

ETHICAL HACKING

LAB ASSIGNMENT -5

Name: Dasari Sanath Kumar

ID: 700760349

CRN: 22285

Lab Report and detailed analysis of *scapy*:

1) **Scapy**: -

Scapy is a robust Python library and tool for interactive packet manipulation. It lets users to create, alter, send, and capture network packets. “Scapy” is a Python tool that is frequently used for a variety of network analysis, testing, and exploitation tasks by researchers, network engineers, and security experts.

Packet Crafting: Scapy enables low-level packet creation and manipulation. Custom packet structures, different packet attributes (such source and destination addresses, ports, protocols, etc.), and packet payload modifications are all available to users.

Sending Packets: Scapy has the ability to send specially constructed packets across a network. Sending packets to particular hosts, ports, or whole networks is possible for users. Ethernet, IP, TCP, UDP, ICMP, DNS, DHCP, HTTP, and many more protocols are supported by Scapy.

Scapy's packet sniffing, and capture: This feature enables users to intercept and record packets transmitted over a network. It can read packet capture files (such as pcap files) and capture live packets from network interfaces. Users can carry out several network analysis activities, extract information, and examine captured packets.

Network Discovery and Exploration: *Scapy* can be used for network discovery and exploration tasks. Users can send probes, scan network hosts and ports, detect active hosts, identify network services, and map network topologies.

Network Testing and Fuzzing: Scapy is often used for network testing, stress testing, and fuzzing purposes. Users can send malformed or invalid packets to test the robustness and security of network devices and applications.

Integration and Extensibility: Scapy can be easily integrated into Python scripts and programs. It provides a flexible and extensible architecture, allowing users to create custom tools and utilities for various network-related tasks.

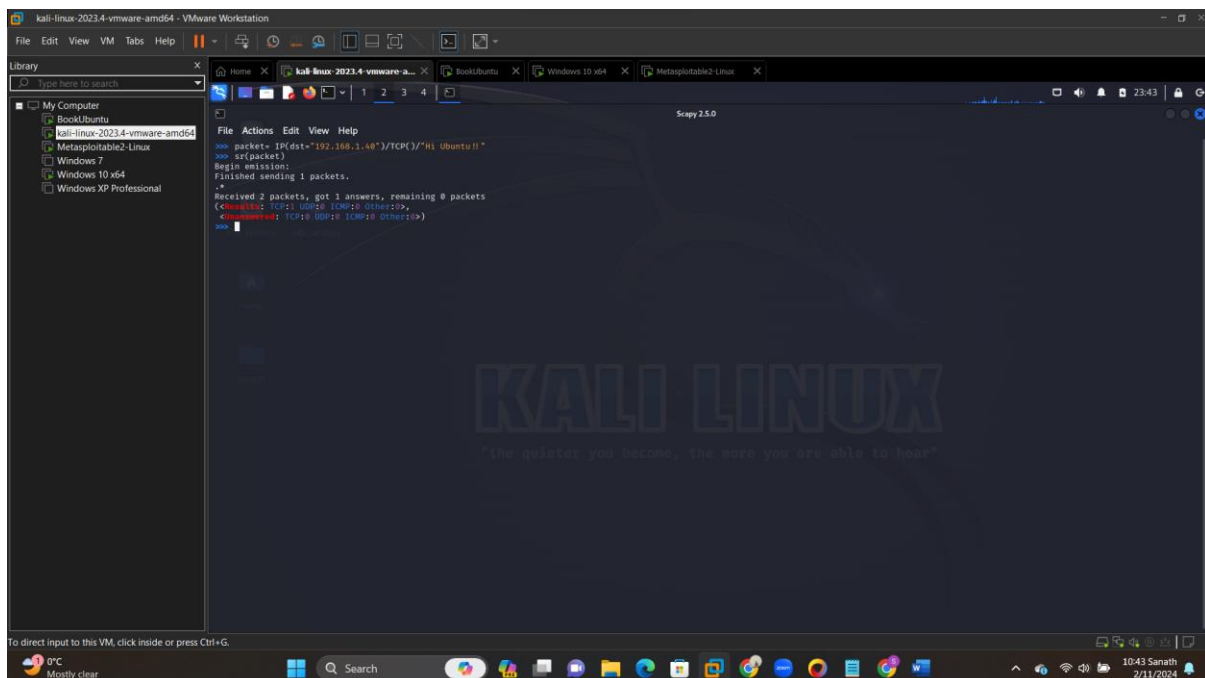
These are the main features of scapy, and Scapy is a flexible tool for network analysis, network protocol development, security research, and pen-testing.

tcpdump: - On Unix-like operating systems, tcpdump is a command-line packet analyzer program. It enables users to record and view network traffic in real-time for study. Network administrators, security experts, and researchers frequently utilize tcpdump for a variety of purposes, including protocol development, security analysis, network troubleshooting, and monitoring.

2) Now, let's construct a packet using the default settings of TCP and a payload of "Hi Ubuntu!!" using tcpdump

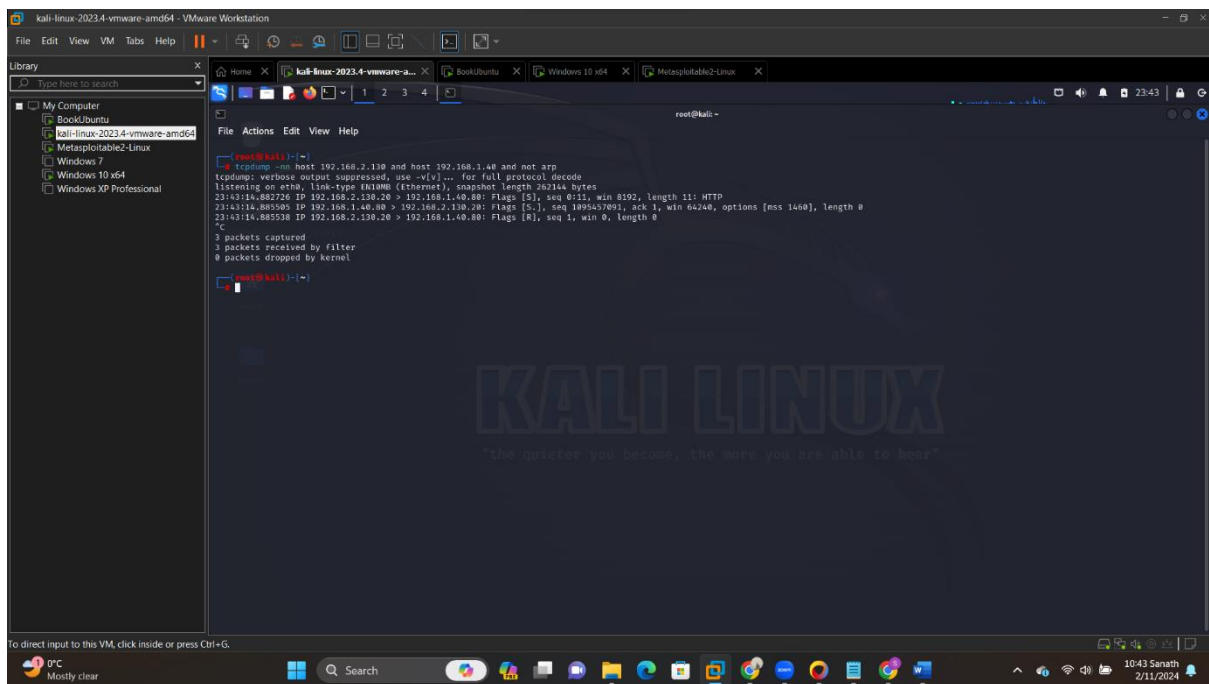
```
>>> packet= IP(dst="Ubuntu IP-Address")/TCP()/"Hi Ubuntu!!"
```

```
>>> sr(packet)
```



