Packet Crafting with Scapy

Objectives

In this lab, you 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.

Background / Scenario

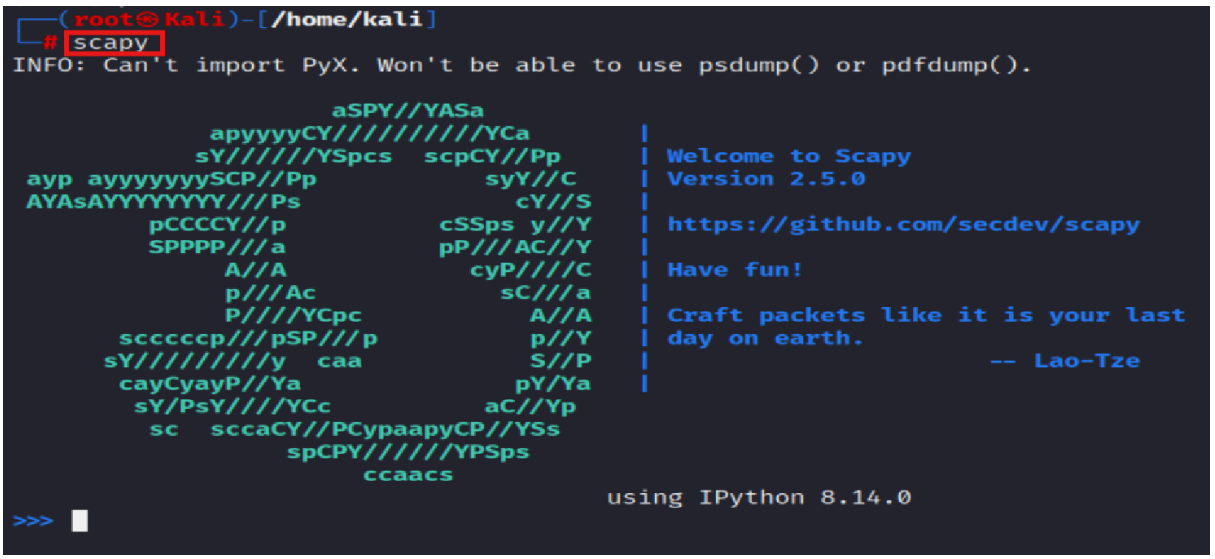
Penetration testers and ethical hackers often use specially-crafted packets to discover and/or exploit vulnerabilities in clients' infrastructure and systems. You have used Nmap to scan and analyze vulnerabilities in a computer attached to the local network. In this lab, you will perform further reconnaissance on the same computer using custom ICMP and TCP packets.

