploring OpenSSH: Hands-On Workshop for Beginners

William Robinet (Conostix S.A.) - 2025-03-18 & 2025-03-25

Matrix Room: *https://t.ly/hPRg5* – GitHub Repository: *https://t.ly/ITiS6*

About me

- Introduced to Open Source & Free Software around the end of the 90’s
- CompSci studies, CTO at Conostix S.A. - AS197692
- Open Source contributions:
  - ssldump improvements (build system, bug fixes, JSON output, IPv6 & ja3(s), …)
  - asn1template: painless ASN.1 editing
  - dispki: One-Shot Disposable PKI
- BSides Luxembourg & LUNOG
- Conference workshops & presentations
- 🎸 🏃 🚵 🔭 ⏚ …
- Contact info
  - GitHub: **https://github.com/wllm-rbnt**
  - Mastodon: **https://infosec.exchange/@wr**

Before we begin 1/2

Workshop resources

Matrix Room:

`https://matrix.to/#/#LGK_2025-OpenSSH_Workshop:matrix.org`

Used to exchange links and commands.

Workshop repository:

https://github.com/wllm-rbnt/lgk-2025-openssh-workshop

```
git clone https://github.com/wllm-rbnt/lgk-2025-openssh-workshop.git
cd lgk-2025-openssh-workshop
```

Shorter URLs:

- Matrix Room -> **https://t.ly/hPRg5**
- Repository -> **https://t.ly/ITiS6**

Before we begin 2/2

Slides are written in Markdown

Get the *PDF*/*HTML* versions or use *patat* to render the presentation in your terminal

Go to release page **https://github.com/jaspervdj/patat/releases** and download version 0.12.0.1

or

```
wget https://github.com/jaspervdj/patat/releases/download/v0.12.0.1/patat-v0.12.0.1-linux-x86_64.tar.gz
tar xzf patat-v0.12.0.1-linux-x86_64.tar.gz patat-v0.12.0.1-linux-x86_64/patat
patat-v0.12.0.1-linux-x86_64/patat lgk-2025-openssh-workshop.patat.md
```

The Markdown version can be converted to PDF & HTML by using the provided *ws_gen* script (*pandoc* & *chromium* must be installed first)

Local Machine Setup (1/2)

Docker Installation

Reference documentation: **https://docs.docker.com/engine/install/**

This will provide `docker compose` v2 command (with a space).

On Debian 12 (bookworm), the following command will provide `docker-compose` v1 command (with a dash).

```
sudo apt install docker.io docker-compose
```

On Ubuntu 24.10, the `docker compose` v2 command can be installed directly:

```
sudo apt install docker.io docker-compose-v2
```

On Rocky Linux 9, install `docker-ce` with `docker compose` v2 commmand:

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
sudo dnf -y install docker-ce docker-ce-cli containerd.io docker-compose-plugin
sudo systemctl --now enable docker
```

Various other tools

We will use `netcat` (`netcat-traditional` on Debian/Ubuntu), `jq`, `curl`, `wireshark` (or `tcpdump`).

Local Machine Setup (2/2)

Docker Installation on MacOS

The Docker containers used in this workshop can be built and run on MacOS.

Install Docker using brew:

```
brew install jq
brew install --cask docker
```

But access can only be made through locally forwarded ports on *localhost* (aka *127.0.0.1* or *::1*).

```
ssh user@127.0.0.1 -p 8022
```

instead of

```
ssh user@172.18.0.2
```

Labs Containers (1/2)

- 2 containers will be used during this workshop, one for *gateway* and a second for *internal*
- Build and start containers with:

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

- Print setup information:

```
(local)$ ./get_info.sh
```

- Stop containers with:

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

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

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

Labs Containers (2/2)

Pre-built versions of the containers (provided via USB drives) can be loaded from the `docker/images/` directory with the following command:

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

Locally built container images, can be exported to files in the `docker/images/` directory with the following command:

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

Images downloaded from local webserver can be verified for integrity with the following command:

```
(local)$ cd docker/images
(local)$ sha256sum -c SHA256SUMS
docker-gateway.image: OK
docker-internal.image: OK
```

Labs Network Layout

