The task is to:

- Investigate the Scapy Tool.
- Use Scapy to Sniff Network Traffic.
- Create and Send an ICMP Packet.
- Create and Send TCP SYN Packets.

Scapy is a powerful **Python-based tool and library** used for **packet crafting, manipulation, sending, sniffing, and network testing**.

It allows cybersecurity professionals, penetration testers, and network engineers to **create custom packets**, analyse traffic, and perform advanced network tasks that tools like Nmap cannot easily do.

What Scapy Can Do

1. Create and Send Packets

Scapy can craft **any type of packet**:

- IP
- TCP
- UDP
- ICMP
- ARP
- DNS
- HTTP

2. Sniff (capture) network traffic

It works like a lightweight packet sniffer (similar to Wireshark).

3. Perform Network Scans

Scapy can build custom scanning tools—ARP scans, port scans, ping sweeps, traceroute.

4. Test Firewall and IDS behavior

Because you can control every bit of a packet.

5. Automate Security Tasks

Since it's Python-based, you can script:

- Reconnaissance
- Packet injection
- Exploit testing
- Protocol fuzzing

Why Scapy Is Used

- Flexible (create any packet you want)
- Great for learning how protocols work
- Used in penetration testing and red teaming
- Good for bypassing basic security controls
- Perfect for custom network experiments

Note Scapy requires **Python** and usually **root/administrator privileges** to send low-level packet.

Enter sudo su

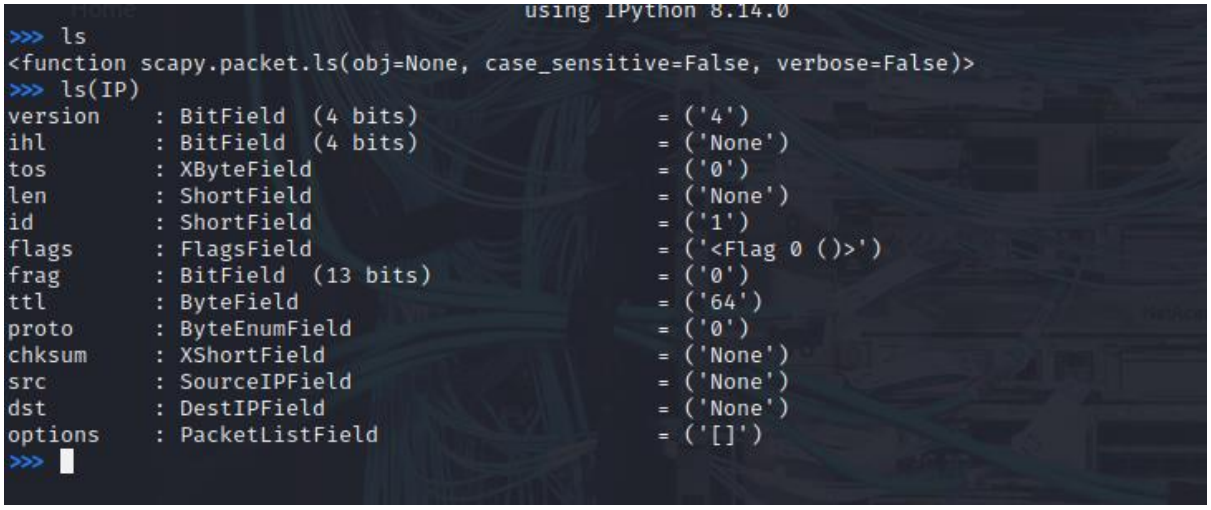
At the >>> prompt within the Scapy shell, enter the ls() function to list all of the available default formats and protocols included with the tool. The list is quite extensive and will fill multiple screens.