Part 1: Investigate the Scapy Tool

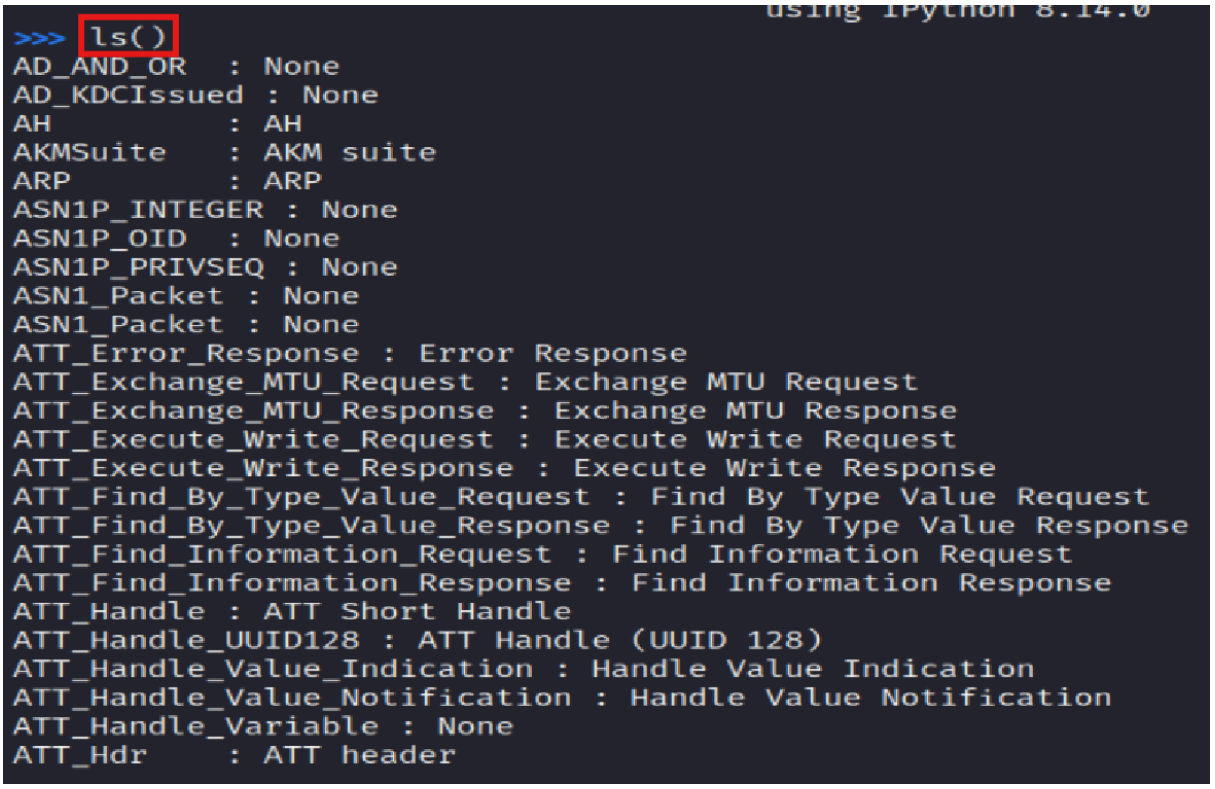
In this part, you will load the Scapy tool and explore some of its capabilities. Tools like Scapy should only be used to scan or communicate with machines that you own or have written permission to access. **Scapy** is a Python program that enables the user to **send, sniff, dissect and forge network packets**. This capability allows **construction of tools** that can **probe, scan or attack networks**.

Use Scapy interactive command mode.

Enter the **scapy** command in a terminal window to load the Python interpreter. By using this command, the interpreter runs pre-loaded with the Scapy classes and objects. The commands to craft and send packets require root privileges to run. Use the **sudo su** command to obtain root privileges before starting Scapy. Load the Scapy tool using the **scapy** command. The interactive Python interpreter will load and present a screen image similar to that shown.



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.