```
┌────────────────────┐              ┌────────────────────────────────────────────┐
│ Local Network      │              │ Lab network (containers)                   │
│ ┌───────────────┐  │              │ ┌─────────────┐         ┌───────────────┐  │
│ │     local     │  │<------------>│ │   gateway   │         │   internal    │  │
│ └───────────────┘  │ The_Internet │ └─────────────┘         └───────────────┘  │
│  IP: 172.18.0.1    │              │  Pub 172.18.0.2                            │
│                    │              │  Priv 172.19.0.2 <-LAN-> Priv 172.19.0.3   │
└────────────────────┘              └────────────────────────────────────────────┘
```

(IP addresses may differ from your Docker setup)

Your *local* machine can reach `gateway` server over ‘The Internet’

- *local* machine is your personal laptop or VM. It is located “somewhere on the Internet” It is able to reach `gateway` 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, `gateway` is privately known as 172.19.0.2.
- `gateway` is connected to another machine named **internal** (172.19.0.3)

Usernames and Passwords

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

Passwords are the same as usernames. *user* has **sudo** access on each machine (no password required).

Shell commands

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

```
(local)$ <local command>
(gateway)$ <remote command on gateway machine>
(internal)$ <remote command on internal machine>
```

IP addresses

- IP addresses are configured statically when you execute `start_containers.sh`
- 3 IP addresses will appear during this workshop
  - <gateway_pub>
  - <gateway_priv>
  - <internal_priv>

Illustration: Telnet is not secure

- A *telnet* server is listening on *gateway*, TCP port 23

- Start a traffic capture on TCP port 23 in another terminal:

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

- Start a capture on your main network interface (*eth0*) or *any*
- 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*

- Finally, right-click on the first TCP packet that belongs to this connection (port 23), then *Follow* -> *TCP Stream*

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 **S**ecure **SH**ell

Protocol Versions

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

Implementations

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

Focus on OpenSSH Tool suite (on GNU/Linux)

- 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 **C**ommand **LI**ne 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

Alternate syntax: `ssh ssh://<username>@<host>/`

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 any -XXX tcp and port 22
```

- Retrieve the server keys fingerprints through a secure channel:

  https://github.com/wllm-rbnt/lgk-2025-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@<gateway_pub>
or
(local)$ ssh -o VisualHostKey=true user@<gateway_pub>
The authenticity of host '172.18.0.2 (172.18.0.2)' can't be established.
ED25519 key fingerprint is SHA256:HFofTLfh2W/1IR3+g0sXGAcRs4ZnVsWwGKmbOzeMefk.
+--[ED25519 256]--+
|           . +B=*o|
|          o ooBX.o|
|         o oo=Oo=.|
|        + 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 know 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

```
(gateway)$ 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

```
(gateway)$ for i in $(ls -1 /etc/ssh/ssh_host*pub); do ssh-keygen -lf $i; done
256 SHA256:gbF30TEqv4ucpI3VFIEjq0dnrji5woxacnPe+N9mFX8 root@460a6cac3a3c (ECDSA)
256 SHA256:/hUAOroJsQzhM4f9qSZxcBLqEYqmoPi03pVX2fQUxrg root@460a6cac3a3c (ED25519)
3072 SHA256:D0gvg+2kFzvrLjqi0OEZ23tnQN3H/+oB3cqm0VZHWiQ root@460a6cac3a3c (RSA)
```

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

Configuration (1/2)

Configuration files

Client:

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

Server:

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

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 gateway
    Hostname <gateway_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 gateway
```

The following command should output your username:

```
(local)$ ssh -G gateway | 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@<gateway_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 new line
- Commands:
  - Quit current session ~.
  - List Forwarded connections ~#
  - Decrease the verbosity (LogLevel) ~V
  - Increase the verbosity (LogLevel) ~v
- Repeat ~ char in order to type it (~~)

Public Key Authentication (1/2)

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@<gateway_pub>
```

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

Public Key Authentication (2/2)

- Login again with your new key pair:

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

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

```
Host gateway
    Hostname <gateway_pub>
    User user
    IdentityFile ~/.ssh/my-ssh-key
```

Authentication Agent (1/3)

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
```

Authentication Agent (2/3)

Load private key into 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@<gateway_pub>
```

Authentication Agent (3/3)

Keychain

Going further, keychain can be used to manage ssh-agent & keys across logins sessions.

Install `keychain`:

```
(local)$ apt install keychain
```

Add the following to your `~/.bashrc` or `~/.zshrc`:

```
keychain --dir ~/.keychain --agents "ssh" -q ~/.ssh/my-ssh-key
```

Now, each new shell session will inherit `ssh-agent` environment variables corresponding to the running agent.