>>> ls()

The **ls() **function can also be used to list details of the fields and options available in each protocol header. The syntax to use a function in Scapy is ********function_name(arguments). ********Use the **ls(IP) **function to list the available fields in an IP packet header.

ls(IP)

Output



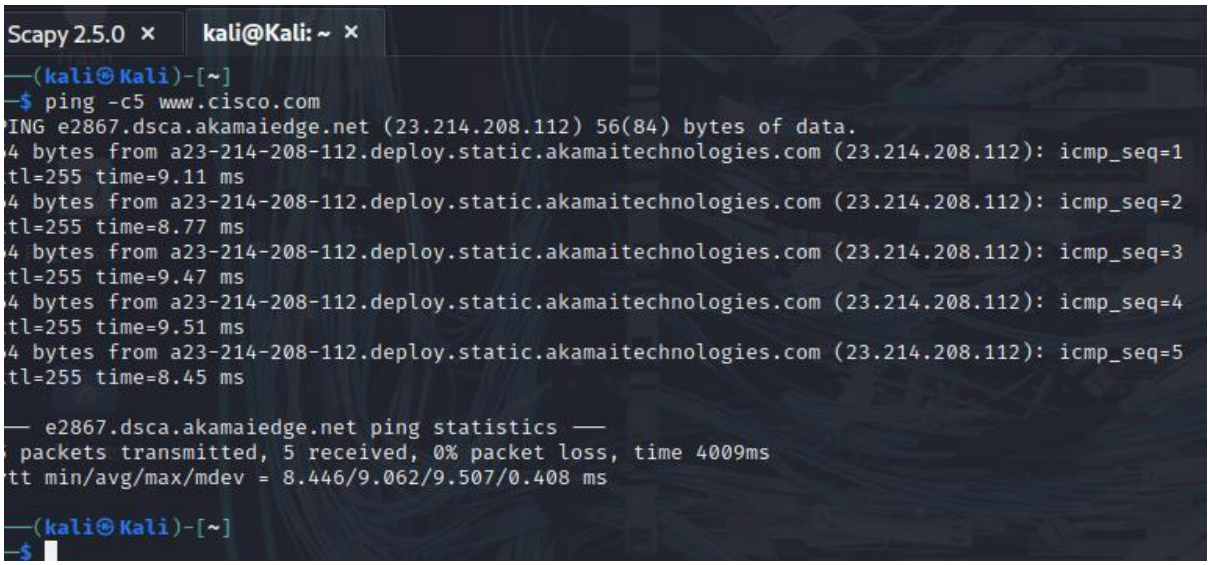
Use the sniff() function Use the **sniff() **function to collect traffic using the default eth0 interface of your VM. Start the capture with the **sniff() **function without specifying any arguments. {#use-the-sniff-function-to-collect-traffic-using-the-default-eth0-interface-of-your-vm.-start-the-capture-with-the-sniff-function-without-specifying-any-arguments.}

sniff()

THEN

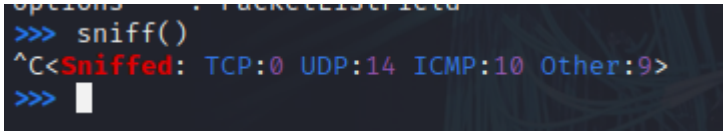
****Open a second terminal window and **ping **an internet address, such as www.cisco.com. Remember to specify the count using the -c argument

output



Return to the terminal window that is running the Scapy tool. Press CTRL-C to stop the capture. You should receive output similar to what is shown here:

output



View the captured traffic using the summary() function. The a=_ 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.
>>> a=_

>>> a.summary()

output

```
>>> sniff()
^C<Sniffed: TCP:0 UDP:14 ICMP:10 Other:9>
>>> a=_
>>> a.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 > 23.214.208.112 echo-request 0 / Raw
Ether / IP / ICMP 23.214.208.112 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'112.208.214.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-214-208-112.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 23.214.208.112 echo-request 0 / Raw
Ether / IP / ICMP 23.214.208.112 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'112.208.214.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-214-208-112.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 23.214.208.112 echo-request 0 / Raw
Ether / IP / ICMP 23.214.208.112 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'112.208.214.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-214-208-112.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 23.214.208.112 echo-request 0 / Raw
Ether / IP / ICMP 23.214.208.112 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'112.208.214.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-214-208-112.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 23.214.208.112 echo-request 0 / Raw
Ether / IP / ICMP 23.214.208.112 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'112.208.214.23.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a23-214-208-112.deploy.static.akamaitechnologies.com.'"
Ether / ARP who has 10.0.2.3 says 10.0.2.15
Ether / ARP who has 10.0.2.2 says 10.0.2.15
Ether / ARP is at 52:55:0a:00:02:03 says 10.0.2.3 / Padding
Ether / ARP is at 52:55:0a:00:02:02 says 10.0.2.2 / Padding
Ether / fe80::a00:27ff:fe4a:f36e > ff02::2 (58) / ICMPv6ND_RS
Ether / IPv6 / ICMPv6ND_RA / ICMPv6NDOptSrcLLAddr / ICMPv6NDOptPrefixInfo / ICMPv6 Neighbor Disc
overy Option - Recursive DNS Server Option fd00::3
Ether / IPv6 / ICMPv6ND_RA / ICMPv6NDOptSrcLLAddr / ICMPv6NDOptPrefixInfo / ICMPv6 Neighbor Disc
overy Option - Recursive DNS Server Option fd00::3
Ether / fe80::a00:27ff:fe4a:f36e > ff02::16 (0) / IPv6ExtHdrHopByHop / ICMPv6MLReport2
Ether / fe80::a00:27ff:fe4a:f36e > ff02::16 (0) / IPv6ExtHdrHopByHop / ICMPv6MLReport2
>>>
```