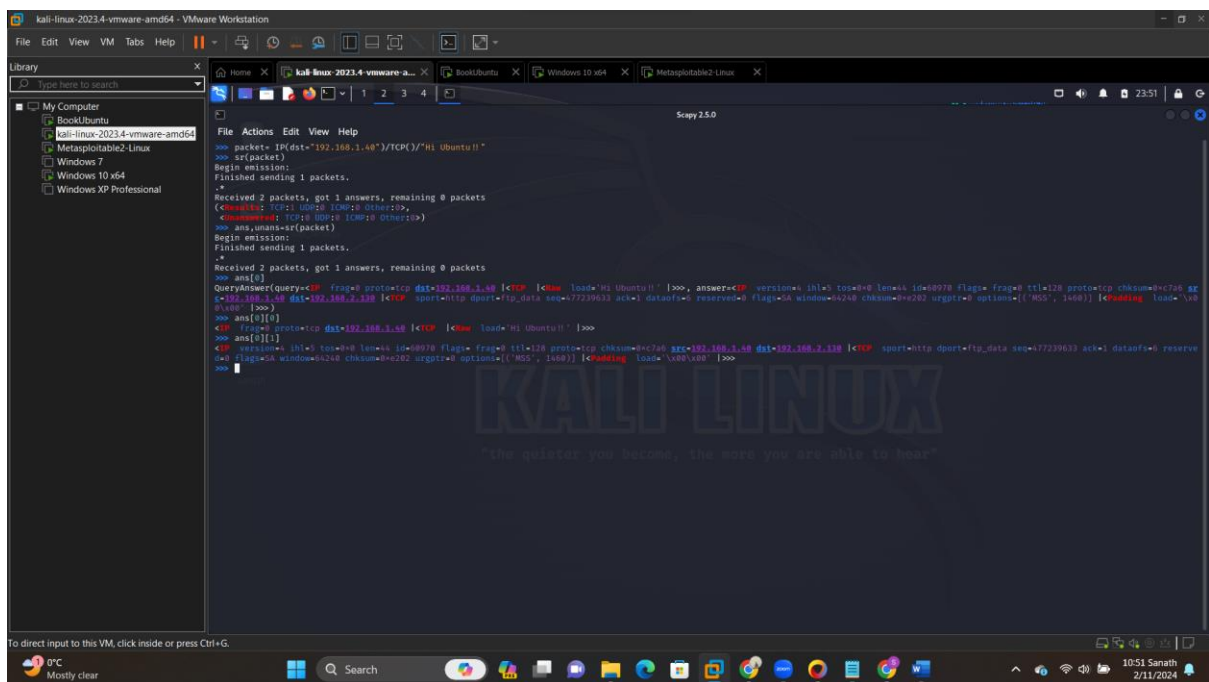
```
kali-linux-2023.4-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
  BookBuntu
  kali-linux-2023.4-vmware-amd64
  Metasploitable2-Linux
  Windows 7
  Windows 10 x64
  Windows XP Professional
kali-linux-2023.4-vmware-amd64
Scapy 2.5.0
File Actions Edit View Help
>>> packet= IP(dst="192.168.1.48")/TCP()/"Hi Ubuntu!!"
>>> sr(packet)
Begin emission:
Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
(<Header: TCP:1.00010.10010.00010.00010>,
<Checksum: TCP:1.00010.10010.00010>)
>>>
```

3)



Here in the below screenshot we can clearly see the payload = "Hi Ubuntu"

And source and destination IP



ans[0] - It shows the packet that Ubuntu received from Kali and the packet that Kali sends to Ubuntu.

ans[0][0] - want to display the packet Kali sent to Ubuntu

ans[0][1] - To display the packet received from Ubuntu

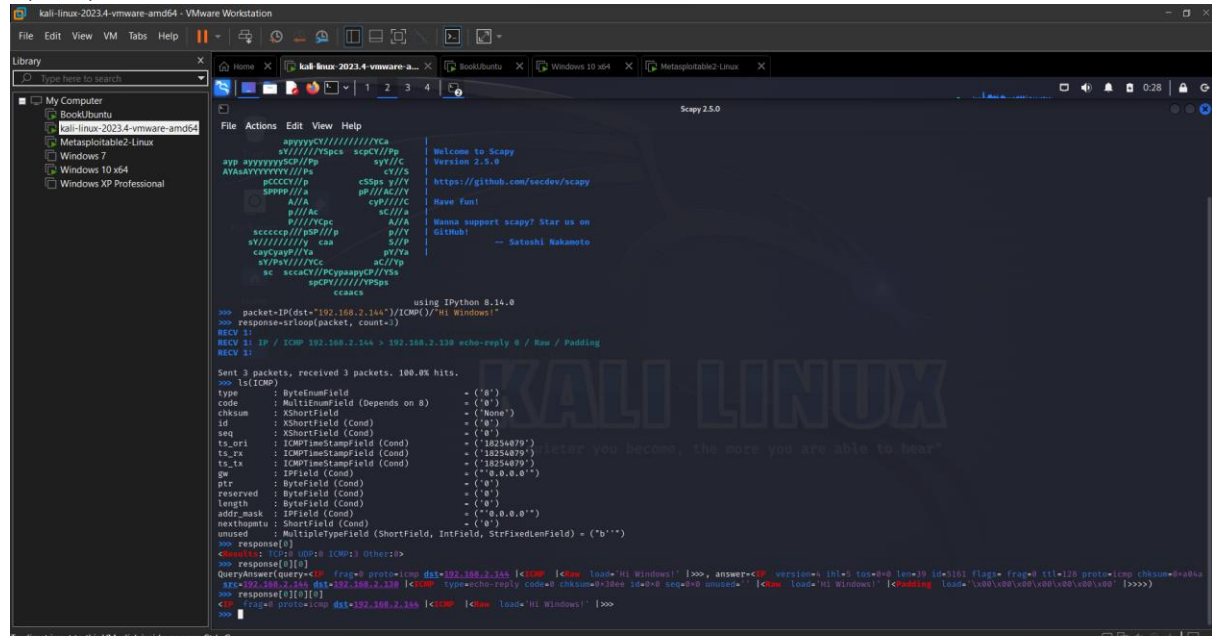
4) Now let's ping our windows-10 machine using scapy

```
>>> packet=IP(dst="Windows 10 IP_Address")/ICMP()/ "Hi Windows!"
```

```
>>> response=srloop(packet, count=3)
```