Remote Command Execution (1/2)

Simple command execution:

```
(local)$ ssh user@<gateway_pub> hostname
```

With redirection to local file:

```
(local)$ ssh user@<gateway_pub> hostname > hostname.txt
```

Remote Command Execution (2/2)

With redirection to remote file:

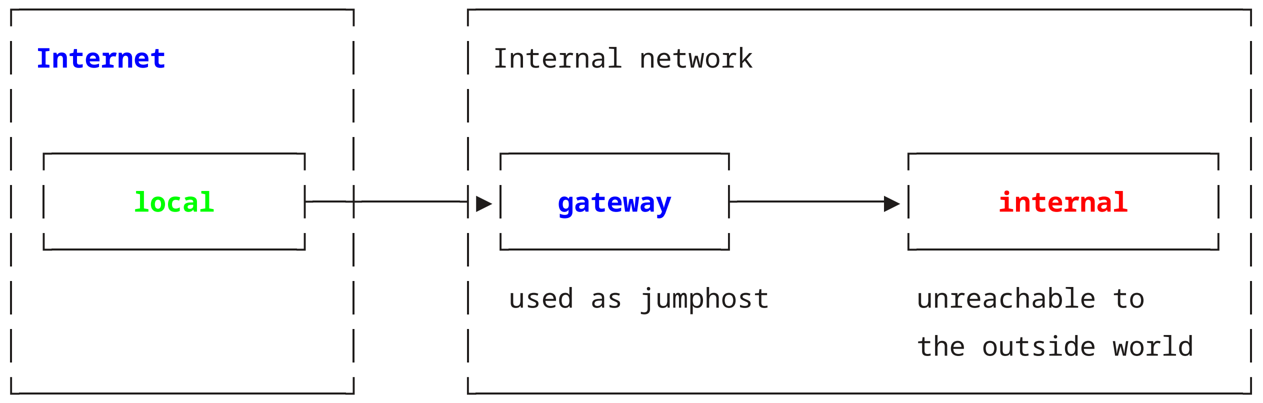
```
(local)$ ssh user@<gateway_pub> "hostname > hostname.txt"
```

With pipes:

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

Jumphost (1/2)

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* from your local machine via SSH with a single command

Jumphost (2/2)

Lab setup:

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

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

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

```
(local)$ ssh user@<gateway_pub>
(gateway)$ ssh-copy-id -f -i my-ssh-key.pub <internal_priv>
```

- Connect to the remote machine with a single command:

```
(local)$ ssh -J user@<gateway_pub> user@<internal_priv>
```

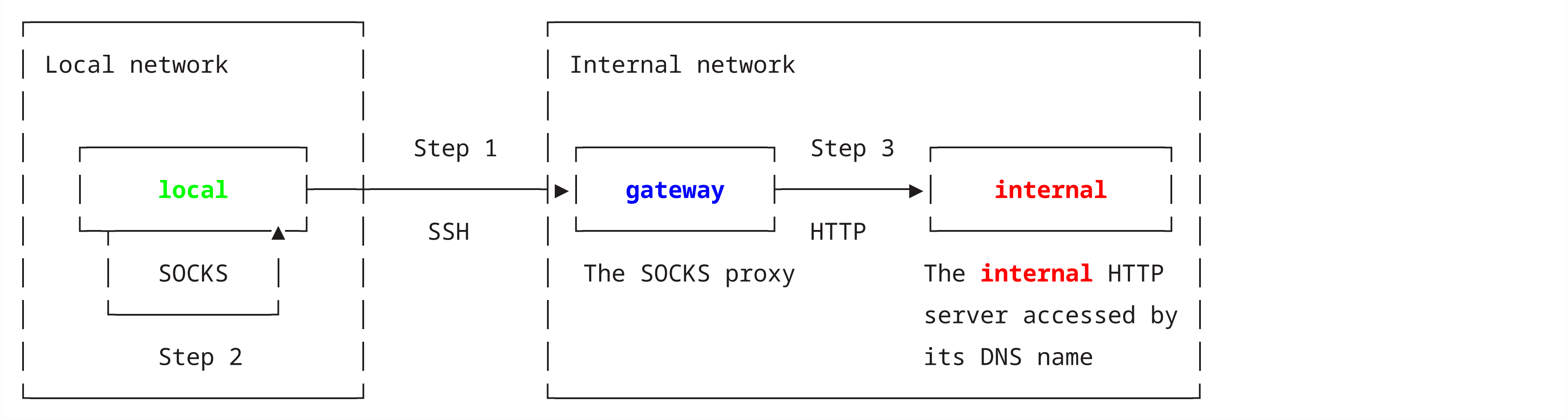
Note: *internal* host key fingerprints available at **https://github.com/wllm-rbnt/lgk-2025-openssh-workshop/blob/main/fingerprints.txt**