Create and Send an ICMP Packet.

Sniff Filters

To collect traffic to only include ICMP traffic, limit the number of packets being collected, and view the individual packet details.

Use interface ID associated with 10.6.6.1 (br-internal) to capture ten ICMP packets sent and received on the internal virtual network. The syntax is sniff(iface="interface name", filter = "protocol", count = integer).

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

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

```
>>> sniff(iface="br-internal",filter ="icmp", count=10)
<Sniffed: TCP:0 UDP:0 ICMP:10 Other:0>
>>> 
```

```
┌──(kali㉿Kali)-[~]
└─$ ping -c10 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=0.143 ms
64 bytes from 10.6.6.23: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 10.6.6.23: icmp_seq=3 ttl=64 time=0.095 ms
64 bytes from 10.6.6.23: icmp_seq=4 ttl=64 time=0.127 ms
64 bytes from 10.6.6.23: icmp_seq=5 ttl=64 time=0.168 ms
64 bytes from 10.6.6.23: icmp_seq=6 ttl=64 time=0.143 ms
64 bytes from 10.6.6.23: icmp_seq=7 ttl=64 time=0.040 ms
64 bytes from 10.6.6.23: icmp_seq=8 ttl=64 time=0.044 ms
64 bytes from 10.6.6.23: icmp_seq=9 ttl=64 time=0.045 ms
64 bytes from 10.6.6.23: icmp_seq=10 ttl=64 time=0.047 ms

--- 10.6.6.23 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9166ms
rtt min/avg/max/mdev = 0.040/0.091/0.168/0.047 ms
```

View the captured traffic with line numbers using the **nsummary() **function.

>>> a=_

>>> a.summary()

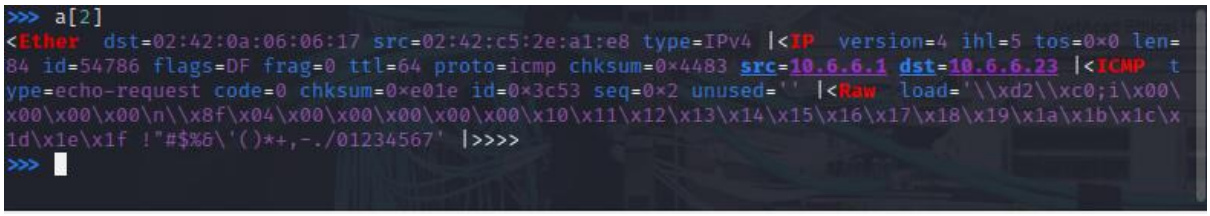
output

```
>>> sniff(iface="br-internal",filter ="icmp", count=10)
<Sniffed: TCP:0 UDP:0 ICMP:10 Other:0>
>>> a=_
>>> a.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
>>> 
```

To view details about a specific packet in the series. USE
>>> a[2]

The detail output shows the layers of information about the protocol data units (PDUs) that make up the packet. The protocol layer names appear in red in the output. Examine the source (src) and destination (dst) addresses as well as the raw data (load=) portion of the collected packet.