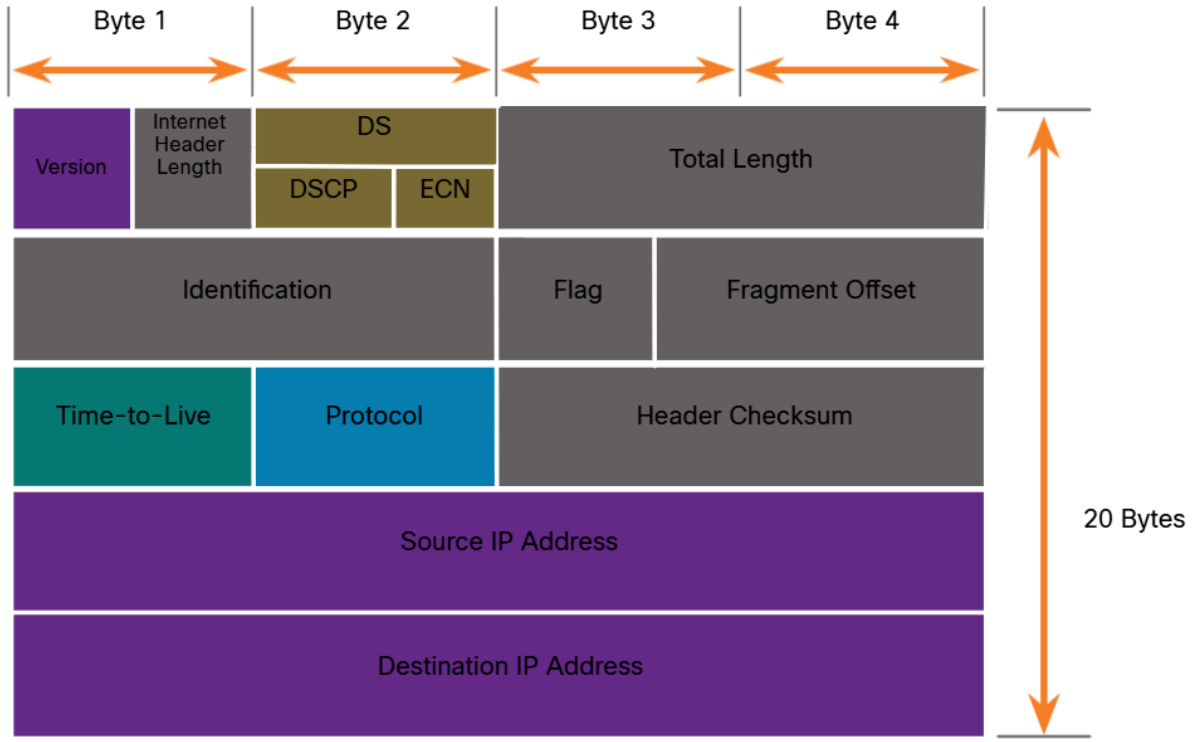


Examine the fields in an IPv4 packet header.

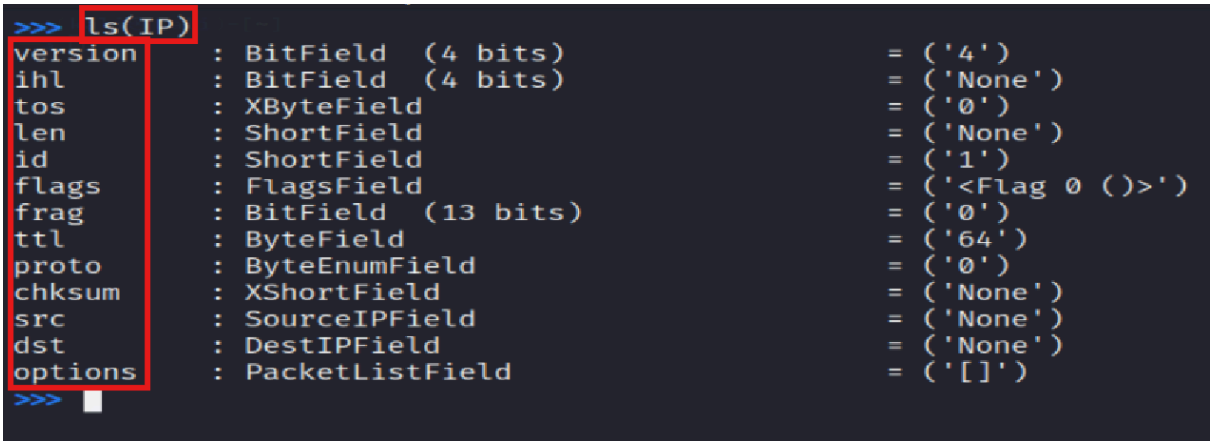
It is important to understand the structure of an IP packet before creating and sending custom packets over the network. Each IP packet has an associated header that provides information about the structure of the packet. Review this information before continuing with the lab.

Protocol header diagrams, which are read left to right, and top down, provide a visual to refer to when discussing protocol fields. The IP protocol header diagram in the figure identifies the fields of an IPv4 packet.

