Performing Active Reconnaissance with Nmap, Wireshark, and Scapy

This lab work was done as given in the "**information gathering and vulnerability scanning**" module in **Cisco's ethical hacking** course.

Lab1- Enumeration with Nmap

In this lab, we will complete the following objectives:

- Investigating Nmap
- Performing the Basic Nmap Scans

Nmap Scan Types

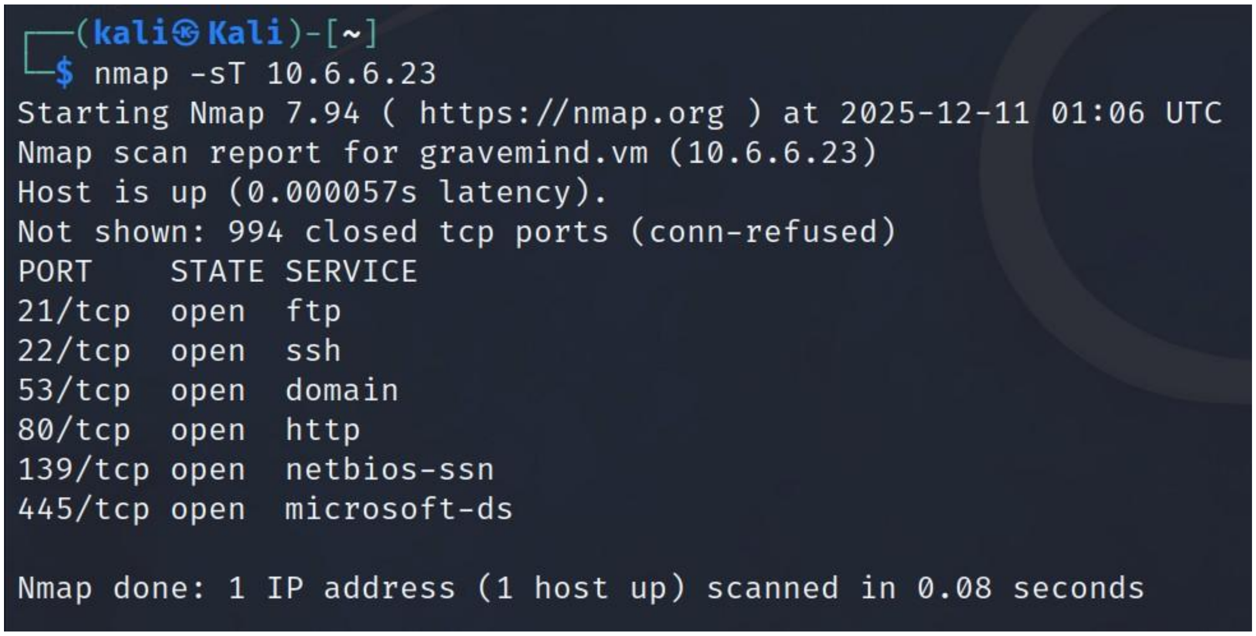
The following are some of the most common Nmap scanning options used for specific scenarios:

- TCP Connect Scan (-sT)
- UDP Scan (-sU)
- TCP FIN Scan (-sF)
- Host Discovery Scan (-sn)
- Timing Options (-T 0-5)

TCP Connect Scan (-sT)

A TCP connect scan actually makes use of the underlying operating system's networking mechanism to establish a full TCP connection with the target device being scanned. It generates more traffic (and thus takes longer to run). This is the default scan type that is used if no scan type is specified with the nmap command.

```
Syntax: #nmap -sT <Ip address>
Example: #nmap -sT 10.6.6.23
```

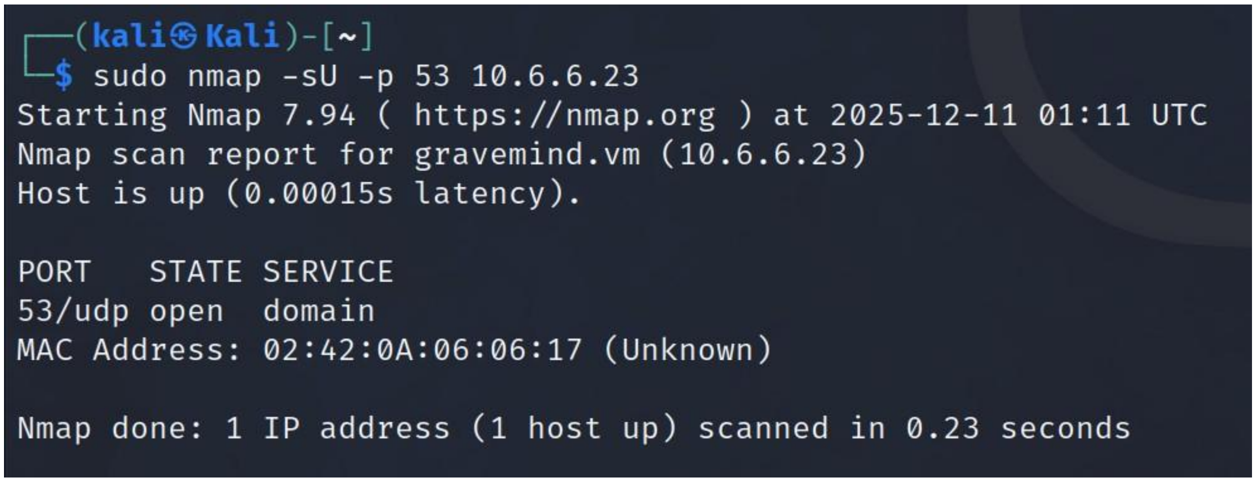


```
┌──(kali㉿Kali)-[~]
└─$ nmap -sT 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:06 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.000057s latency).
Not shown: 994 closed tcp ports (conn-refused)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
53/tcp  open  domain
80/tcp  open  http
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

UDP Scan (-sU)

It is required to scan for UDP ports to enumerate a DNS, SNMP, or DHCP server. These services all use UDP for communication between client and server. While scanning, if it receives an ICMP port unreachable message back from a target, that port is marked as closed. If it gets no response from the target UDP port, Nmap marks the port as open/filtered.

```
Syntax: # sudo nmap -sU -p 53 <Ip address>
Example: # sudo nmap -sU -p 53 10.6.6.23
```

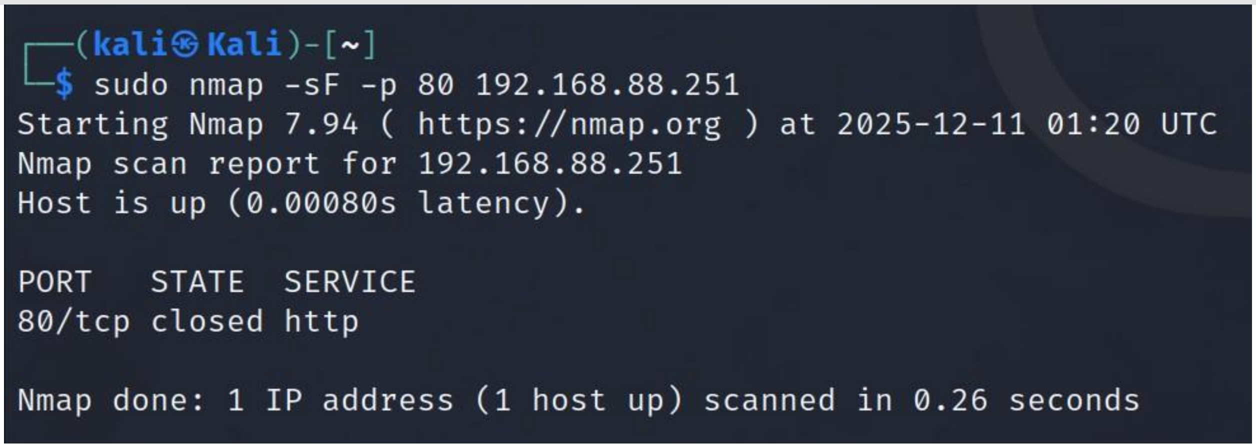


```
┌──(kali㉿Kali)-[~]
└─$ sudo nmap -sU -p 53 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:11 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00015s latency).

PORT   STATE SERVICE
53/udp open  domain
MAC Address: 02:42:0A:06:06:17 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
```

TCP FIN Scan (-sF)

With the TCP FIN scan, a FIN packet is sent to a target port. If the port is actually closed, the target system sends back an RST packet. If no response is received from the target port, it can be considered that the port is open, as the normal behavior would be to ignore the FIN packet.

Syntax: # sudo nmap -sF -p 80 <IP address>
Example: # sudo nmap -sF -p 80 192.168.88.251

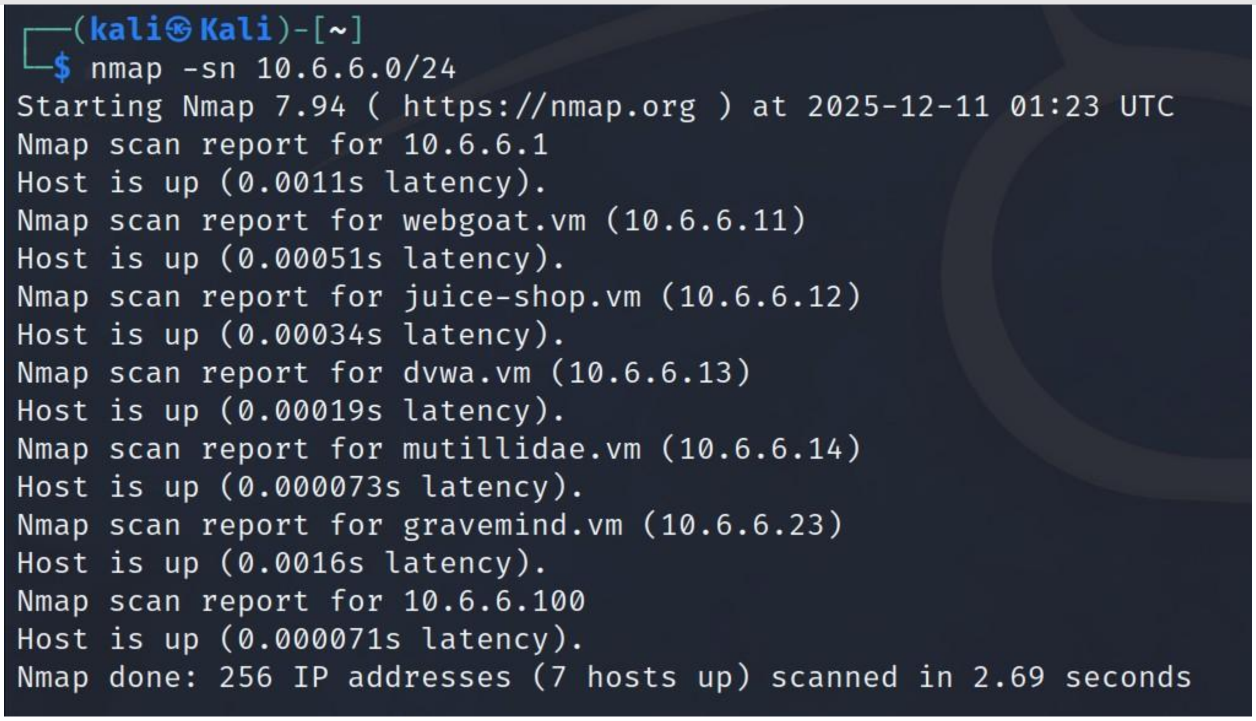


```
┌──(kali㉿Kali)-[~]
└─$ sudo nmap -sF -p 80 192.168.88.251
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:20 UTC
Nmap scan report for 192.168.88.251
Host is up (0.00080s latency).