output



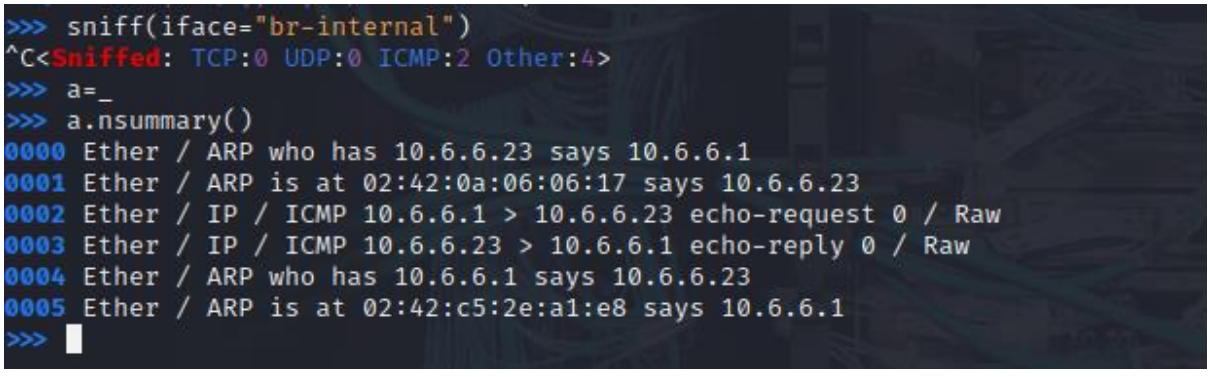
Use the wrpcap() function to save the captured data to a pcap file that can be opened by Wireshark and other applications. The syntax is wrpcap(“filename.pcap", variable name), in this example the variable that you stored the output is “a".
>>> wrpcap(“capture1.pcap", a)

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")

command to sniff traffic from the interface connected to the 10.6.6.0/24 network.

output



Start a second instance of Scapy. Enter the **send**() function to send a packet to 10.6.6.23 with a modified ICMP payload.

output

Create and Send a TCP SYN Packet.

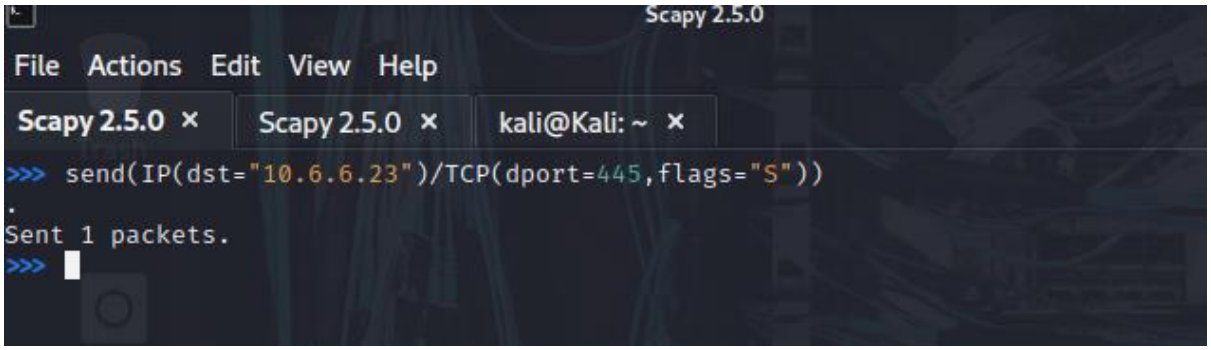
In this part, you will use Scapy to determine if port 445, a Microsoft Windows drive share port, is open on the target system at 10.6.6.23.

Start the packet capture on the internal interface.

a. In the original Scapy terminal window, begin a packet capture on the internal interface attached to the 10.6.6.0/24 network. Use the interface name that you obtained previously.