SOCKS proxy (1/2)

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

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

Your *local* machine is placed in the same networking context as *gateway*



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

SOCKS proxy (2/2)

- Start a local SOCKS Proxy by establishing an SSH connection to *gateway* with parameter `-D`:

```
(local)$ ssh -D 1234 user@<gateway_pub>
```

- 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:

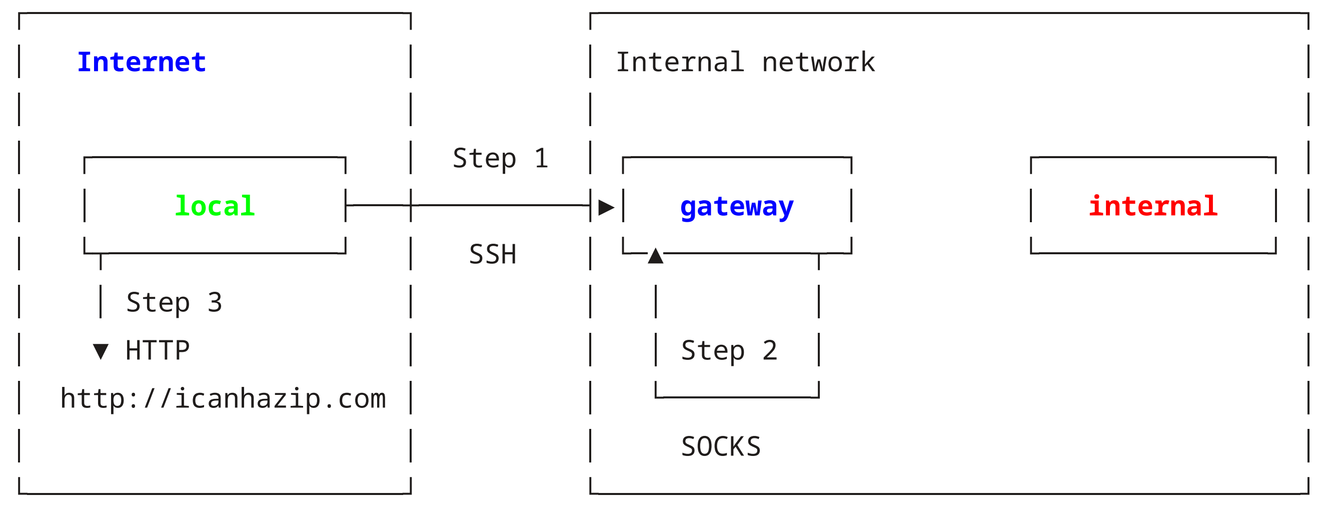
```
(local)$ http_proxy=socks5h://127.0.0.1:1234 curl http://secret-intranet
This is the secret Intranet on internal machine listening on 127.0.0.1 port 80.
```

- Bonus: look at your local traffic with *tcpdump*, you shouldn't see any DNS exchanges

Reverse SOCKS proxy (1/2)

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

gateway is placed in the same networking context as your *local* machine.



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

Reverse SOCKS proxy (2/2)

Setup:

- Start a remote SOCKS Proxy by establishing an SSH connection to *gateway* with parameter -R:

```
(local)$ ssh -R 1234 user@<gateway_pub>
```