Here we used ICMP -Internet control message protocol to ping our windows-10 machine, for detailed information please check the below screenshots

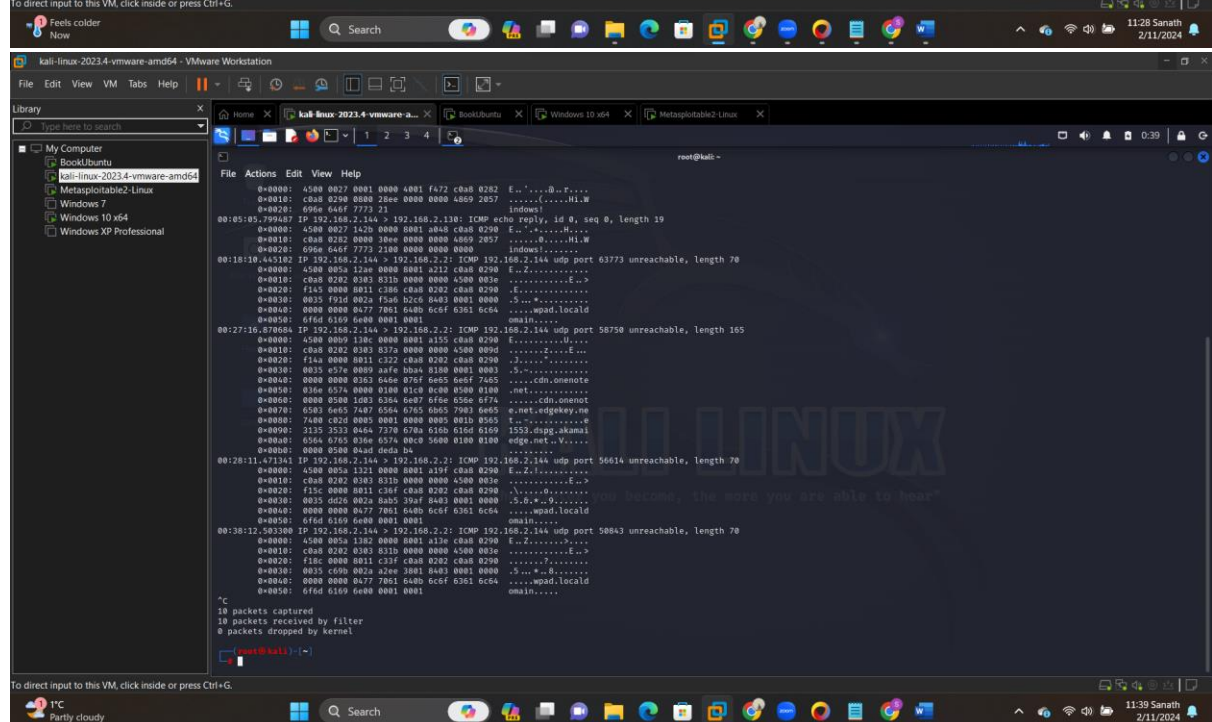
5)&6)



```
kali-linux-2023.4-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
BookLibuntu
kali-linux-2023.4-vmware-amd64
MetasploitTable2-Linux
Windows 7
Windows 10 x64
Windows XP Professional

Scapy 2.5.0
File Actions Edit View Help
Welcome to Scapy
Version 2.5.0
https://github.com/secdev/scapy
Have fun!
Mama support scapy? Star us on GitHub!
-- Satoshi Nakamoto

using IPython 8.14.0
>>> packet=IP(dst="192.168.2.144")/ICMP()/ "Hi Windows!"
>>> response=srloop(packet, count=3)
RECV 1: IP / ICMP 192.168.2.144 > 192.168.2.130 echo-reply 0 / Raw / Padding
RECV 2:
Sent 3 packets, received 3 packets. 100.0% hits.
>>> ls(ICMP)
type : ByteEnumField
code : MultiEnumField (Depends on 0)
chksum : XShortField (Cond)
id : XShortField (Cond)
seq : XShortField (Cond)
ts_ori : ICMPTimeStampField (Cond)
ts_rx : ICMPTimeStampField (Cond)
ts_tx : ICMPTimeStampField (Cond)
gw : IPField (Cond)
ptr : ByteField (Cond)
reserved : ByteField (Cond)
length : ByteField (Cond)
addr_mask : IPField (Cond)
nexthopmtu : ShortField (Cond)
unused : MultipleTypeField (ShortField, IntField, StrFixedlenField) = ('b'')
>>> response[0]
<<<Raw: TCP: UDP: ICMP: Other:>
QueryAnswer(query=ICMP, frag=0 proto=icmp dst=192.168.2.144 <ICMP Echo load='Hi Windows!' [seq=0, version=1, ttl=64, len=30, id=5161, flags=0, frag=0, ttl=64, proto=icmp, chksum=0x8ba, src=192.168.2.130, dst=192.168.2.130] <ICMP type=echo-reply code=0 chksum=0x3dee id=0, seq=0, unused='' <Raw load='Hi Windows!' <Padding load='\x00\x00\x00\x00\x00\x00' [seq=0]
>>> response[0][0]
<<Raw proto=icmp dst=192.168.2.144 <ICMP Echo load='Hi Windows!' [seq=0]
>>>
```