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

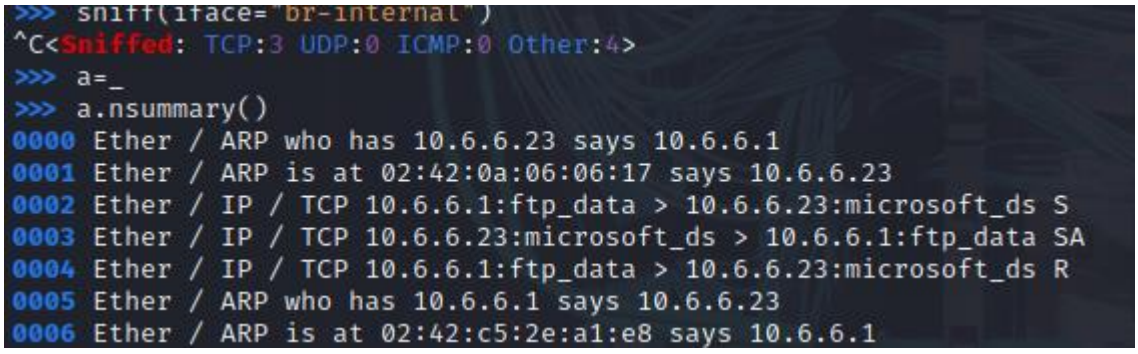
```
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
```



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.

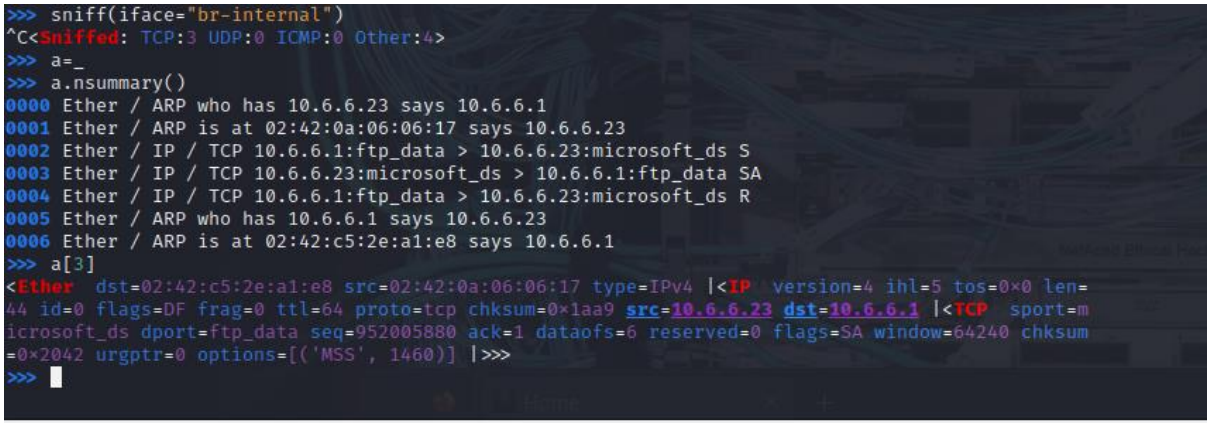
```
>>> sniff(iface="br-internal")
```

Review the captured packets. {#review-the-captured-packets.}



Send Tcp packet

Review the captured packets


>>> sniff(iface="br-internal")
^C<Sniffed: TCP:3 UDP:0 ICMP:0 Other:4>
>>> a=_
>>> a.nsummary()
0000 Ether / ARP who has 10.6.6.23 says 10.6.6.1
0001 Ether / ARP is at 02:42:0a:06:06:17 says 10.6.6.23
0002 Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds S
0003 Ether / IP / TCP 10.6.6.23:microsoft_ds > 10.6.6.1:ftp_data SA
0004 Ether / IP / TCP 10.6.6.1:ftp_data > 10.6.6.23:microsoft_ds R
0005 Ether / ARP who has 10.6.6.1 says 10.6.6.23
0006 Ether / ARP is at 02:42:c5:2e:a1:e8 says 10.6.6.1
>>> a[3]
<Ether dst=02:42:c5:2e:a1:e8 src=02:42:0a:06:06:17 type=IPv4 |<IP version=4 ihl=5 tos=0x0 len=
44 id=0 flags=DF frag=0 ttl=64 proto=tcp chksum=0x1aa9 src=10.6.6.23 dst=10.6.6.1 |<TCP sport=m
icrosoft_ds dport=ftp_data seq=952005880 ack=1 dataofs=6 reserved=0 flags=SA window=64240 chksum
=0x2042 urgptr=0 options=[('MSS', 1460)] |>>>
>>>