PORT   STATE  SERVICE
80/tcp closed http

Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
```

Host Discovery Scan (-sn)

A host discovery scan is one of the most common types of scans used to enumerate hosts on a network, as it utilizes various types of ICMP messages to determine whether a host is online and responding on the network.

Syntax: #nmap -sn <Ip range>
Example #nmap -sn 10.6.6.0/24



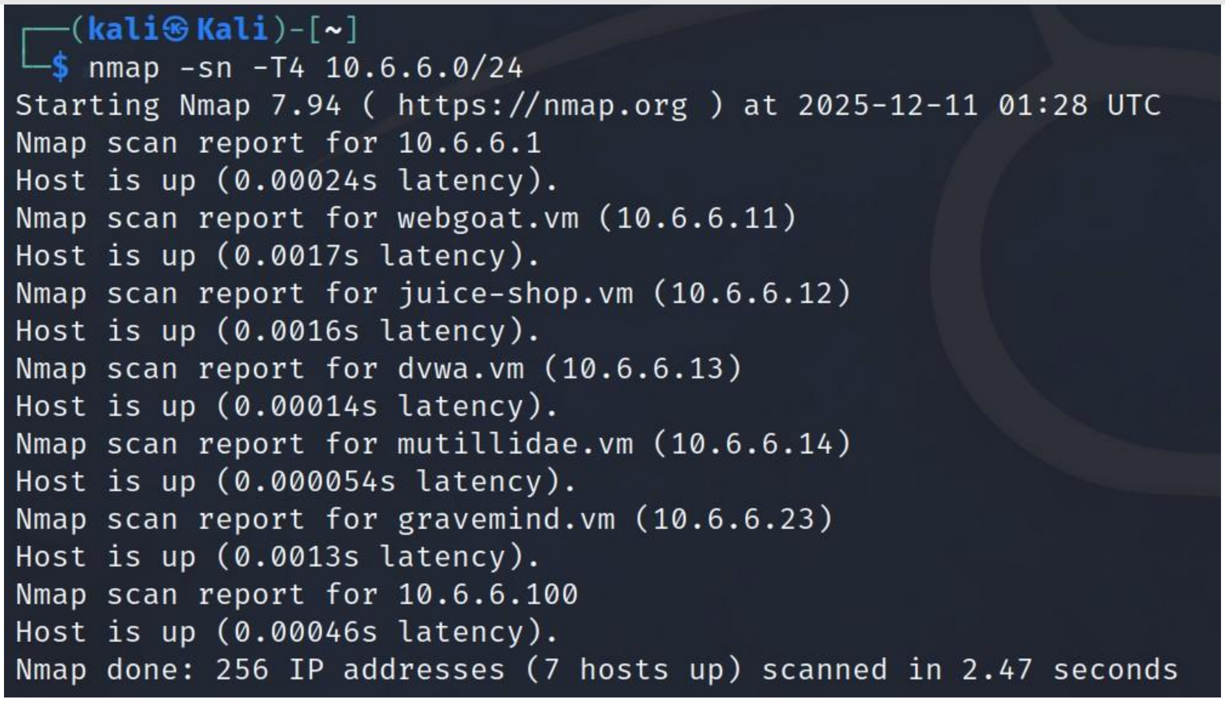
```
┌──(kali㉿Kali)-[~]
└─$ nmap -sn 10.6.6.0/24
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:23 UTC
Nmap scan report for 10.6.6.1
Host is up (0.0011s latency).
Nmap scan report for webgoat.vm (10.6.6.11)
Host is up (0.00051s latency).
Nmap scan report for juice-shop.vm (10.6.6.12)
Host is up (0.00034s latency).
Nmap scan report for dvwa.vm (10.6.6.13)
Host is up (0.00019s latency).
Nmap scan report for mutillidae.vm (10.6.6.14)
Host is up (0.000073s latency).
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.0016s latency).
Nmap scan report for 10.6.6.100
Host is up (0.000071s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 2.69 seconds
```

Timing Options (-T 0-5)

The Nmap scanner provides six timing templates that can be specified with the -T option and the template number (0 through 5) or name. Nmap timing allows setting how aggressive a scan will be, while leaving Nmap to pick the exact timing values.

- -T0 (Paranoid) : Very slow, used for IDS evasion
- -T1 (Sneaky) : Quite slow, used for IDS evasion
- -T2 (Polite) : Slows down to consume less bandwidth, runs about 10 times slower than the default
- -T3 (Normal) : Default, a dynamic timing model based on target responsiveness
- -T4 (Aggressive) : Assumes a fast and reliable network and may overwhelm targets
- -T5 (Insane) : Very aggressive; will likely overwhelm targets or miss open ports

Syntax: nmap -sn -T4 <IP> or <IP range>
Example: #nmap -sn -T4 10.6.6.0/24



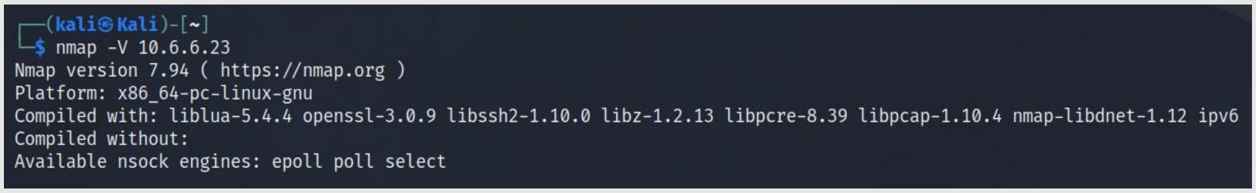
```
┌──(kali㉿Kali)-[~]
└─$ nmap -sn -T4 10.6.6.0/24
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:28 UTC
Nmap scan report for 10.6.6.1
Host is up (0.00024s latency).
Nmap scan report for webgoat.vm (10.6.6.11)
Host is up (0.0017s latency).
Nmap scan report for juice-shop.vm (10.6.6.12)
Host is up (0.0016s latency).
Nmap scan report for dvwa.vm (10.6.6.13)
Host is up (0.00014s latency).
Nmap scan report for mutillidae.vm (10.6.6.14)
Host is up (0.000054s latency).
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.0013s latency).
Nmap scan report for 10.6.6.100
Host is up (0.00046s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 2.47 seconds
```

Performing Enumeration with Nmap

1. Verifying the environment of Nmap

Use the nmap -V command to verify that Nmap is installed and to display the Nmap version.

Syntax: $ nmap -V <Ip address>
Example: $ nmap -V 10.6.6.23

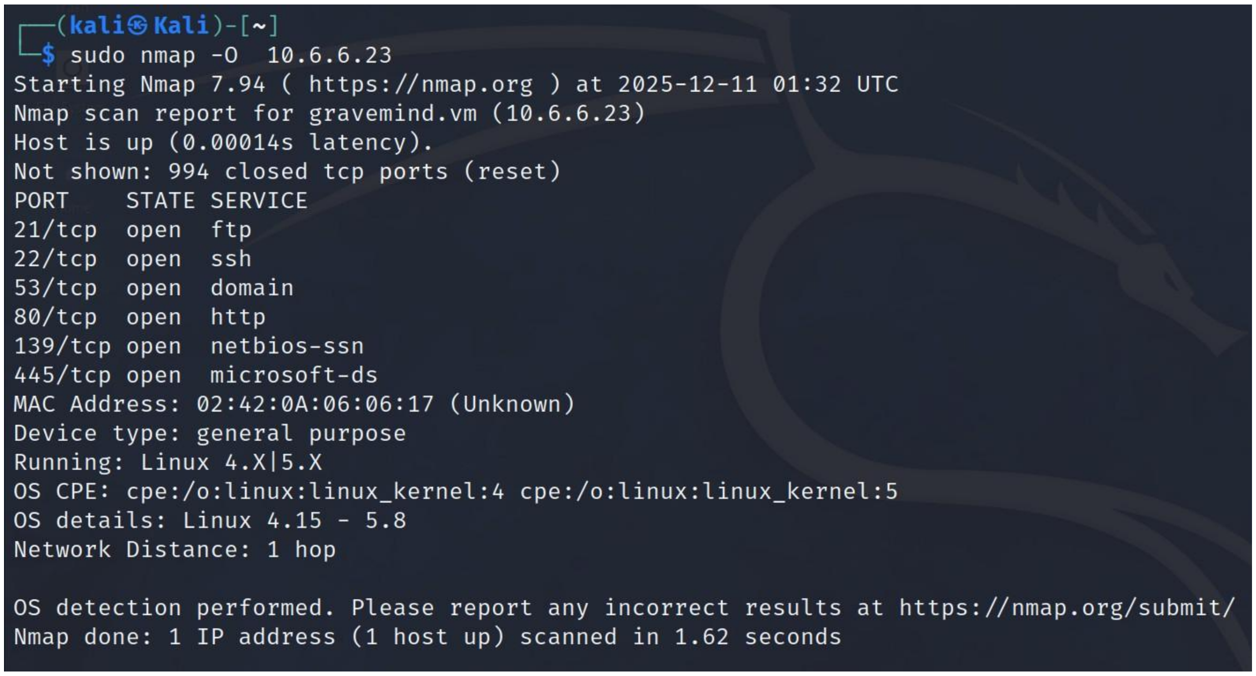


```
┌──(kali㉿Kali)-[~]
└─$ nmap -V 10.6.6.23
Nmap version 7.94 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.4.4 openssl-3.0.9 libssh2-1.10.0 libz-1.2.13 libpcre-8.39 libpcap-1.10.4 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

2. Using the different Nmap Options and Features

Using the command nmap without specifying any options or targets returns a list of commonly used Nmap options. To access the Nmap help system, use the command nmap -h. The help output is divided into sections based on the type of detection that the option supports. The man page for Nmap provides additional information. To access the man page, enter the command man nmap.

Syntax: $ sudo nmap -A/-O/-v/--open <Ip address>
Example1: $ sudo nmap -A 10.6.6.23
Example2: $ sudo nmap -O 10.6.6.23
Example1: $ sudo nmap -v 10.6.6.23
Example1: $ sudo nmap --open 10.6.6.23



```
┌──(kali㉿Kali)-[~]
└─$ sudo nmap -O  10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:32 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00014s latency).
Not shown: 994 closed tcp ports (reset)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
53/tcp  open  domain
80/tcp  open  http
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 02:42:0A:06:06:17 (Unknown)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.8
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 1.62 seconds
```

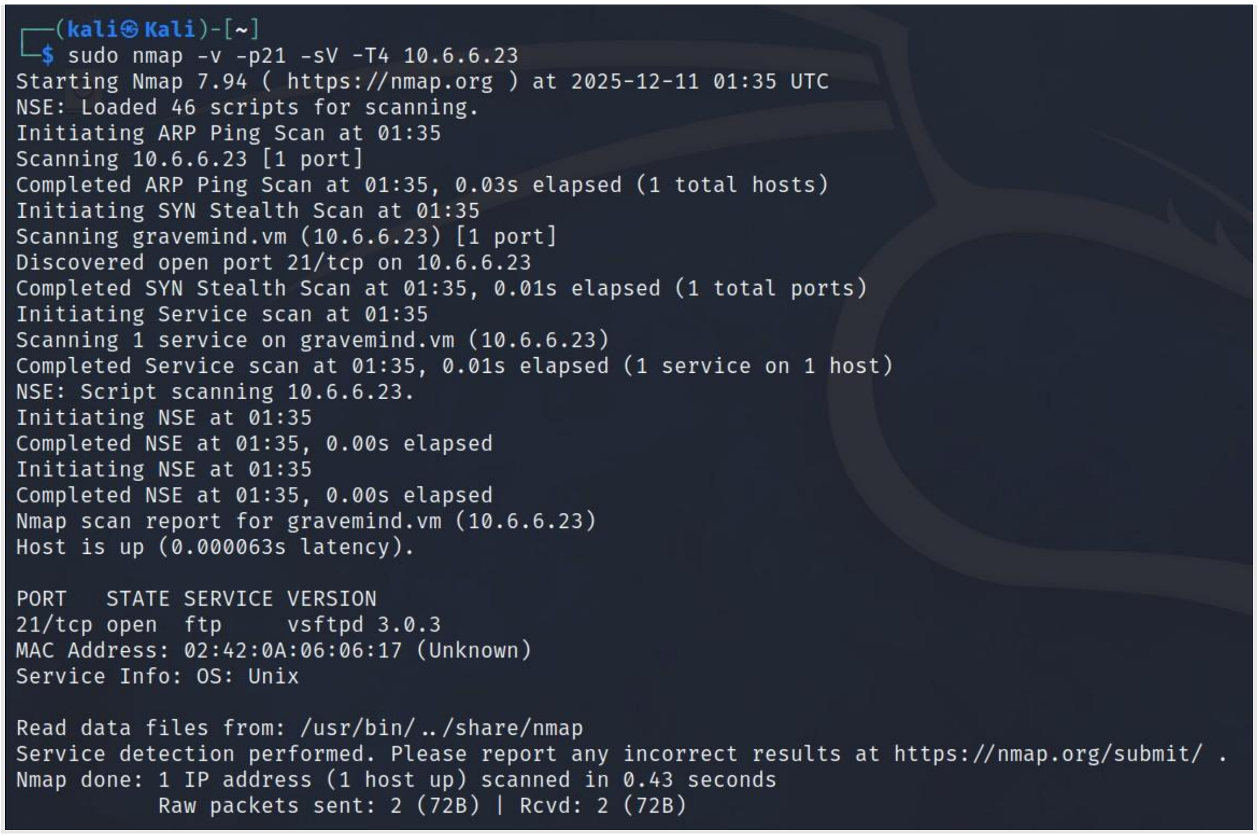
Options:

-A	- Aggressive scan
-O	- OS detection
-v	- Increases the verbosity
--open	- Only reports open ports