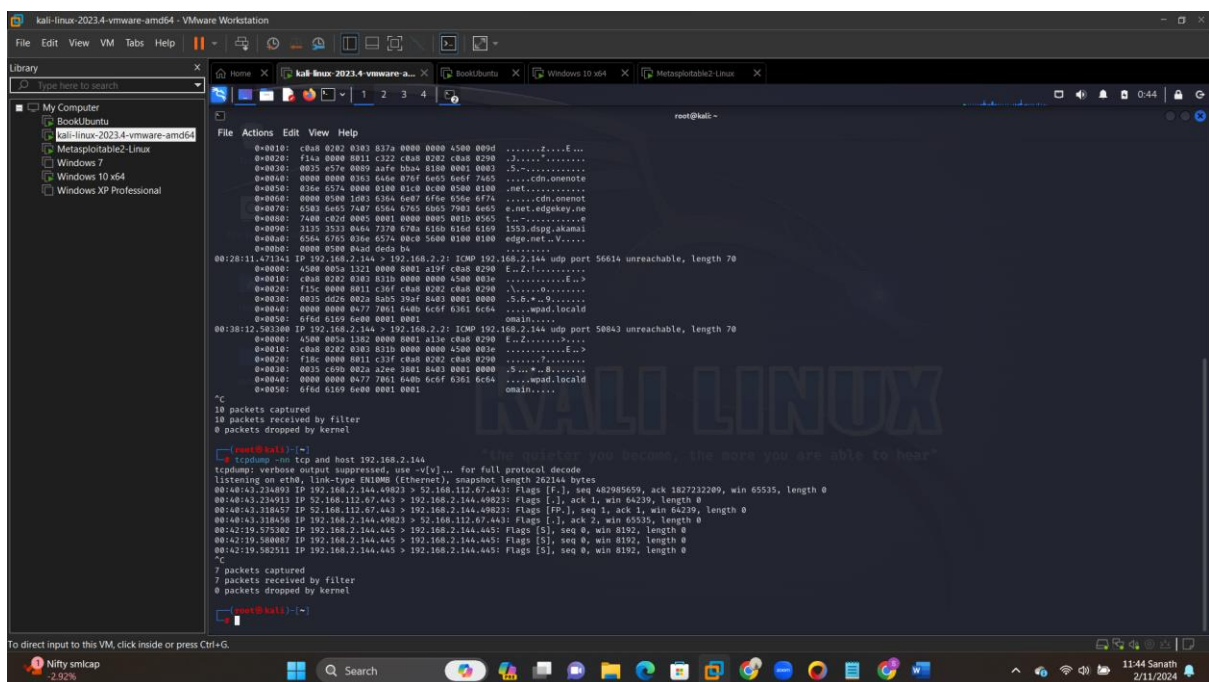
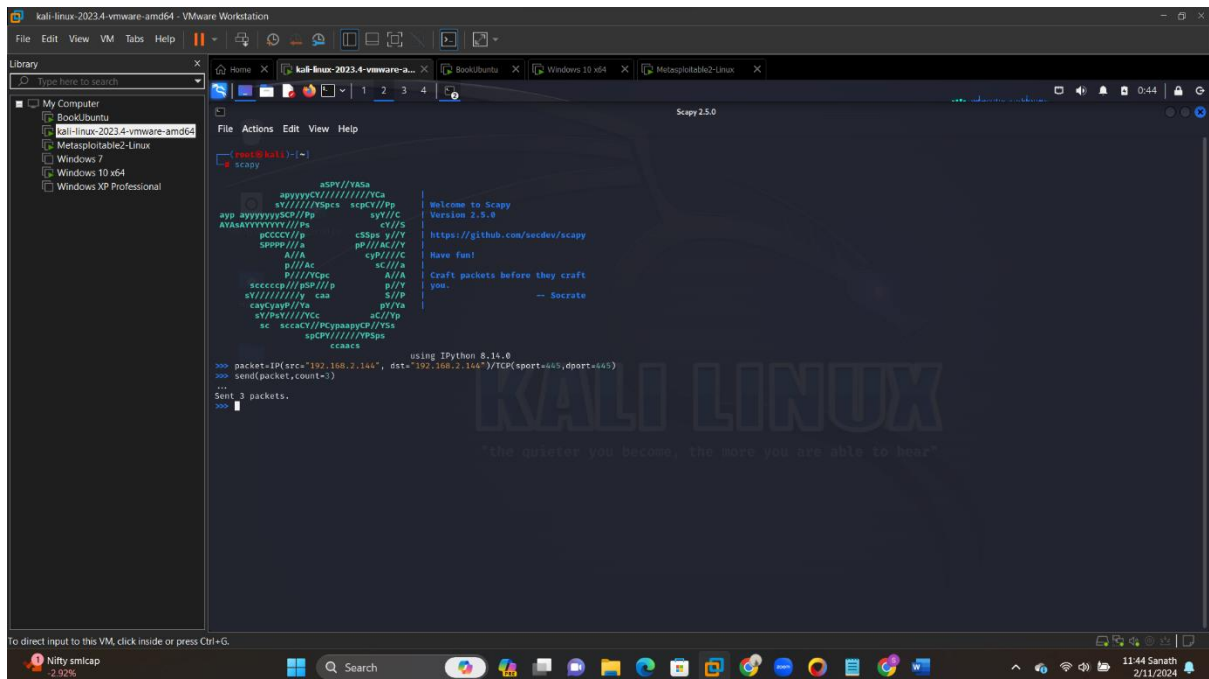
```
kali-linux-2023.4-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
BookLibuntu
kali-linux-2023.4-vmware-amd64
MetasploitTable2-Linux
Windows 7
Windows 10 x64
Windows XP Professional

root@kali:~# ls(ICMP)
type : ByteEnumField
code : MultiEnumField (Depends on 0)
chksum : XShortField (Cond)
id : XShortField (Cond)
seq : XShortField (Cond)
ts_ori : ICMPTimeStampField (Cond)
ts_rx : ICMPTimeStampField (Cond)
ts_tx : ICMPTimeStampField (Cond)
gw : IPField (Cond)
ptr : ByteField (Cond)
reserved : ByteField (Cond)
length : ByteField (Cond)
addr_mask : IPField (Cond)
nexthopmtu : ShortField (Cond)
unused : MultipleTypeField (ShortField, IntField, StrFixedlenField) = ('b'')
>>>
```

When we execute >>>ls(ICMP) we will see all detailed information

And response[0],response[0][0],response[0][0][0] gives detailed info of packets

7) We will simulate a Denial-of-Service assault known as a Land assault (an old-school attack) using Scapy. A machine's CPU would spike to 100% and occasionally crash if you sent it a spoof TCP SYN packet over an open port with the source and destination IP addresses configured to the same values. To end Tcpcat, use CTRL-C into the terminal window.