Fields in the IPv4 Packet Header



The **Is()** function can also be used to list details of the fields and options available in each protocol header. Use the **Is(IP)** function to list the available fields in an IP packet header.



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

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.
- Open a second terminal window and **ping** an internet address, such as **www.cisco.com**. Remember to specify the count using the **-c** argument.
- Return to the terminal window that is running the Scapy tool. Press **CTRL-C** to stop the capture.
- 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.

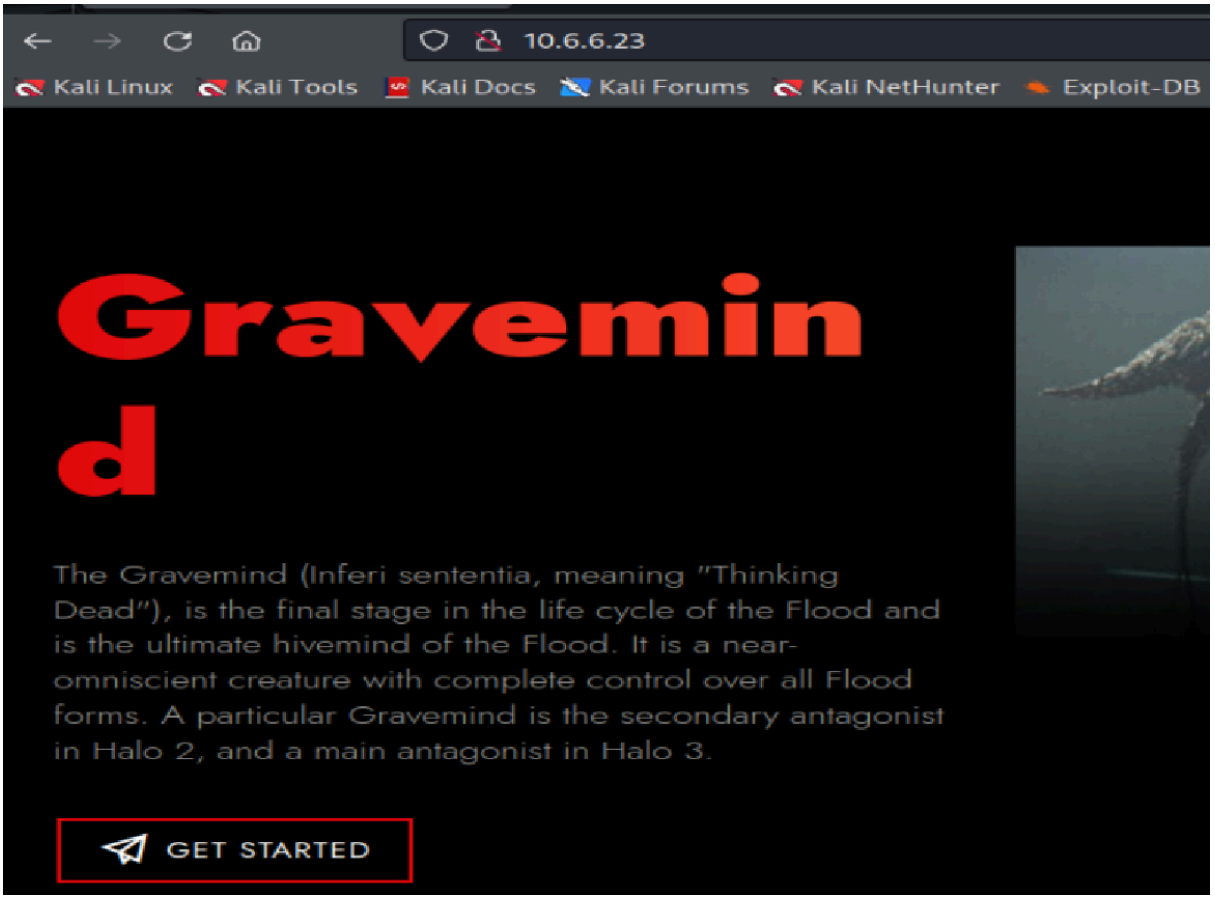
```
┌──(kali㉿Kali)-[~]
└─$ ping -c 5 www.cisco.com
PING e2867.dsca.akamaiedge.net (2.17.168.94) 56(84) bytes of data.
64 bytes from a2-17-168-94.deploy.static.akamaitechnologies.com (2.17.168.94): icmp_seq=1 ttl=255 t
ime=13.4 ms
64 bytes from a2-17-168-94.deploy.static.akamaitechnologies.com (2.17.168.94): icmp_seq=2 ttl=255 t
ime=14.6 ms
64 bytes from a2-17-168-94.deploy.static.akamaitechnologies.com (2.17.168.94): icmp_seq=3 ttl=255 t
ime=14.1 ms
64 bytes from a2-17-168-94.deploy.static.akamaitechnologies.com (2.17.168.94): icmp_seq=4 ttl=255 t
ime=16.6 ms
64 bytes from a2-17-168-94.deploy.static.akamaitechnologies.com (2.17.168.94): icmp_seq=5 ttl=255 t
ime=14.0 ms

--- e2867.dsca.akamaiedge.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 13.409/14.566/16.634/1.104 ms
```

```
>>> sniff()
^C<Sniffed: TCP:4 UDP:14 ICMP:10 Other:2>
>>> a=_
>>> a.summary()
Ether / IP / TCP 34.107.243.93:https > 10.0.2.15:32948 PA / Raw
Ether / IP / TCP 10.0.2.15:32948 > 34.107.243.93:https A
Ether / IP / TCP 10.0.2.15:32948 > 34.107.243.93:https PA / Raw
Ether / IP / TCP 34.107.243.93:https > 10.0.2.15:32948 A / Padding
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 > 2.17.168.94 echo-request 0 / Raw
Ether / IP / ICMP 2.17.168.94 > 10.0.2.15 echo-reply 0 / Raw
Ether / IP / UDP / DNS Qry "b'94.168.17.2.in-addr.arpa.'"
Ether / IP / UDP / DNS Ans "b'a2-17-168-94.deploy.static.akamaitechnologies.com.'"
Ether / IP / ICMP 10.0.2.15 > 2.17.168.94 echo-request 0 / Raw
Ether / IP / ICMP 2.17.168.94 > 10.0.2.15 echo-reply 0 / Raw
```