3. Obtaining additional information about the host and services

To provide additional information about the target computer, it is possible to combine different options into a single command line. The use of options like -v, -p, and -sV provides additional information about the services running on the open ports. The following command provides information about the FTP service running on port 21 on the target in verbose mode, with the timing set to fast (-T4):

```
Syntax: $ sudo nmap -v -p21 -sV -T4 <Ip address>
Example: $ sudo nmap -v -p21 -sV -T4 10.6.6.23
```



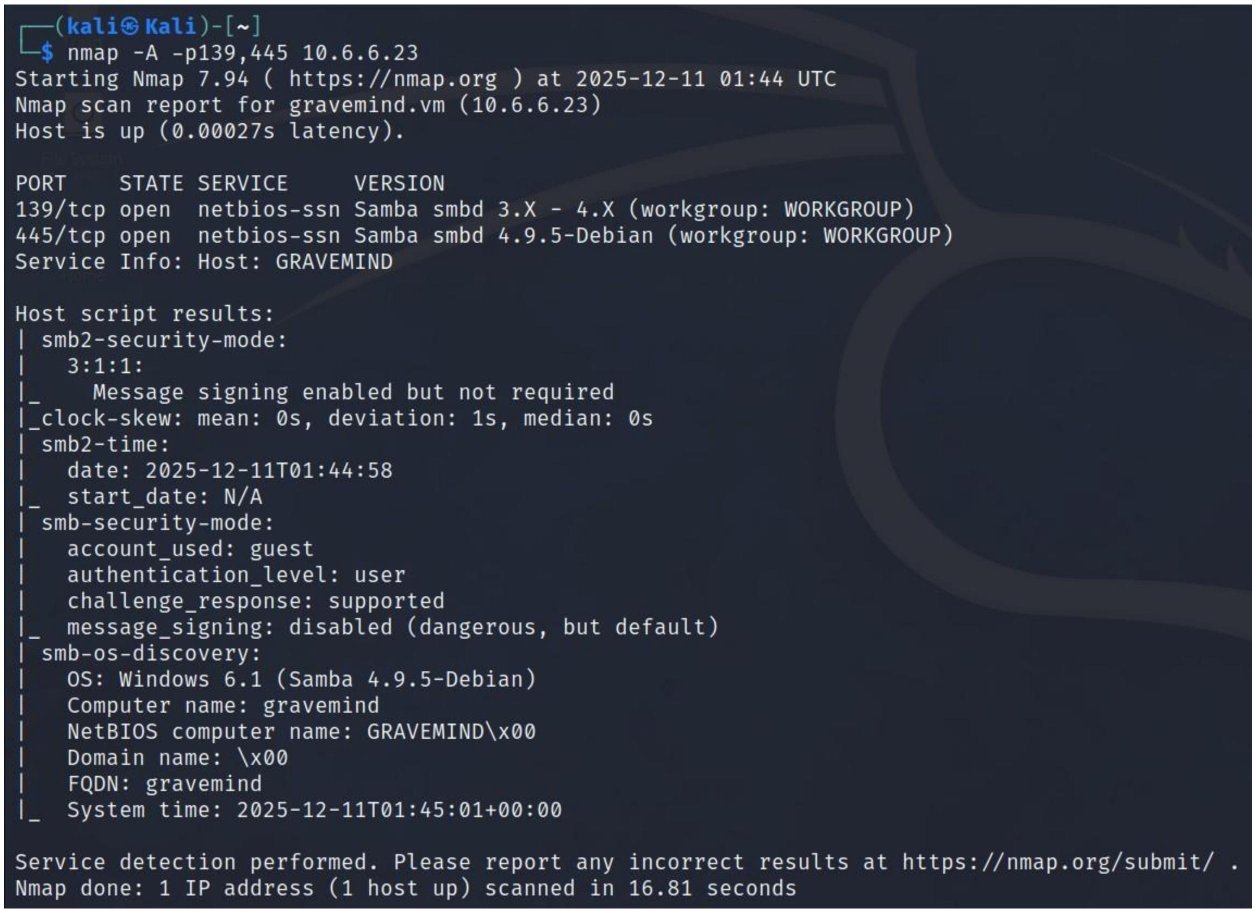
```
┌──(kali㉿Kali)-[~]
└─$ sudo nmap -v -p21 -sV -T4 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:35 UTC
NSE: Loaded 46 scripts for scanning.
Initiating ARP Ping Scan at 01:35
Scanning 10.6.6.23 [1 port]
Completed ARP Ping Scan at 01:35, 0.03s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 01:35
Scanning gravemind.vm (10.6.6.23) [1 port]
Discovered open port 21/tcp on 10.6.6.23
Completed SYN Stealth Scan at 01:35, 0.01s elapsed (1 total ports)
Initiating Service scan at 01:35
Scanning 1 service on gravemind.vm (10.6.6.23)
Completed Service scan at 01:35, 0.01s elapsed (1 service on 1 host)
NSE: Script scanning 10.6.6.23.
Initiating NSE at 01:35
Completed NSE at 01:35, 0.00s elapsed
Initiating NSE at 01:35
Completed NSE at 01:35, 0.00s elapsed
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.000063s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
MAC Address: 02:42:0A:06:06:17 (Unknown)
Service Info: OS: Unix

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.43 seconds
           Raw packets sent: 2 (72B) | Rcvd: 2 (72B)
```

4. Investigating SMB Services with Scripts

The Server Message Block (SMB) protocol is a network file sharing protocol supported on Windows computers and by SAMBA on Linux. SMB enables applications to read and write files or request services over a network. Open public shares or shared devices, such as print servers on a network, can be accessed through SMB.

The earlier scan of open ports on the target computer indicates that the SMB ports 139 and 445 are open. We can find more information on these ports using the -A and -p command options as given below.

```
Syntax: $ nmap -A -p139,445 <Ip address>
Example: $ nmap -A -p139,445 10.6.6.23
```



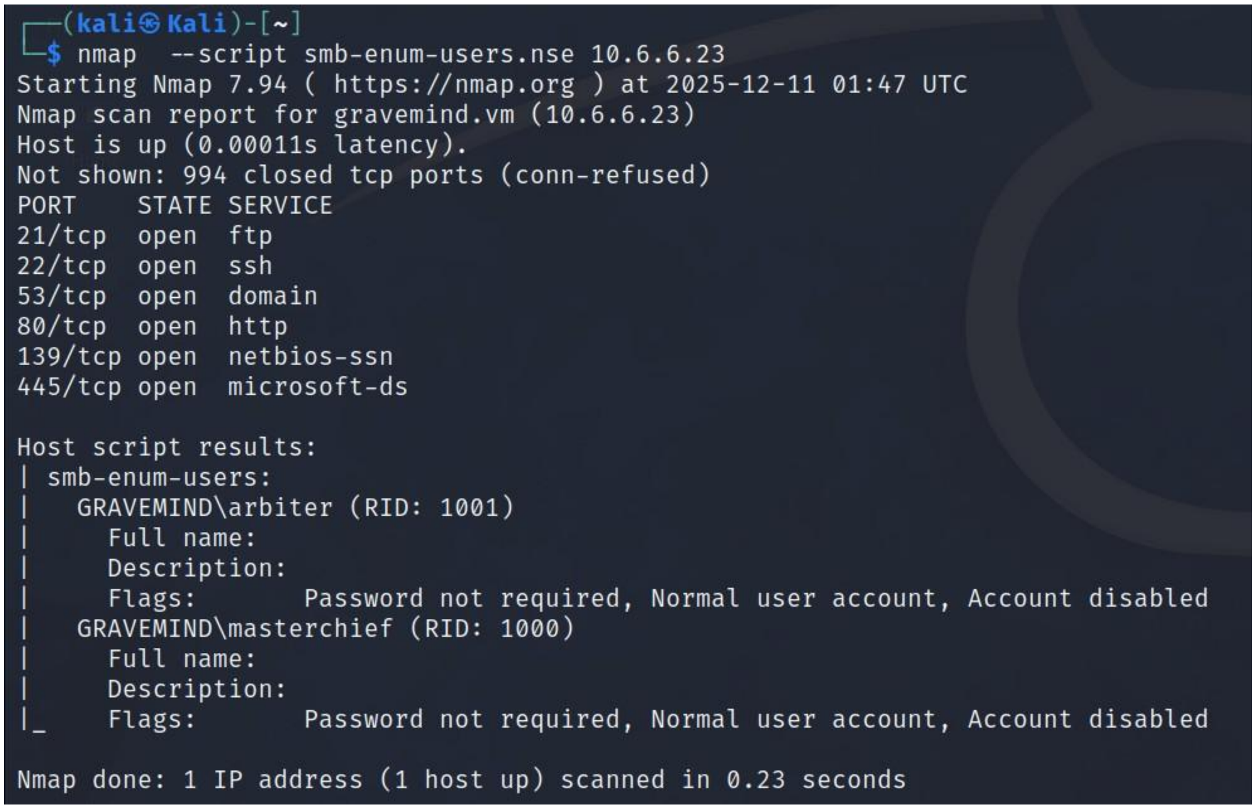
```
┌──(kali㉿Kali)-[~]
└─$ nmap -A -p139,445 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:44 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00027s latency).

PORT    STATE SERVICE     VERSION
139/tcp open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp open  netbios-ssn Samba smbd 4.9.5-Debian (workgroup: WORKGROUP)
Service Info: Host: GRAVEMIND

Host script results:
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required
|_clock-skew: mean: 0s, deviation: 1s, median: 0s
| smb2-time:
|   date: 2025-12-11T01:44:58
|_  start_date: N/A
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb-os-discovery:
|   OS: Windows 6.1 (Samba 4.9.5-Debian)
|   Computer name: gravemind
|   NetBIOS computer name: GRAVEMIND\x00
|   Domain name: \x00
|   FQDN: gravemind
|_  System time: 2025-12-11T01:45:01+00:00

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.81 seconds
```

Nmap contains the powerful Nmap Scripting Engine (NSE), which enables the programming of various Nmap options and conditional actions to be taken as a result of the responses. NSE has built-in scripts that enumerate users, groups, and network shares.

Enumerating Users

Gathering a valid list of users is the first step in cracking a set of credentials. When we have the username, we can then begin brute-force attempts to get the account password. User enumeration can be performed after gaining access to the internal network. It can be done by manipulating the Server Message Block (SMB) protocol, in Windows OS, which uses TCP port 445. One of the more commonly used scripts for SMB discovery is the **smb-enum-users.nse** script.

```
Syntax: #nmap --script smb-enum-users.nse <IP address>
Example: #nmap  --script smb-enum-users.nse 10.6.6.23
```

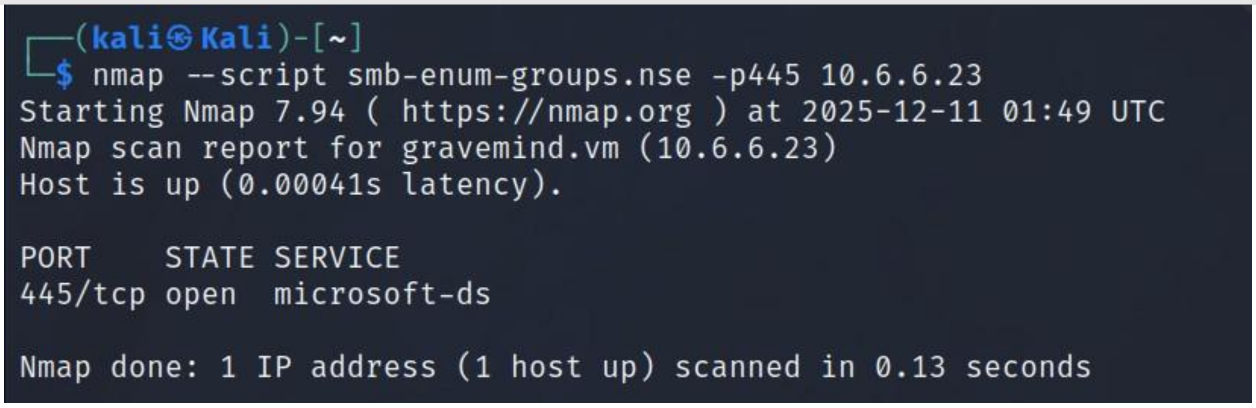


```
┌──(kali㉿Kali)-[~]
└─$ nmap  --script smb-enum-users.nse 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:47 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00011s latency).
Not shown: 994 closed tcp ports (conn-refused)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
53/tcp  open  domain
80/tcp  open  http
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds

Host script results:
| smb-enum-users: 
|   GRAVEMIND\arbiter (RID: 1001)
|     Full name:   
|     Description: 
|     Flags:       Password not required, Normal user account, Account disabled
|   GRAVEMIND\masterchief (RID: 1000)
|     Full name:   
|     Description: 
|_    Flags:       Password not required, Normal user account, Account disabled

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
```

Enumerating Groups

Group enumeration helps determine the authorization roles that are being used in the target environment. The Nmap NSE script for enumerating SMB groups is smb-enum-groups. This script attempts to pull a list of groups from a remote Windows machine.