In the above screenshot we can see land-style attack packets.

8) Now , we will use Scapy as a port scanner to scan our Ubuntu Linux machine

In order to do this, we need to execute the following command

tcpdump -nn tcp and host *Ubuntu IP_Address* → in linux machine

```
>>> packet=IP(dst="Ubuntu IP_Address")/TCP(dport=(20,30))
```

```
>>> sr(packet) → need to execute this in scapy terminal.
```

Please check below screenshots for detailed information.

In tcpdump output, you can infer the state of ports based on the types of packets captured. Here's how you can interpret them:

[S]: This indicates a SYN packet, which is sent by the client to initiate a connection.

[R]: This indicates a response packet, typically a SYN-ACK packet sent by the server in response to the client's SYN packet.

[S.]: This indicates a SYN packet as well but followed by a dot. It's a variation that may indicate the beginning of a connection attempt.

[R.]: This indicates a response packet to a SYN packet, typically a SYN-ACK packet sent by the server, suggesting that the port is open.

Thus, we can readily draw our own conclusions that when we encounter the following situations.

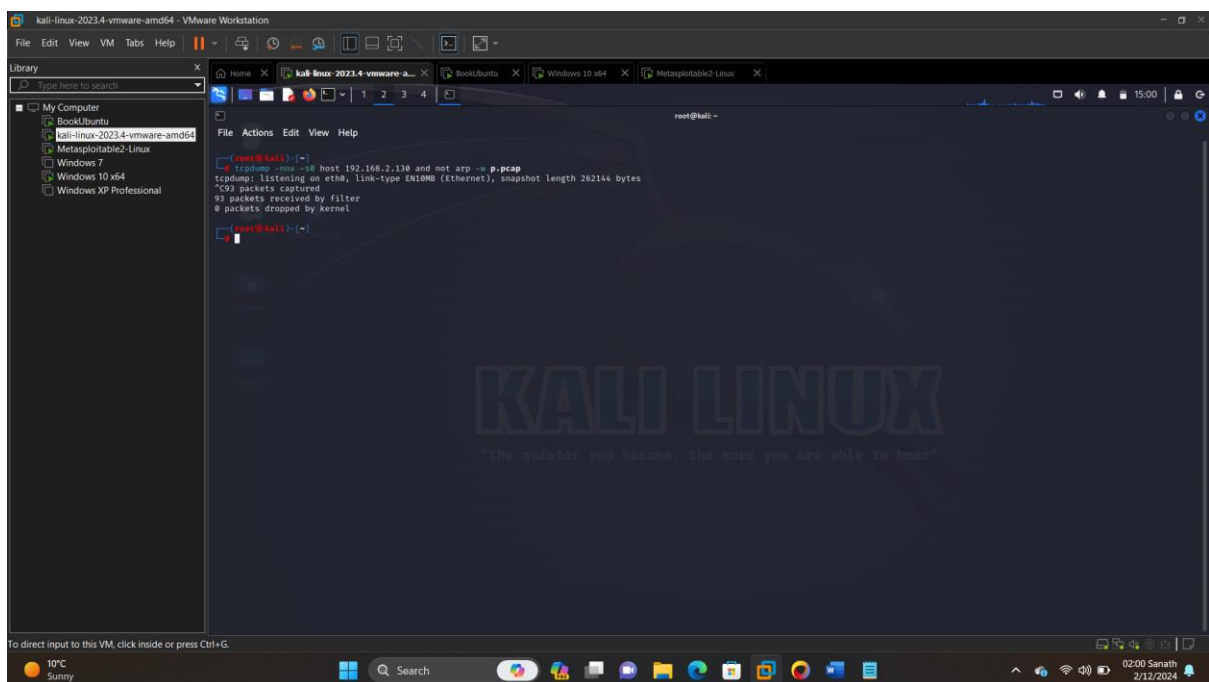
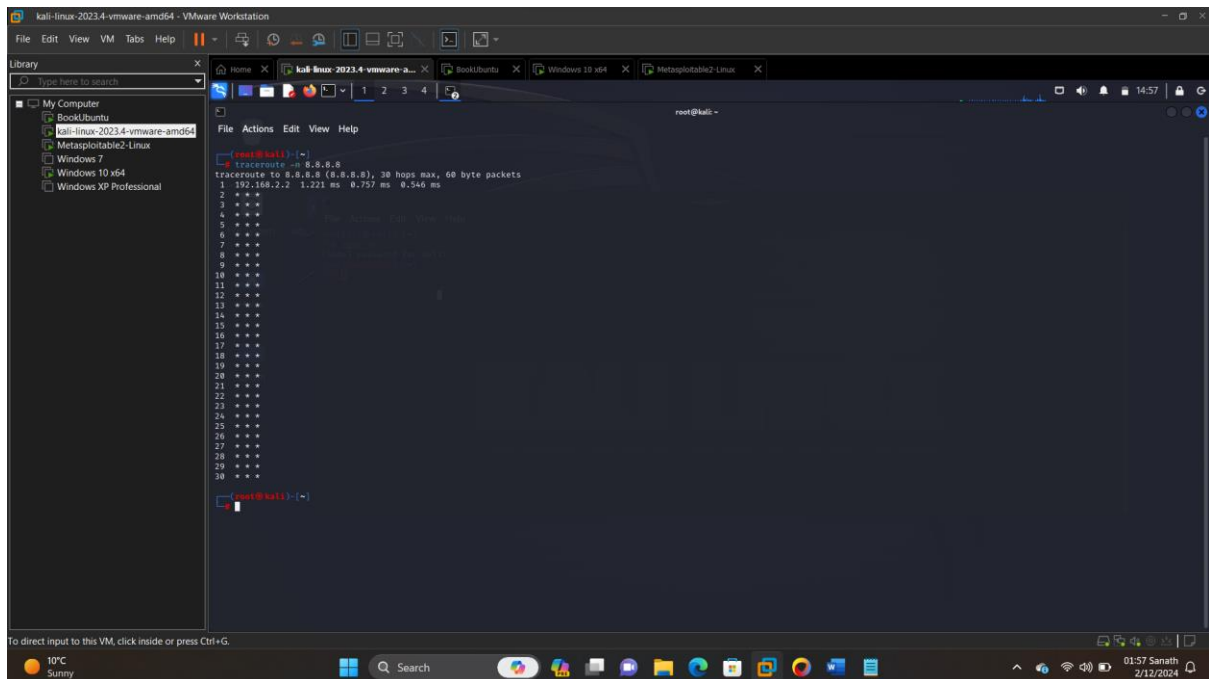
[S] or [S.] followed by [R.]: It suggests that the port is open because the server responded with a SYN-ACK packet, indicating that it received the SYN packet and is willing to establish a connection.