Capture and save traffic on a specific interface.

- Open 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.
- 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**.
- View the captured traffic




```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:171 UDP:0 ICMP:0 Other:6>
>>> a=_
>>> a.summary()
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http S
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 SA
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 PA / Raw
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 PA / Raw
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 PA / Raw
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http A
Ether / IP / TCP 10.6.6.1:46360 > 10.6.6.23:http PA / Raw
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 A
Ether / IP / TCP 10.6.6.23:http > 10.6.6.1:46360 PA / Raw
```

Examine the collected packets.

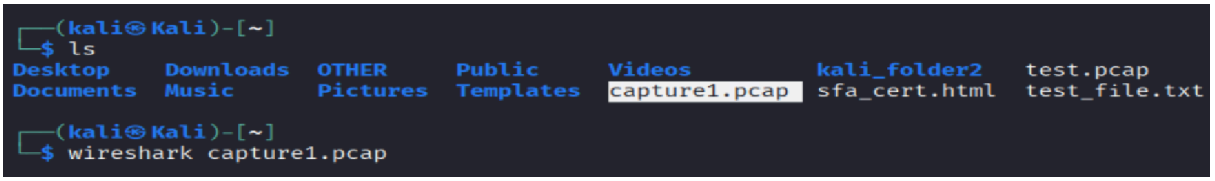
In this step, you will filter the collected traffic to include only ICMP traffic, limit the number of packets being collected, and view the individual packet details.