Syntax: nmap --script smb-enum-groups.nse -p445 <Ip address>
Example: nmap --script smb-enum-groups.nse -p445 10.6.6.23

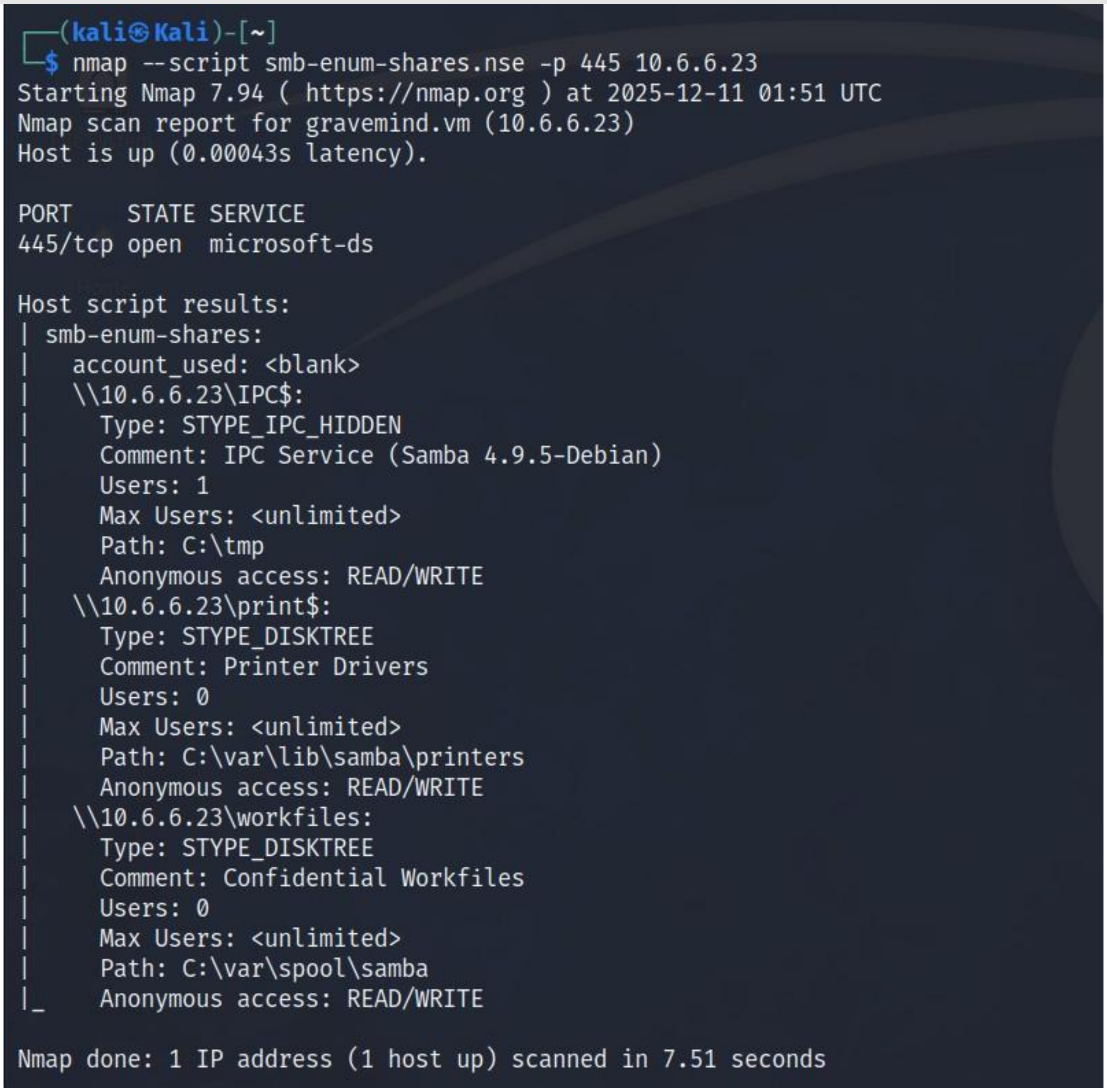


```
┌──(kali㉿Kali)-[~]
└─$ nmap --script smb-enum-groups.nse -p445 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:49 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00041s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 0.13 seconds
```

Enumerating Network Share

Identifying systems on a network that are sharing files, folders, and printers helps build out the attack surface of an internal network. The Nmap smb-enum-shares NSE script uses Microsoft Remote Procedure Call (MSRPC) for network share enumeration.

Syntax: nmap --script smb-enum-shares.nse -p 445 <host>
Example: nmap --script smb-enum-shares.nse -p 445 10.6.6.23



```
┌──(kali㉿Kali)-[~]
└─$ nmap --script smb-enum-shares.nse -p 445 10.6.6.23
Starting Nmap 7.94 ( https://nmap.org ) at 2025-12-11 01:51 UTC
Nmap scan report for gravemind.vm (10.6.6.23)
Host is up (0.00043s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb-enum-shares:
|   account_used: <blank>
|   \\10.6.6.23\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: IPC Service (Samba 4.9.5-Debian)
|     Users: 1
|     Max Users: <unlimited>
|     Path: C:\tmp
|     Anonymous access: READ/WRITE
|   \\10.6.6.23\print$:
|     Type: STYPE_DISKTREE
|     Comment: Printer Drivers
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\var\lib\samba\printers
|     Anonymous access: READ/WRITE
|   \\10.6.6.23\workfiles:
|     Type: STYPE_DISKTREE
|     Comment: Confidential Workfiles
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\var\spool\samba
|_    Anonymous access: READ/WRITE

Nmap done: 1 IP address (1 host up) scanned in 7.51 seconds
```

Lab 2- Packet Crafting with Scapy

In this lab, we will use Scapy, a Python-based packet manipulation tool, to craft custom packets. These custom packets will be used to perform reconnaissance on a target system.

Part 1: Investigate the Scapy Tool.

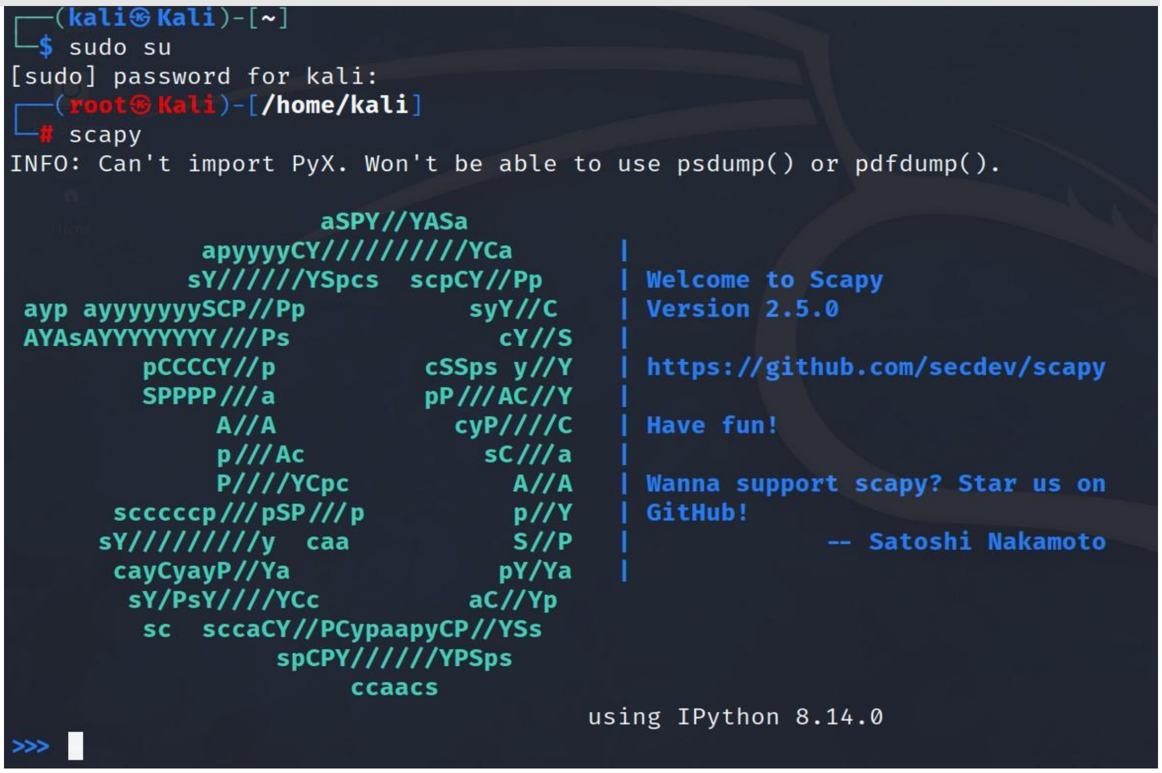
Part 2: Use Scapy to Sniff Network Traffic.

Part 3: Create and Send an ICMP Packet.

Part 4: Create and Send TCP SYN Packets.

Part 1: Investigate the Scapy Tool.

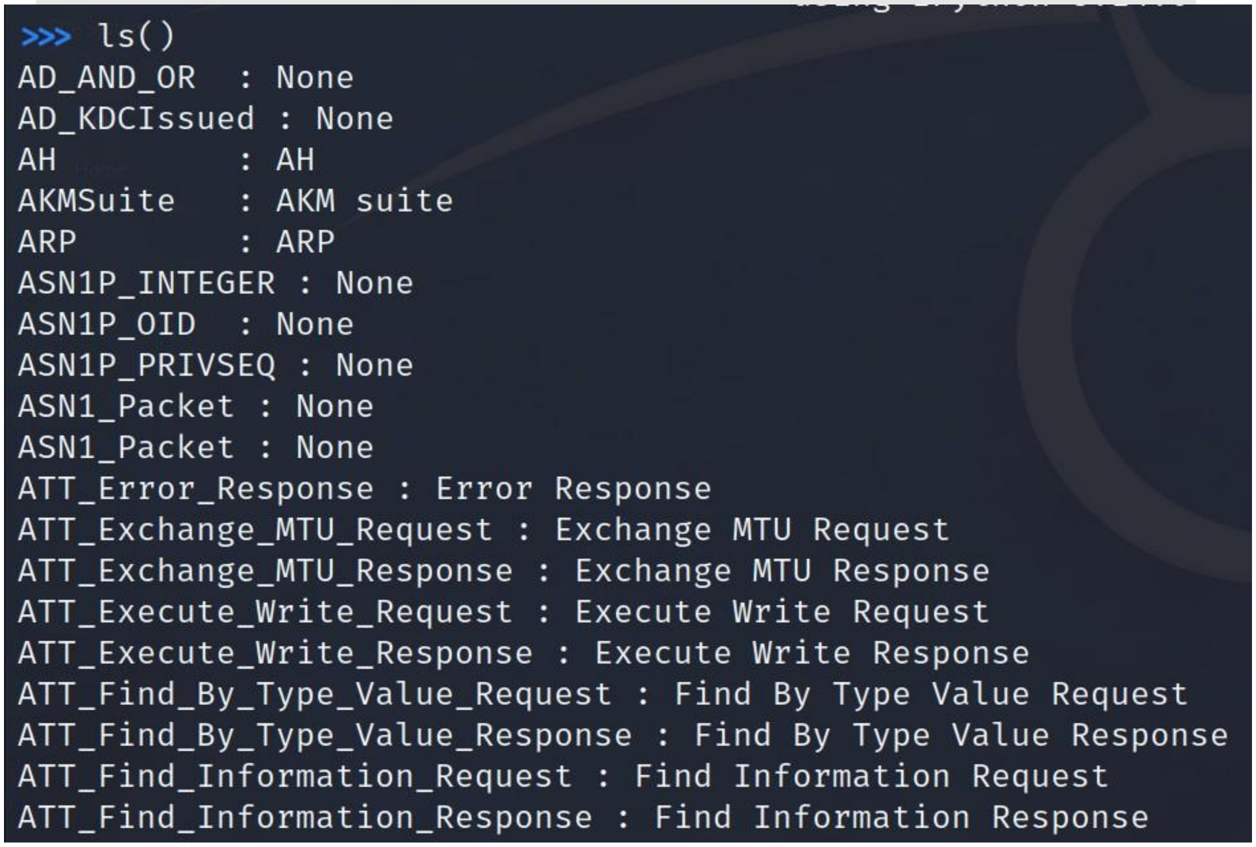
Scapy is a Python program that enables the user to send, sniff and dissect, and forge network packets. This capability allows the construction of tools that can probe, scan, or attack networks.