[S] or [S.] without a corresponding [R.]: It suggests that the port is closed because there was no response from the server, indicating that it likely dropped the connection attempt, or the port is not listening.

[R.] without a corresponding [S] or [S.]: It could indicate unsolicited traffic (e.g., packets not part of an attempted connection) or traffic related to an ongoing connection.

9. Now, we will use Scapy to examine some captured packets.

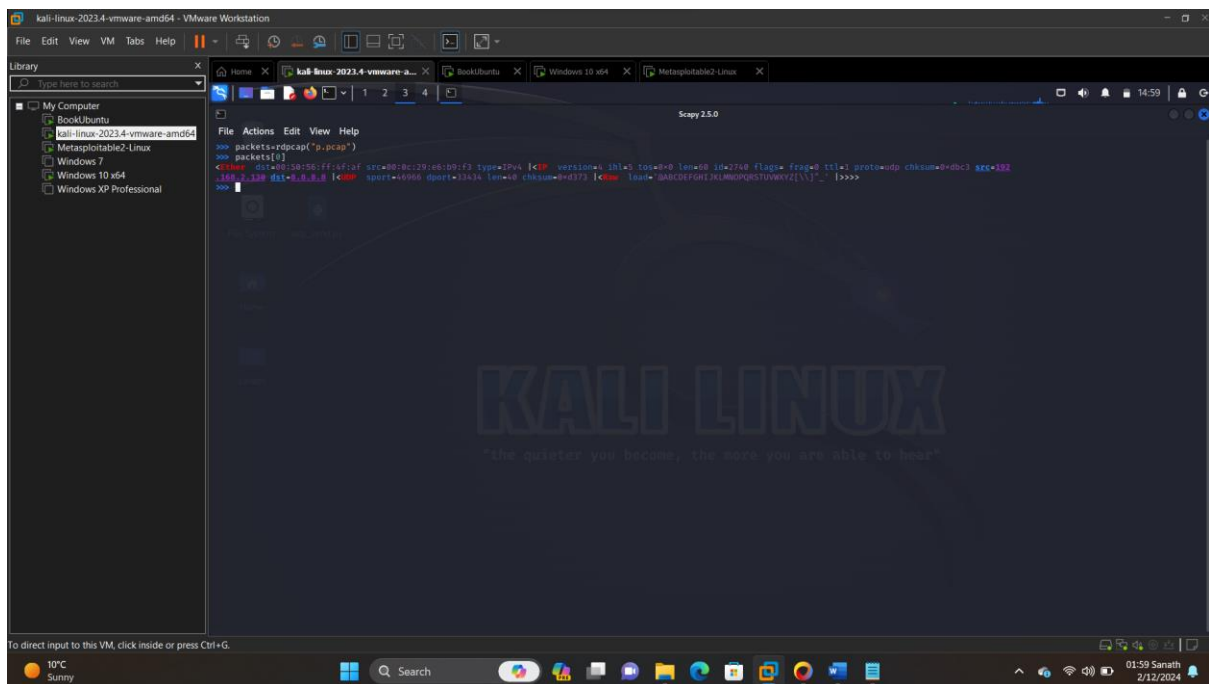
Please check the below screenshots and commands,



Here in the below screenshot we can see that,

TTL value is 1 → In Scapy, the TTL field represents the Time to Live value of an IP packet.