- Check, on *gateway*, for listening TCP port with

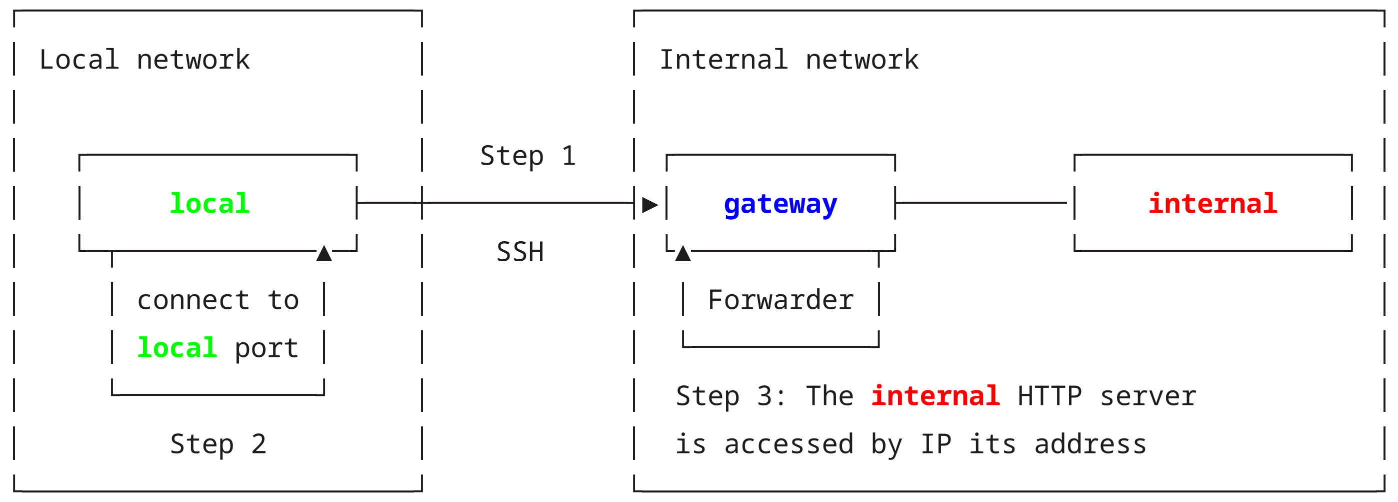
```
(gateway)$ ss -tpln | grep :1234
```

- Point your curl on *gateway* to **http://icanhazip.com** though the SOCKS proxy listening on 127.0.0.1🔢

```
(gateway)$ http_proxy=socks5h://127.0.0.1:1234 curl http://icanhazip.com
<Conference public IP address>
```

LocalForward (1/2)

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 *gateway*

Setup:

- Configure the forwarding while connecting to *gateway* through SSH with -L parameter:

```
(local)$ ssh -L 8888:127.0.0.1:80 user@<gateway_pub>
```

LocalForward (2/2)

`-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:

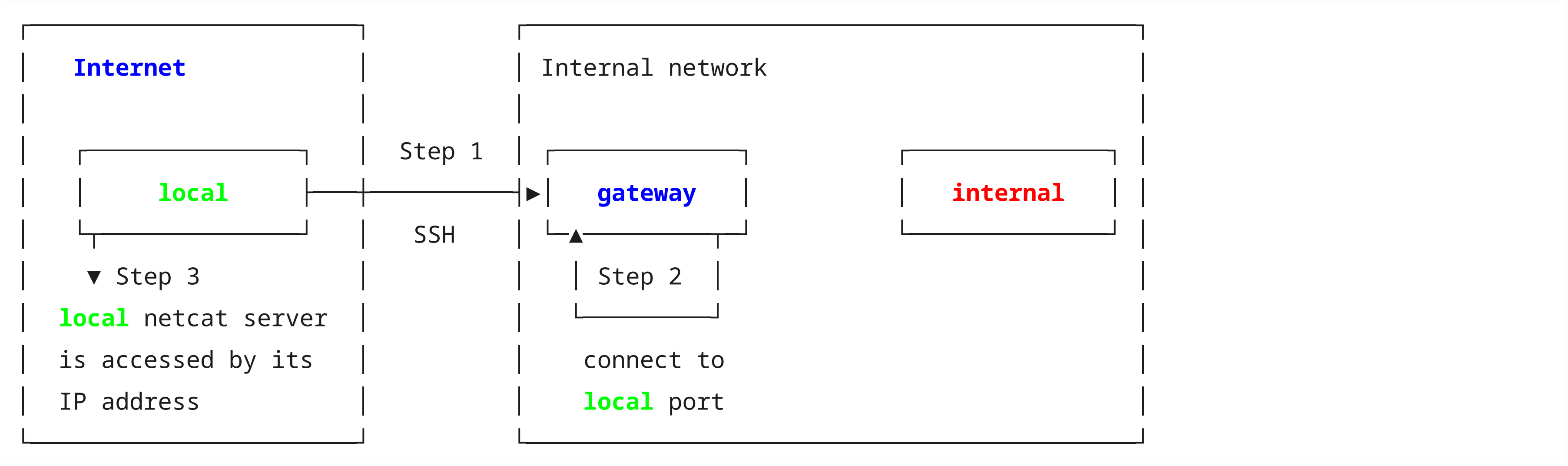
```
(local)$ ss -tpln
```

- Point your browser to **http://127.0.0.1:8888** You should see something like:

  Hello world ! This is gateway listening on 127.0.0.1 port 80.