1. Use the **sudo su** command to obtain root privileges before starting Scapy.
```
–$ sudo su
```
2. Enter the scapy command in a terminal window to load the Python interpreter.
```
# scapy
```


```
┌──(kali㉿Kali)-[~]
└─$ sudo su
[sudo] password for kali:
┌──(root㉿Kali)-[/home/kali]
└─# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.5.0
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Wanna support scapy? Star us on
       scccccp///pSP///p          p//Y   | GitHub!
      sY/////////y  caa           S//P   |             -- Satoshi Nakamoto
       cayCyayP//Ya              pY/Ya   |
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 8.14.0
>>> 
```

3. At the >>> prompt within the Scapy shell, enter the ls() function to list all of the available default formats and protocols included with the tool.

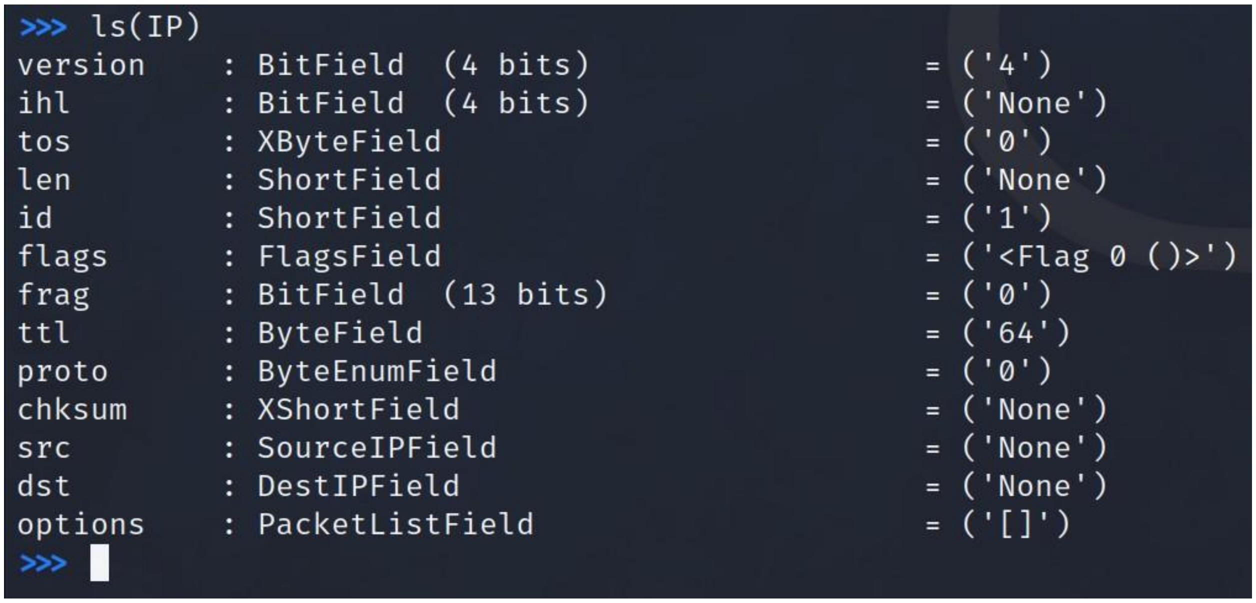
```
>>> ls()
```



```
>>> ls()
AD_AND_OR   : None
AD_KDCIssued : None
AH          : AH
AKMSuite    : AKM suite
ARP         : ARP
ASN1P_INTEGER : None
ASN1P_OID   : None
ASN1P_PRIVSEQ : None
ASN1_Packet : None
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Execute_Write_Request : Execute Write Request
ATT_Execute_Write_Response : Execute Write Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
ATT_Find_By_Type_Value_Response : Find By Type Value Response
ATT_Find_Information_Request : Find Information Request
ATT_Find_Information_Response : Find Information Response
```

4. Use the ls(IP) function to list the available fields in an IP packet header.

```
>>> ls(IP)
```

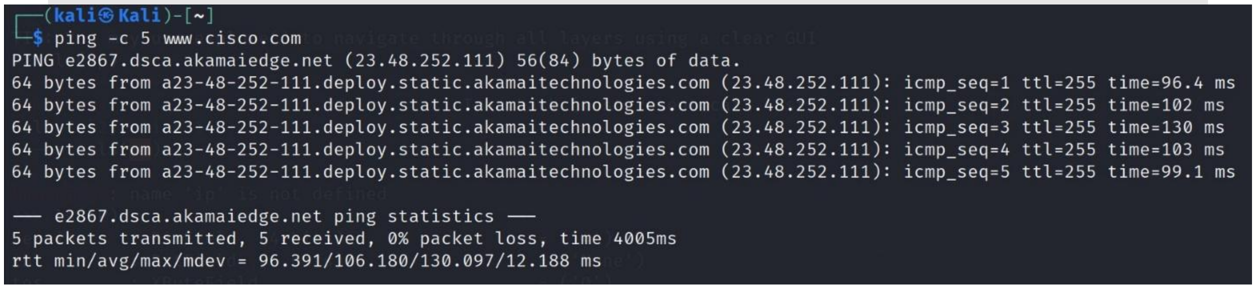


```
>>> ls(IP)
version    : BitField  (4 bits)                  = ('4')
ihl        : BitField  (4 bits)                  = ('None')
tos        : XByteField                          = ('0')
len        : ShortField                          = ('None')
id         : ShortField                          = ('1')
flags      : FlagsField                          = ('<Flag 0 ()>')
frag       : BitField  (13 bits)                 = ('0')
ttl        : ByteField                           = ('64')
proto      : ByteEnumField                       = ('0')
chksum     : XShortField                         = ('None')
src        : SourceIPField                       = ('None')
dst        : DestIPField                         = ('None')
options    : PacketListField                     = ('[]')
>>> 
```

Part 2: Use Scapy to Sniff Network Traffic.

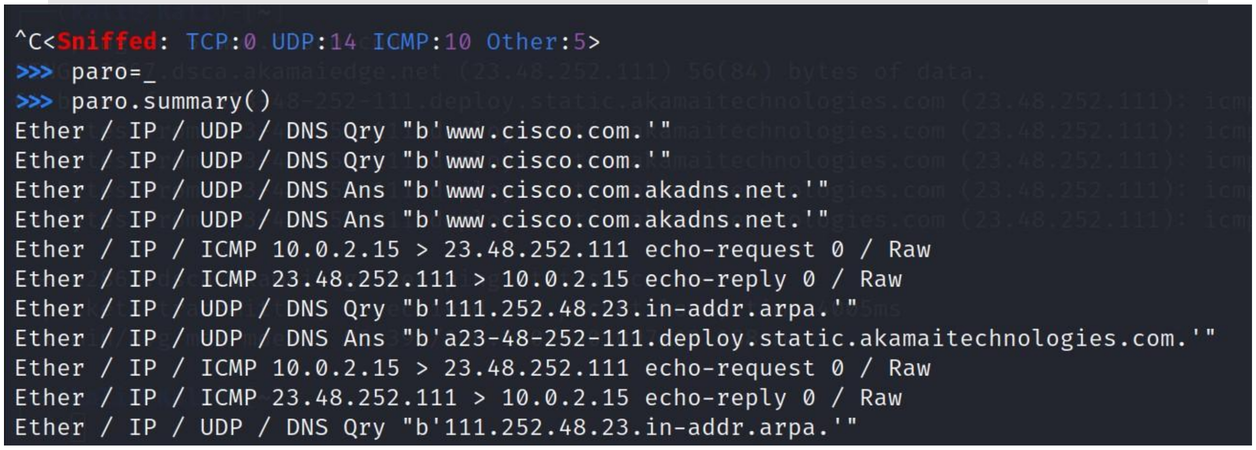
Scapy can be used to capture and display network traffic, similar to a tcpdump or tshark packet collection.

1. Use the sniff() function to collect traffic using the default eth0 interface of wer VM. Start the capture with the sniff() function without specifying any arguments.
   ```
   >>> sniff()
   ```
2. Open a second terminal window and ping an internet address with specifying the count using the -c argument.
   ```
   $ ping -c 5 www.cisco.com
   ```



```
┌──(kali㉿Kali)-[~]
└─$ ping -c 5 www.cisco.com
PING e2867.dsca.akamaiedge.net (23.48.252.111) 56(84) bytes of data.
64 bytes from a23-48-252-111.deploy.static.akamaitechnologies.com (23.48.252.111): icmp_seq=1 ttl=255 time=96.4 ms
64 bytes from a23-48-252-111.deploy.static.akamaitechnologies.com (23.48.252.111): icmp_seq=2 ttl=255 time=102 ms
64 bytes from a23-48-252-111.deploy.static.akamaitechnologies.com (23.48.252.111): icmp_seq=3 ttl=255 time=130 ms
64 bytes from a23-48-252-111.deploy.static.akamaitechnologies.com (23.48.252.111): icmp_seq=4 ttl=255 time=103 ms
64 bytes from a23-48-252-111.deploy.static.akamaitechnologies.com (23.48.252.111): icmp_seq=5 ttl=255 time=99.1 ms

--- e2867.dsca.akamaiedge.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 96.391/106.180/130.097/12.188 ms
```

3. Return to the terminal window that is running the Scapy tool. Press CTRL-C to stop the capture.
4. View the captured traffic using the summary() function. The paro=_ assigns the variable a to hold the output of the sniff() function. The underscore (_) in Python is used to temporarily hold the output of the last function executed.
   ```
   >>> paro=_
   >>> paro.summary()
   ```



```
^C<Sniffed: TCP:0 UDP:14 ICMP:10 Other:5>
>>> paro=_
>>> paro.summary()
Ether / IP / UDP / DNS Qry "b'www.cisco.com.'"
Ether / IP / UDP / DNS Qry "b'www.cisco.com.'"
Ether / IP / UDP / DNS Ans "b'www.cisco.com.akadns.net.'"
Ether / IP / UDP / DNS Ans "b'www.cisco.com.akadns.net.'"
Ether / IP / ICMP 10.0.2.15 > 23.48.252.111 echo-request 0 / Raw
Ether / IP / ICMP 23.48.252.111 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'111.252.48.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-48-252-111.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 23.48.252.111 echo-request 0 / Raw
Ether / IP / ICMP 23.48.252.111 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'111.252.48.23.in-addr.arpa.'"
```

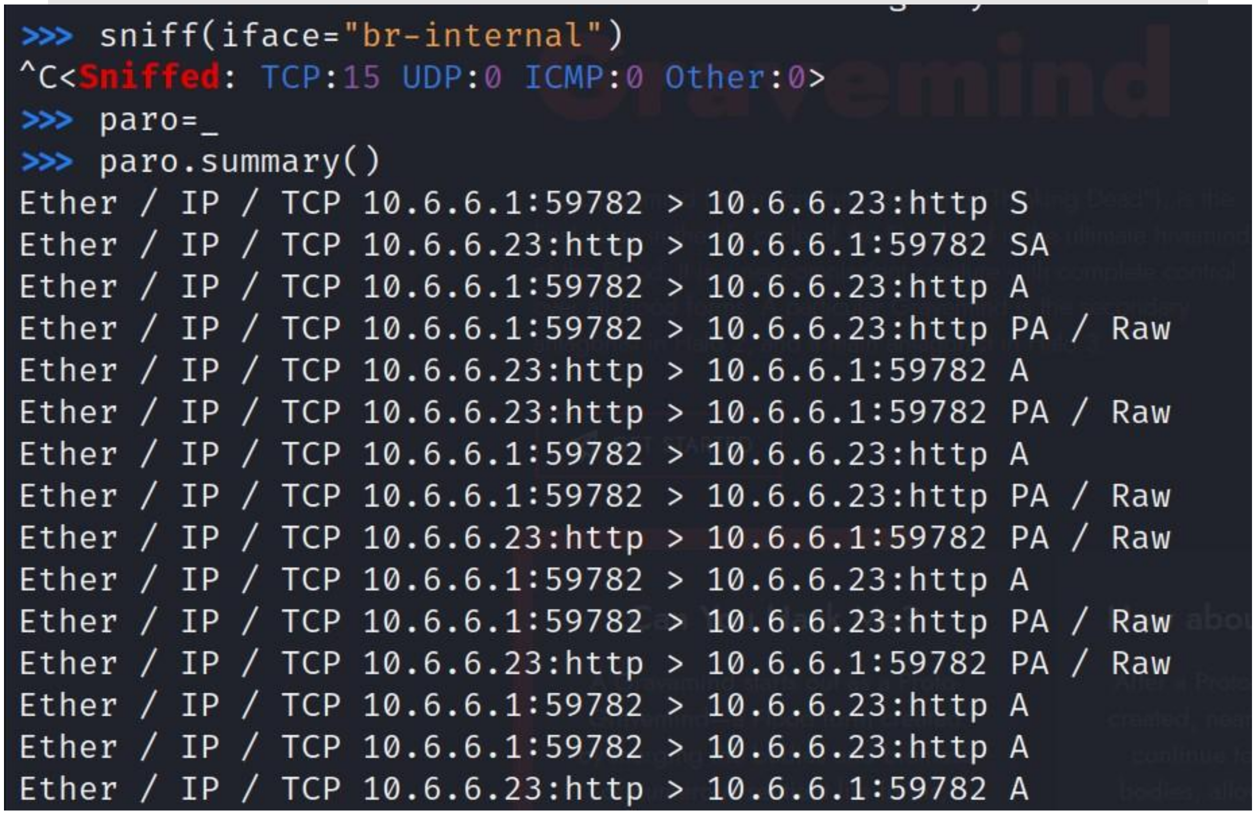
5. Open a new terminal window. Use the ifconfig command to determine the name of the interface that is assigned the IP address 10.6.6.1

