Pivoting: (This guide assumes that you have made your “root” reverse shell and upgraded it)

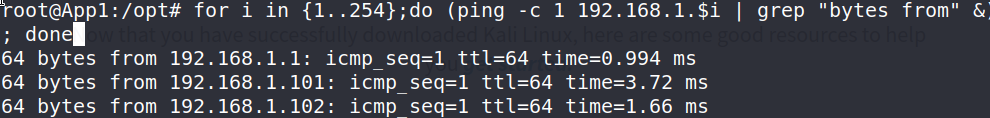
**Step 1. Ping Sweep**

We use a ping sweep to being to interact with local targets to find them. We accomplish with a command that is a bash one-liner that attempts to ping each IP address in the range 192.168.1.1 to 192.168.1.254  


Broken down:

* for i in {1..254}; do: This is a loop that iterates over the numbers from 1 to 254, and in each iteration, it assigns the current number to the variable i.
* (ping -c 1 192.168.1.$i | grep "bytes from" &): This is the command executed in each iteration of the loop.
  + ping -c 1 192.168.1.$i: This pings the IP address formed by appending the current value of i to the base IP address 192.168.1.. The -c 1 option specifies to send only one ping packet.
  + |: This is a pipe, which takes the output of the ping command and passes it as input to the next command.
  + grep "bytes from": This filters the output of the ping command to show only lines that contain the phrase "bytes from".
  + &: This puts the command in the background, allowing the loop to continue without waiting for the ping to complete.
* done: This keyword marks the end of the for loop.

The result should look like this:



What we have done is gotten a list of ip addresses that are active on the target network. We want to save this information, but not on the device we are targeting, so….

**Step 2. Moving valuable information**

Move back to your Kali machine and use “nano ping\_sweep” and this will create a file that you can copy and paste the ping sweep information we just discovered:

A screenshot of a computer

Description automatically generated

We then want to just get the information we need: We do so with Awk”



* Broken down: cat ping\_sweep: This uses the cat command to concatenate and display the content of the file named ping\_sweep.
* |: This is a pipe. It takes the output of the cat command and passes it as input to the next command.
* awk -F ":" '{print $1}': This uses the awk command to process the input with a specified field separator (-F ":"). In this case, the field separator is a colon (:). The command then prints the first field ($1) of each line.

A screenshot of a computer code

Description automatically generated

But want only the exact ip addresses, so we would use



awk -F " " '{print $4}': This awk command processes the input by considering spaces as field separators. It prints the fourth field ($4) of each line, assuming that the content after the IP address is structured in a way that the fourth field is the desired information.

And that leaves us with:

A screen shot of a computer

Description automatically generated

You can use “sort” to rearrange the addresses as you like “sort -u” sorts by words, “sort -V” sorts by numbers

Use sort and then >> to create a file called targets.

A screen shot of a computer code

Description automatically generated

**Step 3. Making Target directories and Files**

We’re going to want to stay organized for every target we want to attack, and the best way is to make directories for each of them, and we’ll use xargs to do this:



Broken down:

* cat {file name}: This uses the cat command to display the contents of the specified file.
* |: This is a pipe, which takes the output of the cat command and passes it as input to the next command.
* xargs mkdir: The xargs command reads the input (which is the content of the file) and executes the mkdir command with each line as an argument. This effectively creates a directory for each line in the file.

A screenshot of a computer

Description automatically generated

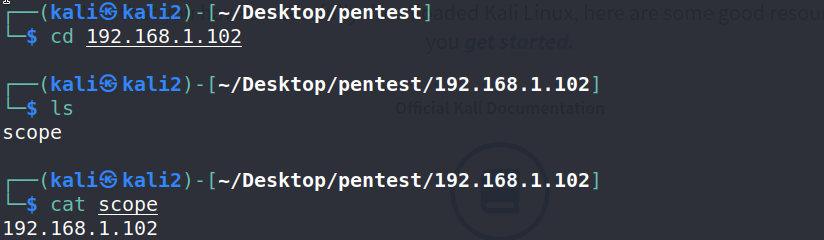
After this we’ll want a scope file for each IP address directory and we can accomplish this by:



Broken down:

* for i in 192.\*: This initializes a loop where the variable i iterates over files or directories in the current directory that match the pattern 192.\*. This typically selects files or directories whose names start with "192."
* do: Marks the beginning of the commands to be executed in each iteration of the loop.
* echo $i > /$i/scope: This command writes the value of i to a file named scope within a directory specified by /$i/. The > operator is used to redirect the output of the echo command to the specified file.
* done: Marks the end of the loop.

