Name **Hafsa Saleem**

Internship Task 1 **Network sniffer**

Last date **31-10-2024**

Submitted to **Rhombix Technologies**

Domain **Cyber Security**

In this project, I have developed a network sniffer in Python, which captures and analyzes network packets to identify information such as source and destination of Ethernet, IP, TCP, UDP, and ICMP packets.

The Project code are as followed:

```
from scapy.all import sniff, Ether, IP, TCP, UDP, ICMP


# Packet Analysis Function

def process_packet(packet):

  # Check if the packet has an Ethernet layer

  if Ether in packet:

    print("\nEthernet Frame:")

    print(f'\tSource MAC: {packet[Ether].src}, Destination MAC: {packet[Ether].dst}')

    if IP in packet:

      print(f'\tProtocol: IPv4')

      print(f'\tSource IP: {packet[IP].src}, Destination IP: {packet[IP].dst}')


      # Check if it's ICMP, TCP, or UDP

      if ICMP in packet:

        print(f'\tICMP Packet: Type={packet[ICMP].type} Code={packet[ICMP].code}')

      elif TCP in packet:

        print(f'\tTCP Segment: Source Port={packet[TCP].sport}, Destination Port={packet[TCP].dport}')

      elif UDP in packet:

        print(f'\tUDP Segment: Source Port={packet[UDP].sport}, Destination Port={packet[UDP].dport}')

    else:

      print("\tProtocol: Non-IP")
```

```
# Main function to start sniffing

def main():

    # Capture packets, apply a filter for only IP packets, and process them with
process_packet function

    print("Starting network sniffer...")

    sniff(filter="ip", prn=process_packet)


if __name__ == "__main__":

    main()
```

Here is the explanation of this code.

This Python code serves as a foundational network packet sniffer utilizing the Scapy library. It intercepts network packets and outputs key details such as protocol types and addresses.

Let's break it down step by step:

1. Importing Required Modules

python

Copy code

from scapy.all import sniff, Ether, IP, TCP, UDP, ICMP

scapy: A powerful Python library used to capture and analyze network packets.

We import specific classes: sniff, Ether, IP, TCP, UDP, and ICMP for handling various protocols.

2. Defining the process_packet Function

python

Copy code

def process_packet(packet):

This function handles each packet captured by the sniffer. It analyzes and prints details about the packet's contents.

Ethernet Frame:

python

Copy code

```
if Ether in packet:
    print("\nEthernet Frame:")
    print(f'\tSource MAC: {packet[Ether].src}, Destination MAC: {packet[Ether].dst}')
```

Ether in packet: Checks if the packet is an Ethernet frame.

Source MAC and Destination MAC: The Ethernet frame contains the MAC (hardware) addresses of the source (sender) and destination (receiver).

IP Packet:

python

Copy code

```
if IP in packet:
    print(f'\tProtocol: IPv4')
    print(f'\tSource IP: {packet[IP].src}, Destination IP: {packet[IP].dst}')
```

IP in packet: Checks if the packet contains an IP header.

Source IP and Destination IP: These are the sender's and receiver's IP addresses.

ICMP, TCP, or UDP Packet:

python

Copy code

```
if ICMP in packet:
    print(f'\tICMP Packet: Type={packet[ICMP].type} Code={packet[ICMP].code}')
elif TCP in packet:
    print(f'\tTCP Segment: Source Port={packet[TCP].sport}, Destination Port={packet[TCP].dport}')
elif UDP in packet:
```

print(f'\tUDP Segment: Source Port={packet[UDP].sport}, Destination Port={packet[UDP].dport}')

ICMP (Internet Control Message Protocol): Often used for diagnostic tools like ping.

Type and Code: Give more details about the ICMP message (e.g., echo request, echo reply).

TCP (Transmission Control Protocol): Used for reliable data transfer (e.g., web traffic, email).

Source Port and Destination Port: The source and destination ports used by applications (e.g., port 80 for HTTP).

UDP (User Datagram Protocol): Used for simpler, connectionless communication (e.g., video streaming).

Source Port and Destination Port: As with TCP, but for UDP traffic.

3. Defining the main Function

python

Copy code

```
def main():
    print("Starting network sniffer...")
    sniff(filter="ip", prn=process_packet)
```

sniff(filter="ip", prn=process_packet): This function starts the packet sniffer:

filter="ip": Only capture IP packets (which include TCP, UDP, ICMP, etc.).

prn=process_packet: For every packet captured, call the process_packet function to analyze and print its details.