6. Return to the terminal window that is running the Scapy tool. Use the syntax sniff(iface="interface name") to begin the capture on the br-internal virtual interface.

```
>>> sniff(iface="br-internal")
```

7. Open Firefox and navigate to the URL **http://10.6.6.23/.** When the Gravemind home page opens, return to the terminal window that is running the Scapy tool. Press **CTRL-C.**
8. View the captured traffic with the following function.

```
>>> paro=_
>>> paro.summary()
```



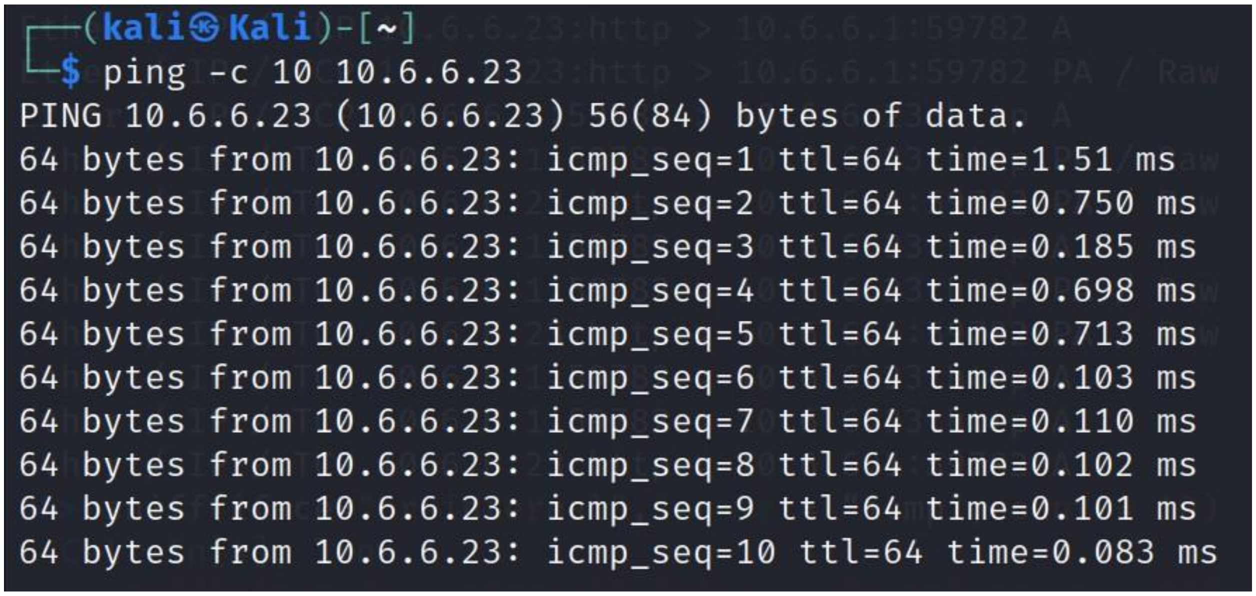
```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:15 UDP:0 ICMP:0 Other:0>
>>> paro=_
>>> paro.summary()
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http S
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 SA
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 PA / Raw
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 PA / Raw
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 PA / Raw
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:59782 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:59782 A
```

9. Use the interface ID associated with 10.6.6.1 (br-internal) to capture ten ICMP packets sent and received on the internal virtual network.

```
>>> sniff(iface="br-internal",filter = “icmp",count = 10)
```

10. Open a second terminal window and ping the host at 10.6.6.23.

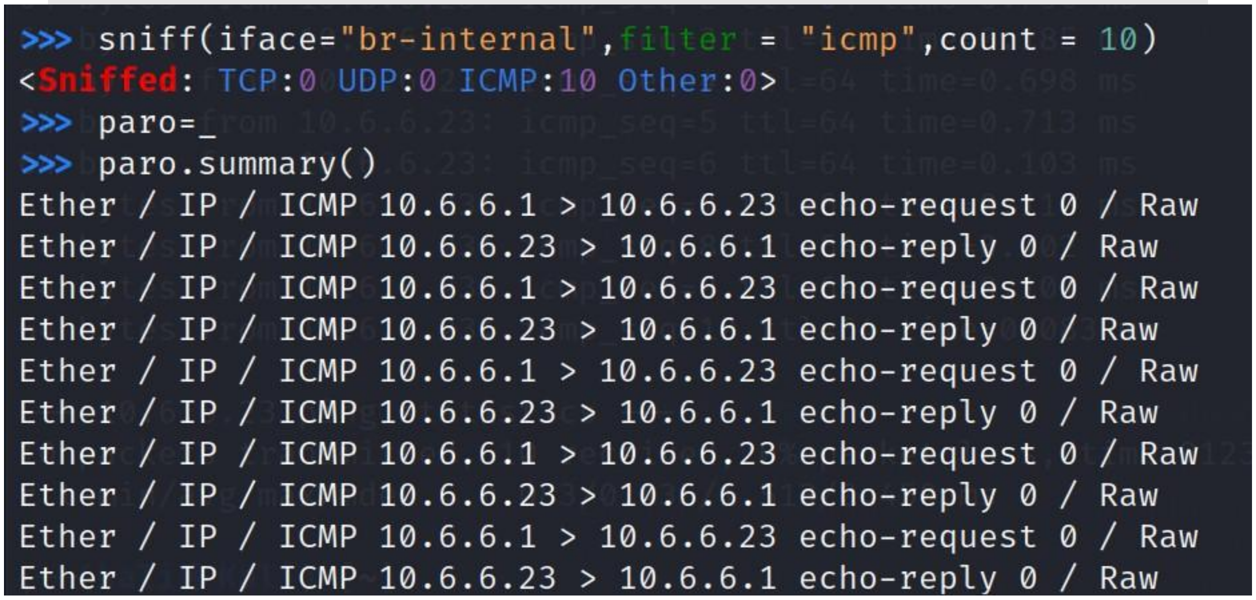
```
$ ping –c 10 10.6.6.23
```



```
┌──(kali㉿Kali)-[~]
└─$ ping -c 10 10.6.6.23
PING 10.6.6.23 (10.6.6.23) 56(84) bytes of data.
64 bytes from 10.6.6.23: icmp_seq=1 ttl=64 time=1.51 ms
64 bytes from 10.6.6.23: icmp_seq=2 ttl=64 time=0.750 ms
64 bytes from 10.6.6.23: icmp_seq=3 ttl=64 time=0.185 ms
64 bytes from 10.6.6.23: icmp_seq=4 ttl=64 time=0.698 ms
64 bytes from 10.6.6.23: icmp_seq=5 ttl=64 time=0.713 ms
64 bytes from 10.6.6.23: icmp_seq=6 ttl=64 time=0.103 ms
64 bytes from 10.6.6.23: icmp_seq=7 ttl=64 time=0.110 ms
64 bytes from 10.6.6.23: icmp_seq=8 ttl=64 time=0.102 ms
64 bytes from 10.6.6.23: icmp_seq=9 ttl=64 time=0.101 ms
64 bytes from 10.6.6.23: icmp_seq=10 ttl=64 time=0.083 ms
```

11. Return to the terminal window running the Scapy tool. The capture automatically stopped when 10 packets were sent or received. View the captured traffic with line numbers using the nsummary() function.

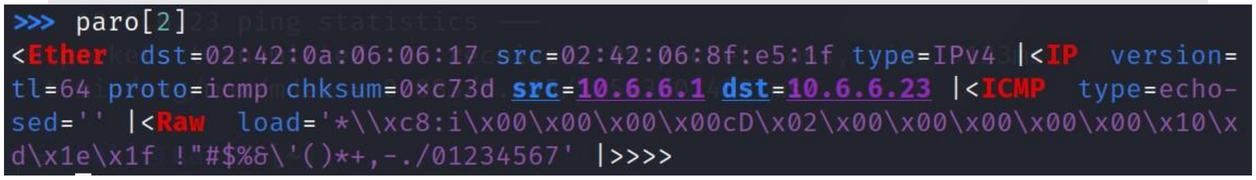
```
>>> paro=_
>>> paro.nsummary()
```



```
>>> sniff(iface="br-internal",filter = "icmp",count = 10)
<Sniffed: TCP:0 UDP:0 ICMP:10 Other:0>
>>> paro=_
>>> paro.summary()
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
```

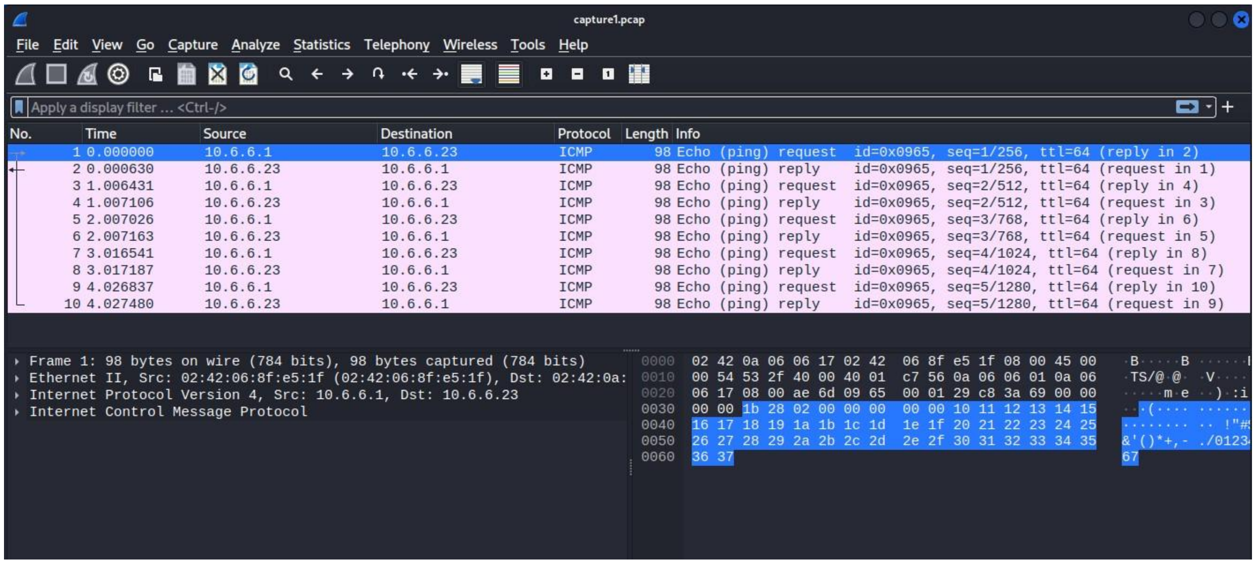
12. View details about a specific packet in the series using the following command, including the blue line number of the packet without the leading zeros.

```
>>> paro[2]
```



```
>>> paro[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:06:8f:e5:1f type=IPv4 |<IP  version=
tl=64 proto=icmp chksum=0×c73d src=10.6.6.1 dst=10.6.6.23 |<ICMP  type=echo-
sed='' |<Raw  load='*\\xc8:i\x00\x00\x00\x00cD\x02\x00\x00\x00\x00\x00\x10\x
d\x1e\x1f !"#$%&\'()*+,-./01234567' |>>>>
```

13. Use the wrpcap() function to save the captured data to a pcap file that can be opened by Wireshark and other applications.
    ```
    >>> wrpcap(“capture1.pcap", paro)
    ```
14. The .pcap file will be written to the default user directory.
15. Open the capture in Wireshark to view the file contents.