- 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***)**.
- Open a second terminal window and ping the host at 10.6.6.23.
- Return to the terminal window running the Scapy tool. The capture automatically stopped when 10 packets were sent or received. View the captured traffic using the **summary()** function.
- 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**".
- The .pcap file will be written to the default user directory. Use a different terminal window to verify the location of the **capture1.pcap** file using the Linux **ls** command.
- Open the capture in Wireshark to view the file contents.

```
┌──(kali㉿Kali)-[~]
└─$ ping -c 5 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=4.74 ms
64 bytes from 10.6.6.23: icmp_seq=2 ttl=64 time=1.23 ms
64 bytes from 10.6.6.23: icmp_seq=3 ttl=64 time=0.148 ms
64 bytes from 10.6.6.23: icmp_seq=4 ttl=64 time=0.149 ms
64 bytes from 10.6.6.23: icmp_seq=5 ttl=64 time=0.146 ms

--- 10.6.6.23 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4042ms
rtt min/avg/max/mdev = 0.146/1.283/4.739/1.778 ms
```

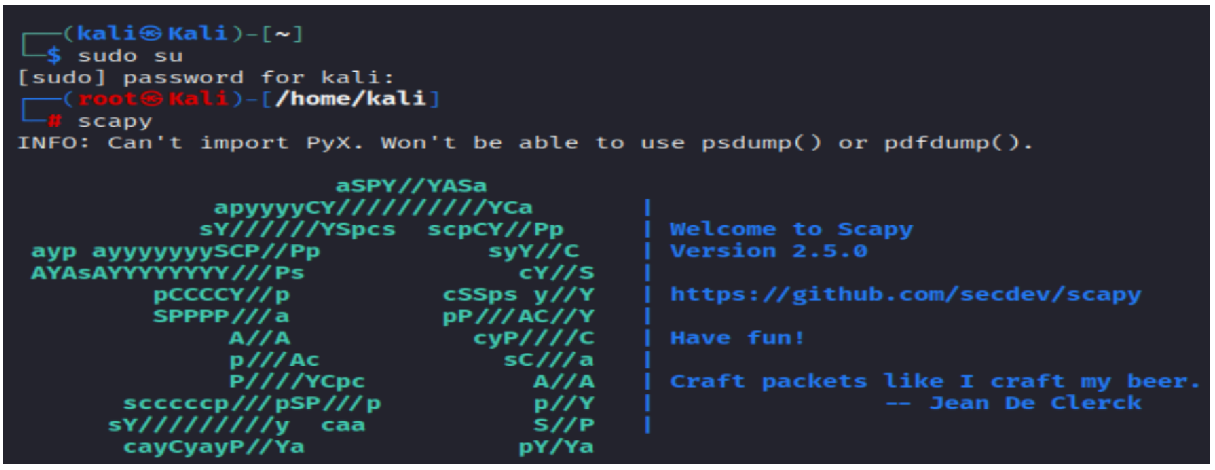
```
>>> sniff(iface="br-internal", filter = "icmp", count = 10)
<Sniffed: TCP:0 UDP:0 ICMP:10 Other:0>
>>> a=_
>>> a.summary()
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
Ether / IP / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 /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

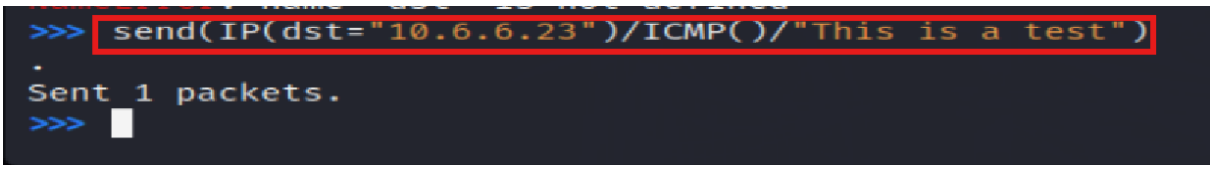


Part 3: Create and Send an ICMP Packet.

ICMP is a protocol designed to send control messages between network devices for various purposes. There are many types of ICMP packets, with echo-request and echo-reply the most familiar to IT technicians.

- In a Scapy terminal window, enter the command to sniff traffic from the interface connected to the 10.6.6.0/24 network.
- 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.
- Return to the first terminal window and press **CTRL-C**.
- Enter the summary command to display the summary with packet numbers.
- Use the packet numbers to view the individual ICMP Echo-request and Echo-reply packets.