And this is the result:



**Step 4. Pivoting and SSH Remote Proxy**

Setting up a remote proxy through ssh is the same as setting up a reverse shell. You need a listener, port-forwarding, and a payload. First we open port 22 which is our SSH:



Feel free to check the status with replacing start w/ status

Next we need our firewall rule on the target network to forward the data to out listening port. Fortunately we have the firewall and familiarity with the rules section to add this:

A screenshot of a computer

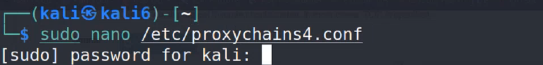
Description automatically generated

The only thing that changes is our destination port range and the redirect target port, make sure you use the right IP address/ports and not mine. The WAN port that it uses is the port we choose, the internal port inside our network will be the SSH -22

That’s your port forwarding.

Finally, the payload, (remember you want your target box to be higher the 7.6 and we go to our root terminal but first we have to configure the proxy config files (you’ll only have to do it once). a proxy chain is a sequence or series of proxy servers that are used to route internet traffic. Each proxy server in the chain acts as an intermediary between your device and the destination on the internet.

We start by entering the config file w/ nano:



First we uncomment dynamic chain and comment strict chain, there by reversing the config:

A screenshot of a computer

Description automatically generated

* **Static Chain:** Fixed and unchanging sequence of proxy servers.
* **Dynamic Chain:** Variable sequence that can adapt based on certain conditions.

You also need to uncomment “quiet mode” (making the sequence less verbose) and comment “proxy DNS” (because we don’t want dns checks that might block us)

A screenshot of a computer screen

Description automatically generated

The last step is to change the proxy list from socks4 to socks 5 and port 9050 to port 9080, this is simply to change it from the default TOR port which is often blocked.

A screenshot of a computer

Description automatically generated

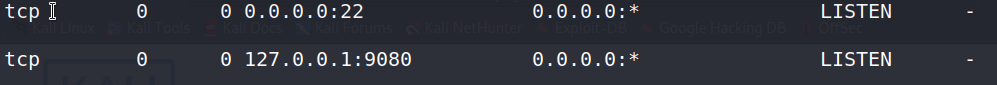
Now that this is done on your machine, you’re free to setup that payload on your target machine app1



Remember to use your port that you set on your fire wall. That being said here’s the break down:

* ssh: Initiates an SSH connection.
* -N: Tells SSH not to execute any commands on the remote host. This is often used when you only want to establish a tunnel and not run any commands on the remote machine.
* -f: Requests the SSH session to go to the background just before command execution.
* -R 127.0.0.1:9080: Sets up a reverse tunnel. Connections to port 9080 on the remote machine (192.168.122.209) will be forwarded through the secure tunnel and redirected to localhost (127.0.0.1) on the local machine.
* kali@192.168.122.209: Specifies the username (kali) and the IP address of the remote machine (192.168.122.209) to connect to.
* -p 55556: Specifies the port number (55556) to connect to on the remote machine. This is an alternative port number to the default SSH port (22).

Last thing to do is to validate the results with a good ol “netstat -natp”, and you should see something like:



Congratulations. You just set up your pivot!

**Step 5. Targeting Specific Data**

Now we return to our Kali machine’s “Desktop/pentest” directory and we type these commands in each IP directory. Now that we have setup our tunnel, we can send from our target to our waiting directories.

To start will need to wrap nmap inside our proxy chains, to do so we start with this command inside one of our target IP address directories:



The break down:

* proxychains: Runs a command through a proxy or proxy chain. It's often used to route network traffic through one or more proxy servers.
* nmap: The network exploration tool used for port scanning and network discovery.
* -iL scope: Specifies a list of target IP addresses to scan. In this case, it's reading the IP addresses from the file named "scope."
* -F: Performs a fast scan. It scans only the most common 100 ports.
* -sT: Specifies the TCP connect scan mode. It attempts to establish a full TCP connection to each scanned port.
* -Pn: Treats all hosts as online, skipping host discovery. This is useful when you already know the hosts are up.
* |: The pipe symbol is used to redirect the output of the nmap command to the next command.
* tee fast\_scan: This command is used to both display the output of the nmap command on the screen and save it to a file named "fast\_scan."