capture1.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time        Source      Destination  Protocol Length Info
   1 0.000000    10.6.6.1    10.6.6.23    ICMP     98 Echo (ping) request  id=0x0965, seq=1/256, ttl=64 (reply in 2)
   2 0.000630    10.6.6.23   10.6.6.1     ICMP     98 Echo (ping) reply    id=0x0965, seq=1/256, ttl=64 (request in 1)
   3 1.006431    10.6.6.1    10.6.6.23    ICMP     98 Echo (ping) request  id=0x0965, seq=2/512, ttl=64 (reply in 4)
   4 1.007106    10.6.6.23   10.6.6.1     ICMP     98 Echo (ping) reply    id=0x0965, seq=2/512, ttl=64 (request in 3)
   5 2.007026    10.6.6.1    10.6.6.23    ICMP     98 Echo (ping) request  id=0x0965, seq=3/768, ttl=64 (reply in 6)
   6 2.007163    10.6.6.23   10.6.6.1     ICMP     98 Echo (ping) reply    id=0x0965, seq=3/768, ttl=64 (request in 5)
   7 3.016541    10.6.6.1    10.6.6.23    ICMP     98 Echo (ping) request  id=0x0965, seq=4/1024, ttl=64 (reply in 8)
   8 3.017187    10.6.6.23   10.6.6.1     ICMP     98 Echo (ping) reply    id=0x0965, seq=4/1024, ttl=64 (request in 7)
   9 4.026837    10.6.6.1    10.6.6.23    ICMP     98 Echo (ping) request  id=0x0965, seq=5/1280, ttl=64 (reply in 10)
  10 4.027480    10.6.6.23   10.6.6.1     ICMP     98 Echo (ping) reply    id=0x0965, seq=5/1280, ttl=64 (request in 9)
```

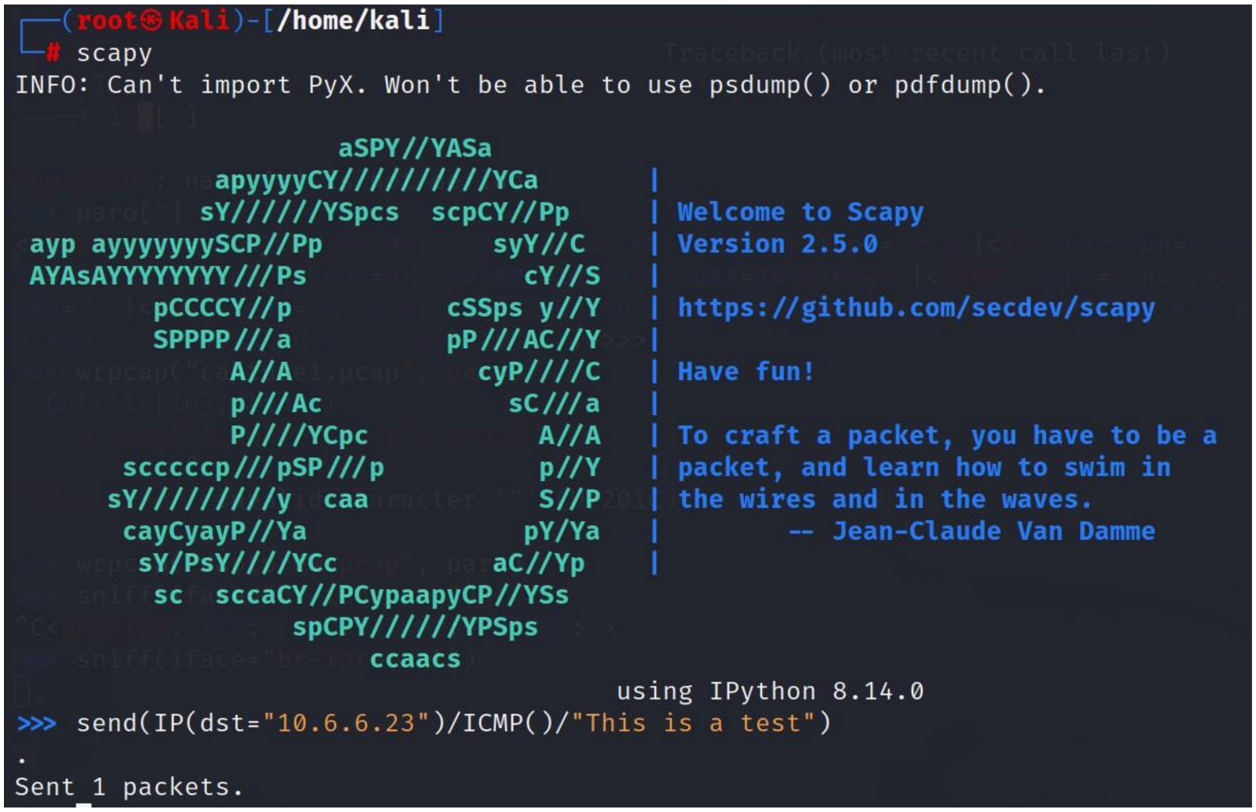
```
› Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
› Ethernet II, Src: 02:42:06:8f:e5:1f (02:42:06:8f:e5:1f), Dst: 02:42:0a:
› Internet Protocol Version 4, Src: 10.6.6.1, Dst: 10.6.6.23
› Internet Control Message Protocol
```

```
0000  02 42 0a 06 06 17 02 42  06 8f e5 1f 08 00 45 00   ·B·····B ······
0010  00 54 53 2f 40 00 40 01  c7 56 0a 06 06 01 0a 06   ·TS/@·@· ·V····
0020  06 17 08 00 ae 6d 09 65  00 01 29 c8 3a 69 00 00   ·····m·e ··)·:i
0030  00 00 1b 28 02 00 00 00  00 00 10 11 12 13 14 15   ···(···· ······
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./0123
0060  36 37                                              67
```

Part 3: Create and Send an ICMP Packet.

ICMP is a protocol designed to send control messages between network devices for various purposes.

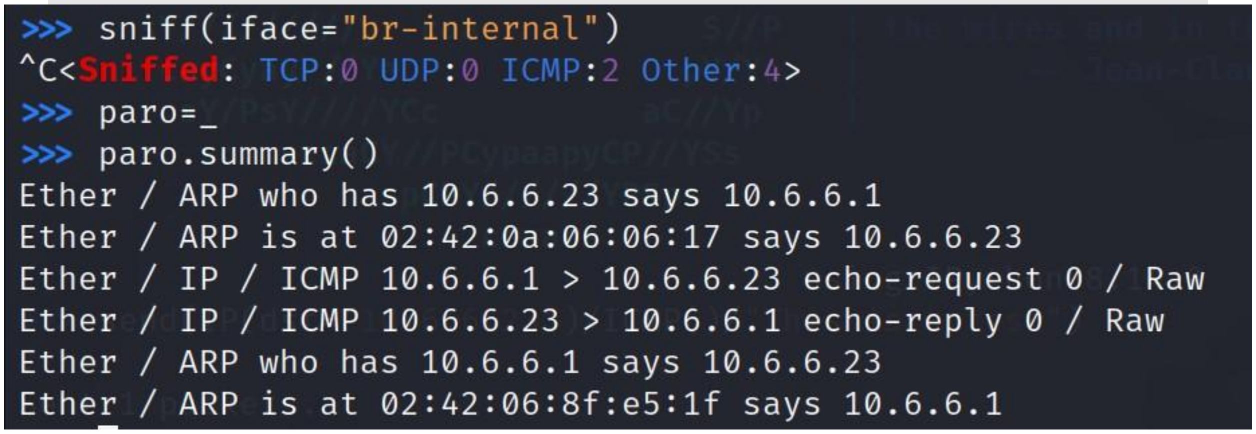
1. In a Scapy terminal window, enter the command to sniff traffic from the interface connected to the 10.6.6.0/24 network.
   ```
   >>> sniff(iface="br-internal")
   ```
2. Open another terminal window, enter sudo su to perform packet crafting as root. Start a second instance of Scapy. Enter the send() function to send a packet to 10.6.6.23 with a modified ICMP payload.
   ```
   $ sudo su
   # scapy
   send(IP(dst="10.6.6.23")/ICMP()/"This is a test")
   ```



```
┌──(root㉿Kali)-[/home/kali]
└─# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.5.0
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | To craft a packet, you have to be a
       scccccp///pSP///p          p//Y   | packet, and learn how to swim in
      sY/////////y  caa           S//P   | the wires and in the waves.
       cayCyayP//Ya              pY/Ya   |        -- Jean-Claude Van Damme
        sY/PsY////YCc          aC//Yp    |
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 8.14.0
>>> send(IP(dst="10.6.6.23")/ICMP()/"This is a test")
.
Sent 1 packets.
```

3. Return to the first terminal window and press CTRL-C.
4. Enter the summary command to display the summary with packet numbers.

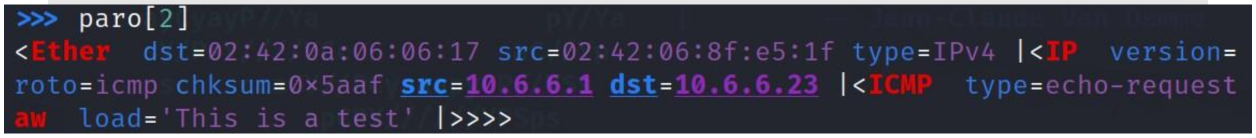
```
>>> paro=_
>>> paro.nsummary()
```



```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:0 UDP:0 ICMP:2 Other:4>
>>> paro=_
>>> paro.summary()
Ether / ARP who has 10.6.6.23 says 10.6.6.1
Ether / ARP is at 02:42:0a:06:06:17 says 10.6.6.23
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / ARP who has 10.6.6.1 says 10.6.6.23
Ether / ARP is at 02:42:06:8f:e5:1f says 10.6.6.1
```

5. Use the packet numbers to view the individual ICMP Echo-request and Echo-reply packets. Compare those packets to the ones that we examined in Part 2, Step 12.

```
>>> paro[2]
```



```
>>> paro[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:06:8f:e5:1f type=IPv4 |<IP  version=
roto=icmp chksum=0x5aaf src=10.6.6.1 dst=10.6.6.23 |<ICMP  type=echo-request
aw  load='This is a test' |>>>>
```

Part 4: Create and Send TCP SYN Packets.

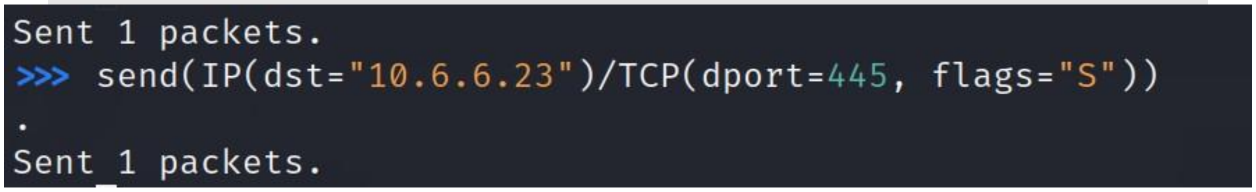
In this part, we will use Scapy to determine if port 445 (Microsoft Windows drive share port) is open on the target system at 10.6.6.23.

1. In the original Scapy terminal window, begin a packet capture on the internal interface attached to the 10.6.6.0/24 network.

```
>>> sniff(iface="br-internal")
```

2. Navigate to the second Scapy window. Create and send a TCP SYN packet using the command shown.

```
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
```

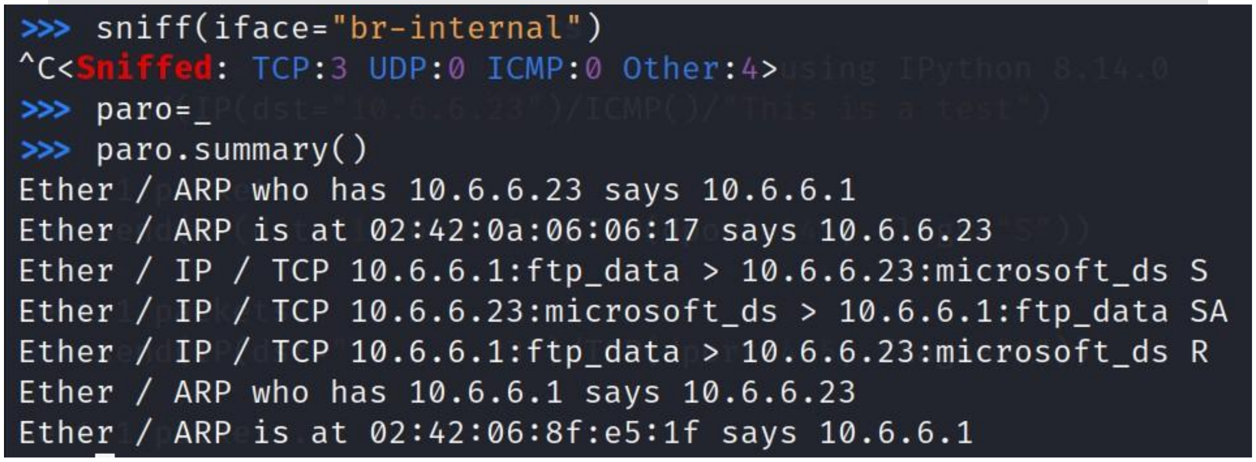


```
Sent 1 packets.
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
.
Sent 1 packets.
```

This command sent an IP packet to the host with IP address 10.6.6.23. The packet is addressed to TCP port 445 and has the S (SYN) flag set.

3. In the original Scapy terminal window, stop the packet capture by pressing CTRL-C.
4. View the captured TCP packets using the nsummary() function.

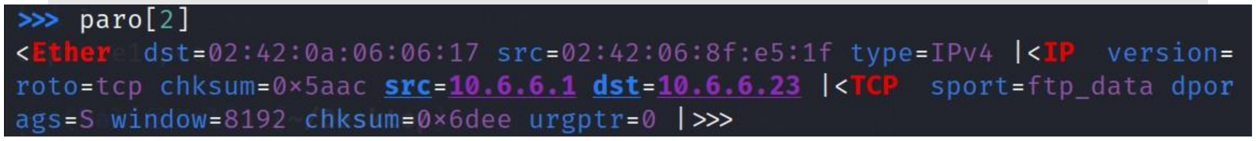
```
>>> paro=_
>>> paro.nsummary()
```



```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:3 UDP:0 ICMP:0 Other:4>
>>> paro=_
>>> paro.summary()
Ether / ARP who has 10.6.6.23 says 10.6.6.1
Ether / ARP is at 02:42:0a:06:06:17 says 10.6.6.23
Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds S
Ether / IP / TCP 10.6.6.23:microsoft_ds > 10.6.6.1:ftp_data SA
Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds R
Ether / ARP who has 10.6.6.1 says 10.6.6.23
Ether / ARP is at 02:42:06:8f:e5:1f says 10.6.6.1
```

5. View details about a specific packet in the series using the following command.