```
>>> sniff(iface="br-internal")
^C<Sniffed: TCP:0 UDP:0 ICMP:2 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 / ICMP 10.6.6.1 > 10.6.6.23 echo-request 0 / Raw
0003 Ether / IP / ICMP 10.6.6.23 > 10.6.6.1 echo-reply 0 / Raw
0004 Ether / ARP who has 10.6.6.1 says 10.6.6.23
0005 Ether / ARP is at 02:42:e6:c6:3d:ac says 10.6.6.1
>>>
```

```
>>> a[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:e6:c6:3d:ac type=IPv4 |<IP  version=4 ihl=5 t
os=0x0 len=42 id=1 flags= frag=0 ttl=64 proto=icmp chksum=0x5aaf src=10.6.6.1 dst=10.
6.6.23 |<ICMP  type=echo-request code=0 chksum=0x5da0 id=0x0 seq=0x0 unused='' |<Raw
 load='This is a test' |>>>>
>>> a[3]
<Ether  dst=02:42:e6:c6:3d:ac src=02:42:0a:06:06:17 type=IPv4 |<IP  version=4 ihl=5 t
os=0x0 len=42 id=3886 flags= frag=0 ttl=64 proto=icmp chksum=0x4b82 src=10.6.6.23 dst
=10.6.6.1 |<ICMP  type=echo-reply code=0 chksum=0x65a0 id=0x0 seq=0x0 unused='' |<Raw
  load='This is a test' |>>>>
>>>
```

Part 4: 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.

- In the original Scapy terminal window, begin a packet capture on the internal interface attached to the 10.6.6.0/24 network.
- Navigate to the second terminal window. Create and send a TCP SYN packet
- Close the terminal window.
- In the original Scapy terminal window, stop the packet capture by pressing **CTRL-C**.
- View the captured TCP packets using the **nsummary()** function. Display the detail of the TCP packet that was returned from the target computer at 10.6.6.23.

```
>>> send(IP(dst="10.6.6.23")/TCP(dport=445, flags="S"))
.
Sent 1 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:e6:c6:3d:ac says 10.6.6.1
```

```
>>> a[2]
<Ether  dst=02:42:0a:06:06:17 src=02:42:e6:c6:3d:ac type=IPv4 |<IP  version=4 ihl=5 t
os=0x0 len=40 id=1 flags= frag=0 ttl=64 proto=tcp chksum=0x5aac src=10.6.6.1 dst=10.6
.6.23 |<TCP  sport=ftp_data dport=microsoft_ds seq=0 ack=0 dataofs=5 reserved=0 flags
=S window=8192 chksum=0x6dee urgptr=0 |>>>
>>> a[3]
<Ether  dst=02:42:e6:c6:3d:ac src=02:42:0a:06:06:17 type=IPv4 |<IP  version=4 ihl=5 t
os=0x0 len=44 id=0 flags=DF frag=0 ttl=64 proto=tcp chksum=0x1aa9 src=10.6.6.23 dst=1
0.6.6.1 |<TCP  sport=microsoft_ds dport=ftp_data seq=3607247616 ack=1 dataofs=6 reser
ved=0 flags=SA window=64240 chksum=0x2042 urgptr=0 options=[('MSS', 1460)] |>>>
>>>
```