4. Running the Program

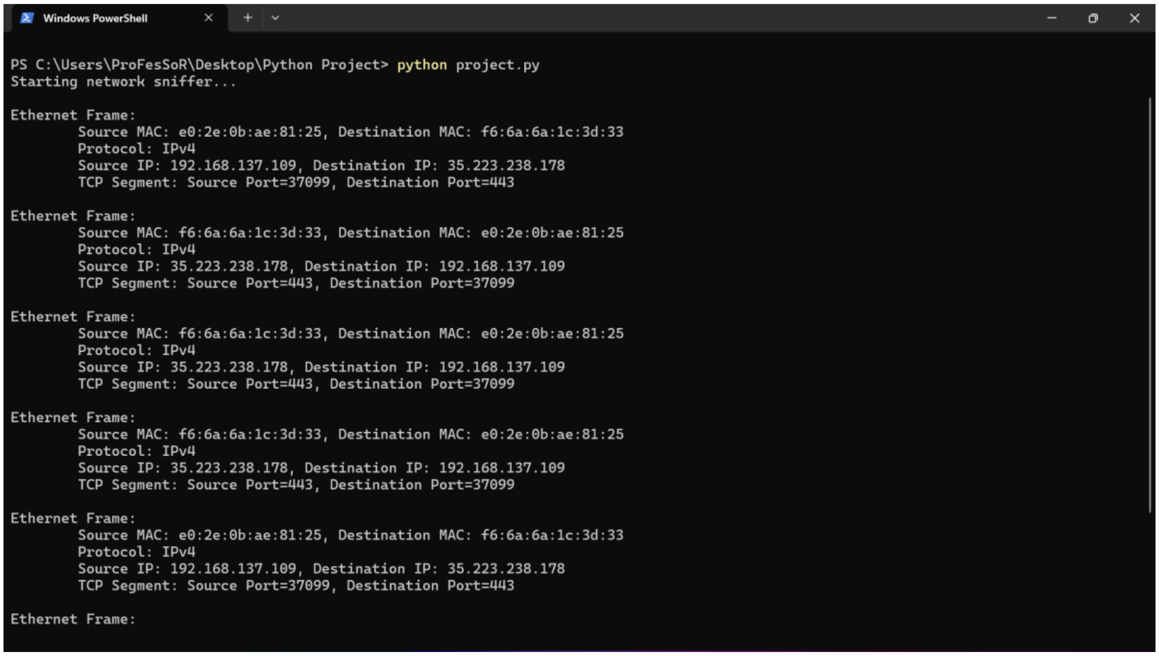
python

Copy code

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

This ensures that when you run the script, the main function is executed, and the packet sniffer starts.

Output Result:



```
Windows PowerShell

PS C:\Users\ProFesSoR\Desktop\Python Project> python project.py
Starting network sniffer...

Ethernet Frame:
        Source MAC: e0:2e:0b:ae:81:25, Destination MAC: f6:6a:6a:1c:3d:33
        Protocol: IPv4
        Source IP: 192.168.137.109, Destination IP: 35.223.238.178
        TCP Segment: Source Port=37099, Destination Port=443

Ethernet Frame:
        Source MAC: f6:6a:6a:1c:3d:33, Destination MAC: e0:2e:0b:ae:81:25
        Protocol: IPv4
        Source IP: 35.223.238.178, Destination IP: 192.168.137.109
        TCP Segment: Source Port=443, Destination Port=37099

Ethernet Frame:
        Source MAC: f6:6a:6a:1c:3d:33, Destination MAC: e0:2e:0b:ae:81:25
        Protocol: IPv4
        Source IP: 35.223.238.178, Destination IP: 192.168.137.109
        TCP Segment: Source Port=443, Destination Port=37099

Ethernet Frame:
        Source MAC: f6:6a:6a:1c:3d:33, Destination MAC: e0:2e:0b:ae:81:25
        Protocol: IPv4
        Source IP: 35.223.238.178, Destination IP: 192.168.137.109
        TCP Segment: Source Port=443, Destination Port=37099

Ethernet Frame:
        Source MAC: e0:2e:0b:ae:81:25, Destination MAC: f6:6a:6a:1c:3d:33
        Protocol: IPv4
        Source IP: 192.168.137.109, Destination IP: 35.223.238.178
        TCP Segment: Source Port=37099, Destination Port=443

Ethernet Frame:
```