```
>>> paro[2]
```



```
>>> paro[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:06:8f:e5:1f type=IPv4 |<IP  version=
roto=tcp chksum=0x5aac src=10.6.6.1 dst=10.6.6.23 |<TCP  sport=ftp_data dpor
ags=S window=8192 chksum=0x6dee urgptr=0 |>>>
```

Lab 3- Network Sniffing with Wireshark

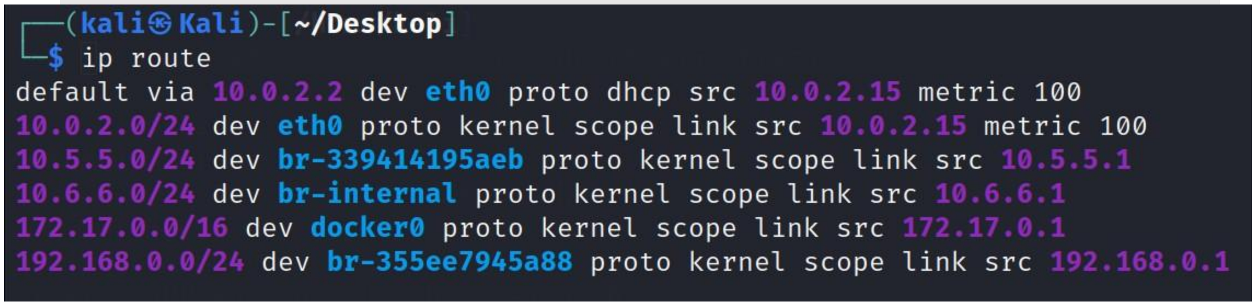
In this lab, we will use the Linux utility **tcpdump** to capture and save network traffic. Then we use Wireshark to investigate the traffic capture.

- Prepare the host to capture network traffic.
- Capture and save network traffic.
- View and analyze the Packet capture.

Part 1: Prepare the Host to Capture Network Traffic

1. Start the Kali workstation virtual machine.
2. Verify the user directory using the **pwd** command that will be used to store the captured traffic.
3. Determine the IP address of the Kali Ethernet interface using the **ifconfig** command.
4. Determine the default gateway assigned to the Kali host using the **ip route** command.

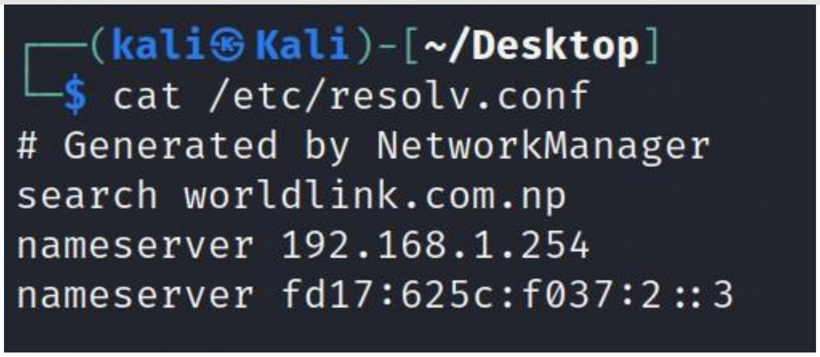
```
ip route
```



```
┌──(kali㉿Kali)-[~/Desktop]
└─$ ip route
default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15 metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15 metric 100
10.5.5.0/24 dev br-339414195aeb proto kernel scope link src 10.5.5.1
10.6.6.0/24 dev br-internal proto kernel scope link src 10.6.6.1
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
192.168.0.0/24 dev br-355ee7945a88 proto kernel scope link src 192.168.0.1
```

5. Determine the address of the configured default DNS server by displaying the contents of the **/etc/resolv.conf** file.
6. View the file using the cat <file name> command.

```
$ cat /etc/resolv.conf
```

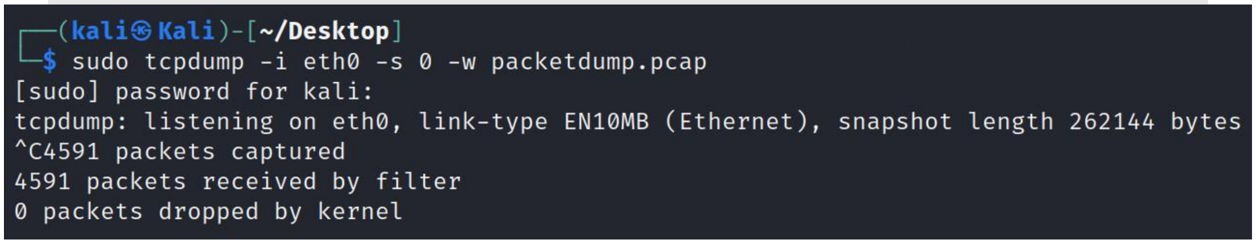


```
┌──(kali㉿Kali)-[~/Desktop]
└─$ cat /etc/resolv.conf
# Generated by NetworkManager
search worldlink.com.np
nameserver 192.168.1.254
nameserver fd17:625c:f037:2::3
```

Part 2: Capture and Save Network Traffic.

1. Open a terminal application and enter the command ifconfig.
2. Enter the sudo tcpdump command as shown.

```
$ sudo tcpdump -i eth0 -s 0 -w packetdump.pcap
```

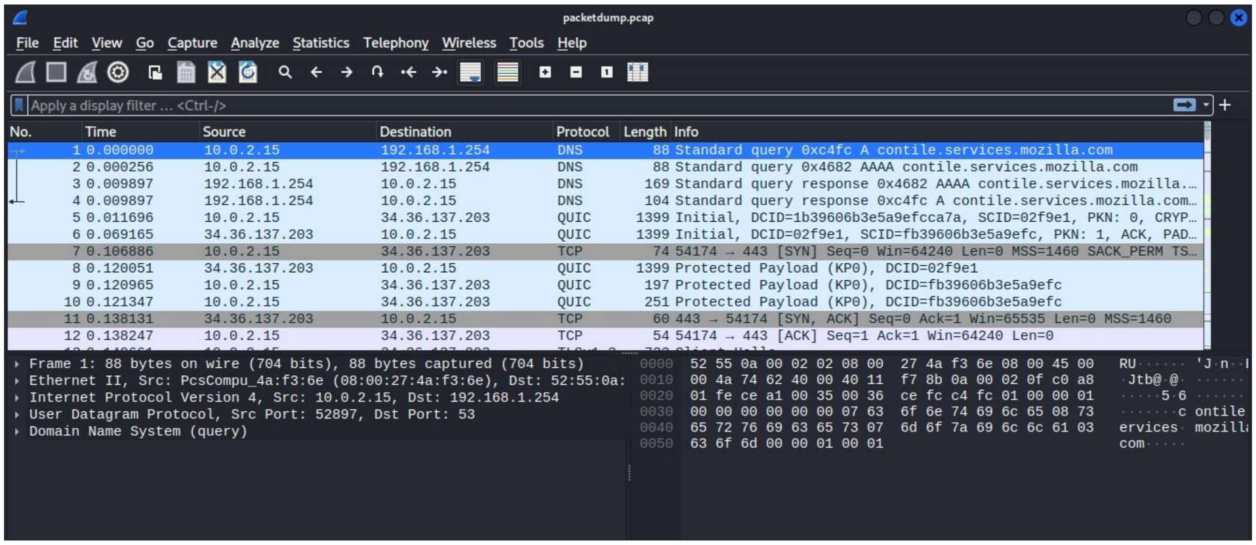


```
┌──(kali㉿Kali)-[~/Desktop]
└─$ sudo tcpdump -i eth0 -s 0 -w packetdump.pcap
[sudo] password for kali:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C4591 packets captured
4591 packets received by filter
0 packets dropped by kernel
```

   Note:
   -i command option allows we to specify the interface.
   -s command option specifies the length of the snapshot for each packet.
   -w command option is used to write the result of the **tcpdump** command to a file.
3. To capture an HTTP request and reply, open a web browser and navigate to Google.com.
4. Open a second tab in the browser, enter netacad.com
5. Return to the terminal window that is running the **tcpdump** utility and enter CTRL-C to complete the packet capture.
6. The tcpdump utility saved the output to a file named packetdump.pcap.

Part 3: View and Analyze the Packet Capture

1. Use Wireshark to view the captured packets.
2. Use the File -> Open menu option and enter packetdump.pcap in the File Name field.
3. Filter the captured traffic to only display DNS queries and responses. Enter dns in the Filter Field on the Wireshark main screen.


packetdump.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time      Source          Destination     Protocol Length Info
   1 0.000000  10.0.2.15       192.168.1.254   DNS        88 Standard query 0xc4fc A contile.services.mozilla.com
   2 0.000256  10.0.2.15       192.168.1.254   DNS        88 Standard query 0x4682 AAAA contile.services.mozilla.com
   3 0.009897  192.168.1.254   10.0.2.15       DNS       169 Standard query response 0x4682 AAAA contile.services.mozilla.…
   4 0.009897  192.168.1.254   10.0.2.15       DNS       104 Standard query response 0xc4fc A contile.services.mozilla.com…
   5 0.011696  10.0.2.15       34.36.137.203   QUIC     1399 Initial, DCID=1b39606b3e5a9efcca7a, SCID=02f9e1, PKN: 0, CRYP…
   6 0.069165  34.36.137.203   10.0.2.15       QUIC     1399 Initial, DCID=02f9e1, SCID=fb39606b3e5a9efc, PKN: 1, ACK, PAD…
   7 0.106886  10.0.2.15       34.36.137.203   TCP        74 54174 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TS…
   8 0.120051  34.36.137.203   10.0.2.15       QUIC     1399 Protected Payload (KP0), DCID=02f9e1
   9 0.120965  10.0.2.15       34.36.137.203   QUIC      197 Protected Payload (KP0), DCID=fb39606b3e5a9efc
  10 0.121347  10.0.2.15       34.36.137.203   QUIC      251 Protected Payload (KP0), DCID=fb39606b3e5a9efc
  11 0.138131  34.36.137.203   10.0.2.15       TCP        60 443 → 54174 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
  12 0.138247  10.0.2.15       34.36.137.203   TCP        54 54174 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
```

```
› Frame 1: 88 bytes on wire (704 bits), 88 bytes captured (704 bits)
› Ethernet II, Src: PcsCompu_4a:f3:6e (08:00:27:4a:f3:6e), Dst: 52:55:0a:
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 192.168.1.254
› User Datagram Protocol, Src Port: 52897, Dst Port: 53
› Domain Name System (query)
```

```
0000  52 55 0a 00 02 02 08 00  27 4a f3 6e 08 00 45 00   RU······ 'J·n··E
0010  00 4a 74 62 40 00 40 11  f7 8b 0a 00 02 0f c0 a8   ·Jtb@·@· ······
0020  01 fe ce a1 00 35 00 36  ce fc c4 fc 01 00 00 01   ·····5·6 ······
0030  00 00 00 00 00 00 07 63  6f 6e 74 69 6c 65 08 73   ·······c ontile
0040  65 72 76 69 63 65 73 07  6d 6f 7a 69 6c 6c 61 03   ervices· mozill
0050  63 6f 6d 00 00 01 00 01                            com····
```