The result should look like this:

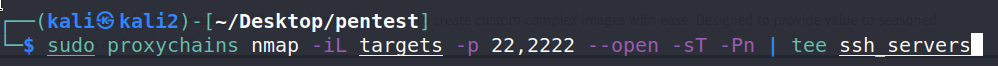
A screen shot of a computer

Description automatically generated

NOTE: Don’t use anything with an IP address with 1.1 or .254, these are likely routers or switches and won’t yield dividends and takes a long time to scan.

Now there’s other scans to do, such as port, full, and service, but I’ll update it later, hopefully when I figure out how to upload this to GitHub.

But one that we should focus on is a scan that will categorize targets, let’s say we want to only scan for IPs that have SSH open. So, we target by “target service” rather than targets alone. Remember we have that “target” directory in our Pentest directory. Let’s use that:



The breakdown:

* proxychains: Executes a command through a proxy or proxy chain. In this context, it's used to route the network traffic through one or more proxy servers.
* nmap: The network exploration tool used for port scanning and network discovery.
* -iL targets: Specifies a list of target IP addresses to scan. It's reading the IP addresses from the file named "targets."
* -p 22,2222: Specifies the ports to scan. In this case, it scans for SSH servers on both port 22 and port 2222.
* --open: Only shows open ports. It filters the output to display only the ports that are open.
* -st: Performs a TCP connect scan. It attempts to establish a full TCP connection to each scanned port.
* -Pn: Treats all hosts as online, skipping host discovery. This is useful when you already know the hosts are up.
* |: The pipe symbol is used to redirect the output of the nmap command to the next command.
* tee ssh\_servers: Saves the output of the nmap command to a file named "ssh\_servers" and displays it on the screen.

So what we have done is scan those IP addresses in “targets” and weeded out the ones that did not have ports 22, 2222 open. Now we organize that data to be more readable to our machine:

A screen shot of a computer

Description automatically generated

And we can further refine and group this by:



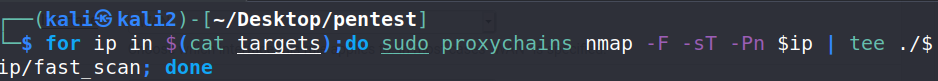
And that leaves us with:

A screen shot of a computer

Description automatically generated

**Step 6: Shortcut for scans and organization:**

This command line short cut will cut down on typing time and populate your IP addresses directories with various scans, before you start, make sure that any 1.1 IP addresses are removed as directories and from your “targets” file. It will make the first fast scan “nmap” go insanely slow.



Here’s the break down:

1. for ip in $(cat targets); do: This part of the command initiates a loop that iterates over each IP address listed in the "targets" file. The $(cat targets) command reads the contents of the "targets" file, which presumably contains a list of IP addresses separated by newlines.
2. sudo proxychains nmap -F -sT -Pn $ip: This is the core command within the loop. It uses nmap for a fast scan (-F) with TCP connect scan (-sT) and disables host discovery (-Pn). The sudo command is used to run nmap with elevated privileges. The IP address being scanned is represented by the $ip variable, which takes each IP address from the "targets" file during each iteration of the loop.
3. | ./$ip/fast\_scan: This part involves piping the output of the nmap command into a script named "fast\_scan" located in a directory with the same name as the IP address. The dot (.) before the script path indicates that the script is in the current directory. The script is likely specific to each IP address and performs additional tasks or analysis based on the output of the nmap scan.

Now we want do this also for an “all, service, and port” scan, so those commands would be:

For an Aggressive (-A) Scan:

A black screen with white text

Description automatically generated

Note you’ll need a “;” before the “done” command for these codes. I put them in like this from chatGPt for ease of reading.

For Port Scan (-p-):

A black background with white text

Description automatically generated

Note this one took a long time, 30 minutes for all 8 or so IP addresses.

If you want to speed it up you can do it for the first 1000 ports:

for ip in $(cat targets); do sudo proxychains nmap -p 1-1000 -Pn $ip | tee ./$ip/port\_scan; done

Or do it in parallel for all the IP’s at once:

cat targets | xargs -n 1 -P 4 -I {} sudo proxychains nmap -p 1-1000 -Pn {} | tee ./{}\_port\_scan

(Note: I haven’t attempted these yet)

For service (-sV):

A black background with white text

Description automatically generated

And that should be it, final note, if you want to hand jam this for each ipaddress, be in that directory, and the command will look like this:

A black background with white text

Description automatically generated