RemoteForward (1/3)

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 *gateway* on port 8123 and connect it to a locally listening netcat on TCP port 1234

RemoteForward (2/3)

Setup:

- Start a listening service on localhost on your local machine on TCP port 1234:

```
# if you use netcat-traditional
(local)$ nc -l -p 1234 -s 127.0.0.1

# or if you use netcat-openbsd
(local)$ nc -l 127.0.0.1 1234
```

- Check that it’s listening with `ss` (`netstat` replacement on GNU/Linux):

```
(local)$ ss -tpln | grep 1234
```

- Configure the forwarding on TCP port 8123 while connecting to *gateway* with `-R` parameter:

```
(local)$ ssh -R 8123:127.0.0.1:1234 user@<gateway_pub>
```

- `ssh` is now listening on TCP port 8123 on *gateway*

RemoteForward (3/3)

`-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 gateway:

```
(gateway)$ ss -tpln | grep 8123
```

- Connect to the forwarded service on remote machine on port 8123 with netcat:

```
(gateway)$ nc 127.0.0.1 8123
```

- Both netcat instances, local & remote, should be able to communicate with each other

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

X11 Forwarding

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

Setup:

- Connect to *gateway* with `-X` parameter:

```
(local)$ ssh -X user@<gateway_pub>
```

- Then, start a graphical application on the remote machine:

```
(gateway)$ xmessage "This is a test !" &!
```

- Check processes on *gateway* and *local* machine:

```
(gateway|local)$ ps auxf
```

Notes:

- 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 (1/4)

Host key algorithm mismatch

```
(local)$ ssh -p 4022 user@<gateway_pub>
Unable to negotiate with 172.18.0.2 port 4022: no matching host key type found. Their offer: ssh-rsa
```

```
(local)$ ssh -o HostKeyAlgorithms=ssh-rsa -p 4022 user@<gateway_pub>
```

- Listing known host key algorithms:

```
(local)$ ssh -Q key
```

Connection to Legacy Systems (2/4)

Wrong key exchange algorithm

```
(local)$ ssh -p 4023 user@<gateway_pub>
Unable to negotiate with 172.18.0.2 port 4023: no matching key exchange method found. Their offer: diffie-
        hellman-group1-sha1,kex-strict-s-v00@openssh.com
```

```
(local)$ ssh -o KexAlgorithms=diffie-hellman-group1-sha1 -p 4023 user@<gateway_pub>
```

- Listing known key exchange algorithms:

```
(local)$ ssh -Q kex
```

Connection to Legacy Systems (3/4)

Wrong cipher

```
(local)$ ssh -p 4024 user@<gateway_pub>
Unable to negotiate with 172.18.0.2 port 4024: no matching cipher found. Their offer: aes256-cbc
```

```
(local)$ ssh -o Ciphers=aes256-cbc -p 4024 user@<gateway_pub>
```

- Listing known ciphers:

```
(local)$ ssh -Q cipher
```

Connection to Legacy Systems (4/4)

Wrong public key signature algorithm

Disclaimer: **This one is broken with the current Docker containers**

```
(local)$ ssh -p 4025 -i ~/.ssh/mykey user@<gateway_pub>
Received disconnect from 172.18.0.2 port 4025:2: Too many authentication failures
Disconnected from 172.18.0.2 port 4025
```

```
"debug1: send_pubkey_test: no mutual signature algorithm" (with ssh -v)
```

```
(local)$ ssh -o PubkeyAcceptedAlgorithms=ssh-rsa -i ~/.ssh/mykey user@<gateway_pub>
```

- Listing known public key sig algorithms:

```
(local)$ ssh -Q key-sig
```

or

```
(local)$ ssh -Q PubkeyAcceptedAlgorithms
```

SSH Tarpit

- The legitimate SSH server is running on port 22 on *gateway*
- `endlessh`, a simple honeypot, is running on port 2222 on *gateway* for demonstration purpose
- Try to connect to port 2222 with

```
(local)$ ssh user@<gateway_pub> -p 2222
```

- Check both ports with `netcat`:

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

(local)$ nc -nv <gateway_pub> 2222
(UNKNOWN) [<gateway_pub>] 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
- Why port 22 ?

Thanks for your attention !