Destination port → `dport = 33434`

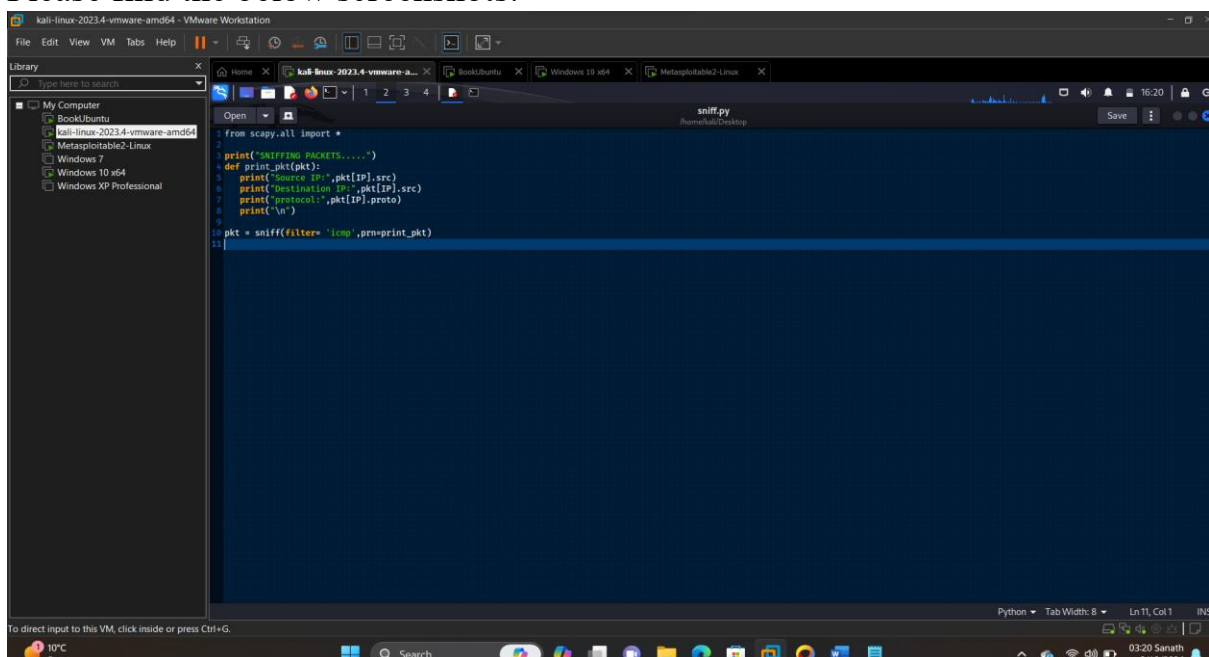


10) Here , we will use scapy's sniff() function to implement a simple sniffer

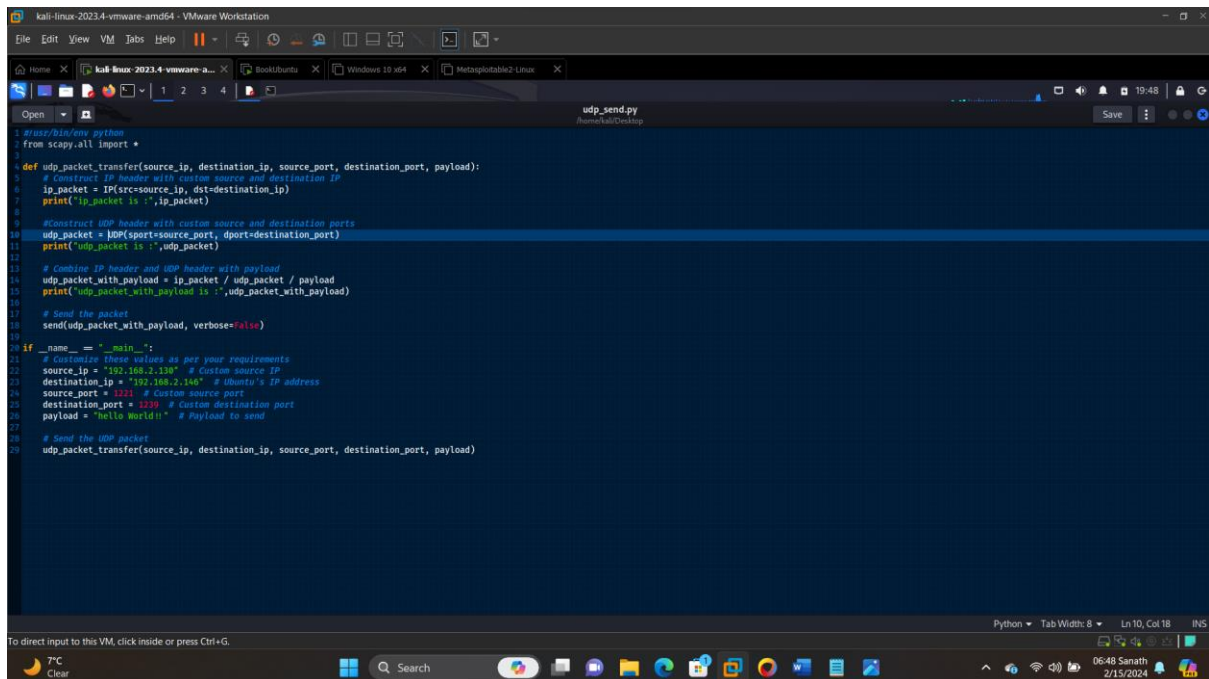
We will use the filter and prn arguments provided by the sniff function.

This is where we have discovered that has both the ICMP echo response packets from 8.8.8.8 and the ICMP echo request packets delivered by the Windows 10 system recorded.

Please find the below screenshots:

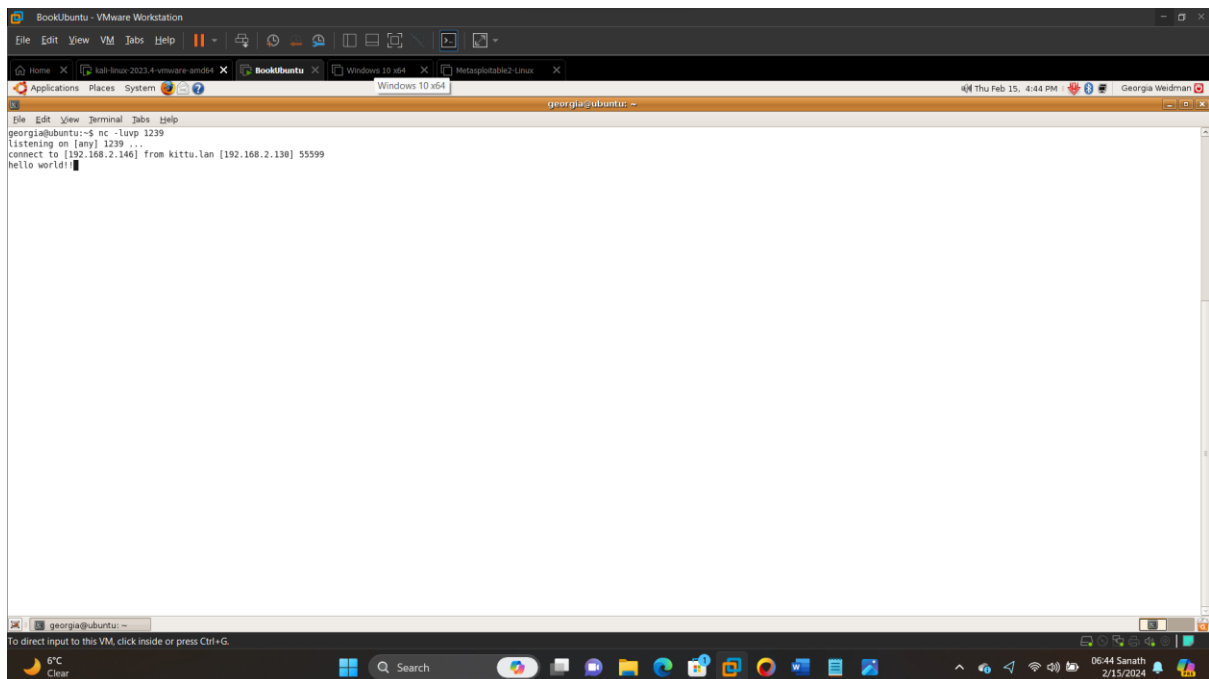


Python script:



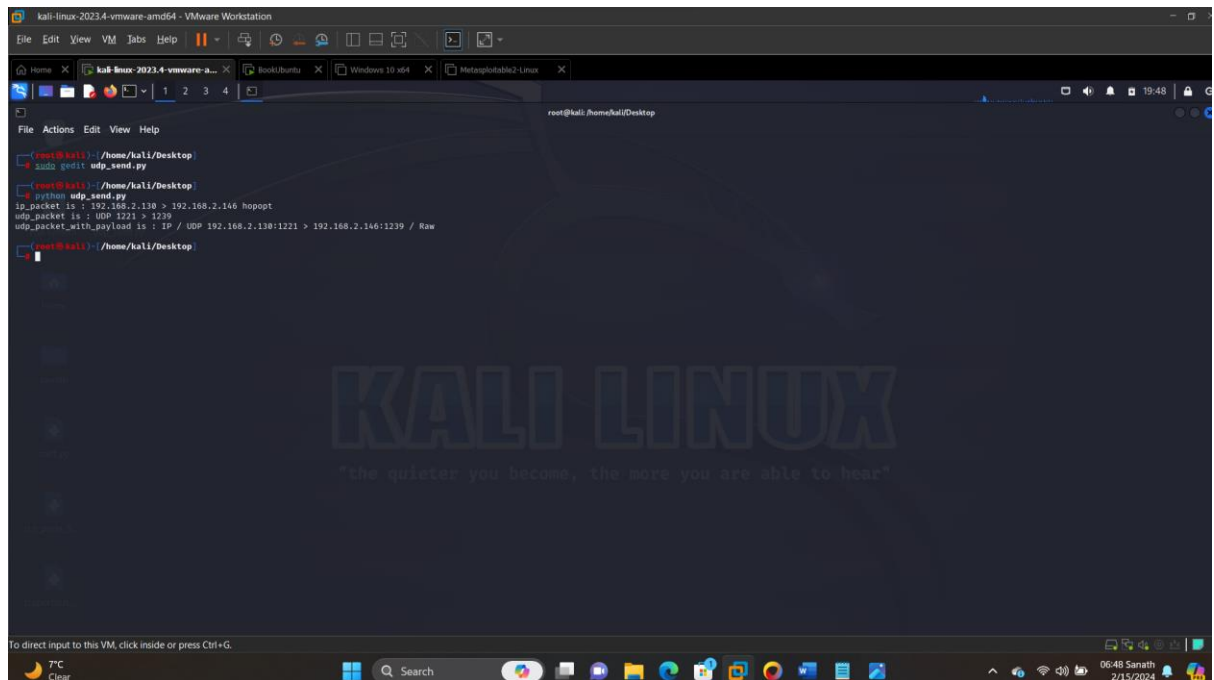
```
1 #!/usr/bin/env python
2 from scapy.all import *
3
4 def udp_packet_transfer(source_ip, destination_ip, source_port, destination_port, payload):
5     # Construct IP header with custom source and destination IP
6     ip_packet = IP(src=source_ip, dst=destination_ip)
7     print("ip_packet is :", ip_packet)
8
9     # Construct UDP header with custom source and destination ports
10    udp_packet = UDP(sport=source_port, dport=destination_port)
11    print("udp_packet is :", udp_packet)
12
13    # Combining IP header and UDP header with payload
14    udp_packet_with_payload = ip_packet / udp_packet / payload
15    print("udp_packet_with_payload is :", udp_packet_with_payload)
16
17    # Send the packet
18    send(udp_packet_with_payload, verbose=False)
19
20 if __name__ == "__main__":
21     # Customize these values as per your requirements
22     source_ip = "192.168.2.130" # Custom source IP
23     destination_ip = "192.168.2.146" # Ubuntu's IP address
24     source_port = 123 # Custom source port
25     destination_port = 1239 # Custom destination port
26     payload = "hello world!!" # Payload to send
27
28     # Send the UDP packet
29     udp_packet_transfer(source_ip, destination_ip, source_port, destination_port, payload)
```

Ubuntu terminal:



```
georgia@ubuntu: ~
georgia@ubuntu:~$ nc -lvp 1239
listening on [any] 1239 ...
connect to [192.168.2.146] from kittu.lan [192.168.2.130] 55599
hello world!!
```

Linux output of python script:

A screenshot of a Kali Linux terminal window. The terminal shows the following commands and output:

```
root@kali:~/home/kali/Desktop# nano udp_send.py
root@kali:~/home/kali/Desktop# python udp_send.py
ip_packet is : 192.168.2.138 > 192.168.2.146 hopopt
udp_packet is : UDP 1221 > 1229
udp_packet_with_payload is : IP / UDP 192.168.2.138:1221 > 192.168.2.146:1239 / Raw
```

The terminal background features the Kali Linux logo and the slogan "the quieter you become, the more you are able to hear". The window title is "kali-linux-2023.4-vmware-amd64 - VMware Workstation".

The commands you provided are using nc (netcat) to listen for incoming connections on port 1234. Let me break down each command:

nc -lulp 1234:

-u: Use UDP instead of TCP.

So, nc -lulp 1234 will start netcat in UDP listen mode on port 1234, showing verbose output.

nc -lvp 1234: here, nc -lvp 1234 will start netcat in TCP listen mode on port 1234, showing verbose output.

These commands are typically used to create a listener on a specific port for testing purposes, for example, to simulate a server or to capture incoming traffic for analysis. The difference between the two commands is the protocol being used (UDP or TCP).

2. Use Scapy to construct packets to scan Metasploitable Linux TCP ports 10-100 (don't change port range). Provide screenshots showing the commands which you use to construct, send, and receive the packets. List all the open ports. Please explain how you determine a port is open based on the packets received from Metasploitable 2 Linux

To construct packets to scan Metasploitable Linux TCP ports we need to follow these steps:

1. Construct TCP SYN packets for ports 10 to 100.
2. Send these packets to the target (Metasploitable Linux)
3. Receive responses from the target.

4. Analyze the responses to determine open ports.

The image shows a Kali Linux virtual machine environment. The top window is a terminal running Scapy 2.5.0. The terminal output includes the Scapy 2.5.0 splash screen with ASCII art, version information (Scapy 2.5.0), and a successful SYN flood attack on port 100. The background features a large 'KALI LINUX' watermark. The bottom window shows the system status bar with the date 05/09/2024 and the user OS09 Sanath.

Here as we see 12,13,14,16,44 and 71 are the list of ports opened and we achieved this with scapy.

Initially, a scanner such as Nmap or a custom script using Scapy sends TCP SYN packets to the target machine on the ports of interest. These packets are sent with the SYN flag set, indicating the initiation of a connection.

Receiving Responses:

Open Port: If the port is open on the target machine, it responds with a TCP SYN-ACK packet. This packet acknowledges the SYN packet and indicates that the port is open and accepting connections.

Closed Port: If the port is closed on the target machine, it responds with a TCP RST packet. This packet indicates that the port is closed and not accepting connections.

Analyzing Responses:

Open Port: If a SYN-ACK packet is received in response to the SYN packet, it confirms that the port is open. The scanner can then mark the port as open in its results.

Closed Port: If a RST packet is received, it confirms that the port is closed. The scanner can mark the port as closed in its results.

Reporting Results:

Based on the analysis of responses received from the target machine, the scanner generates a report indicating which ports are open, closed, or filtered.

the determination of whether a port is open or closed is based on the type of response received from the target machine after sending SYN packets. A SYN-ACK response indicates an open port, a RST response indicates a closed port, and no response may indicate a filtered port. By analyzing these responses, the scanner can determine the state of each port on the target machine.