NETWORK TRAFFIC MONITORING TOOL

Developing a network traffic monitoring tool involves creating a program that captures, analyzes, and displays network traffic in real-time. This can be achieved using packet sniffing techniques. Below, outline a simple implementation using Python with the scapy library for packet capturing and analysis.

Prerequisites

1. **Python**: install on your system
2. **Scapy**: A powerful Python library used for network packet manipulation and analysis.

    - Install the necessary Python packages:

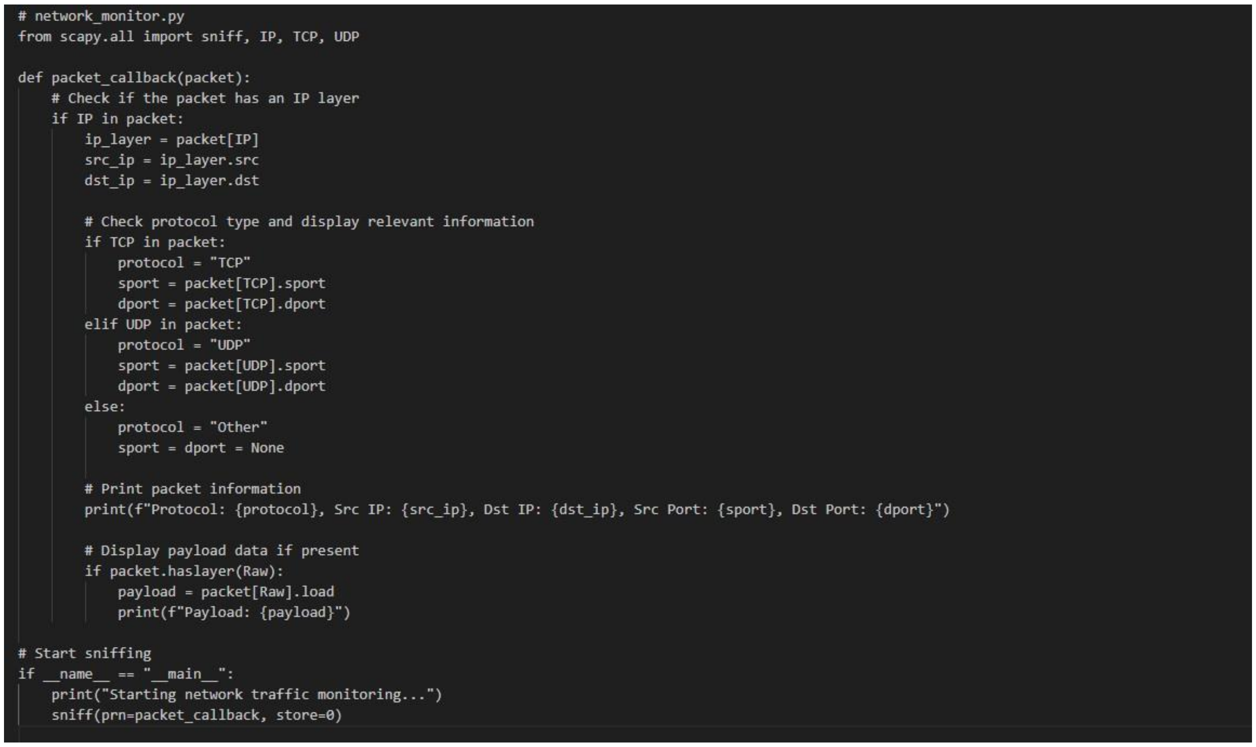
      pip install scapy

IMPLEMENTATION STEPS

1. Set Up Packet Sniffing

The following script sets up basic packet sniffing and real-time display of captured packets, including source/destination IP addresses, protocols, and payloads

Figure 1.Python



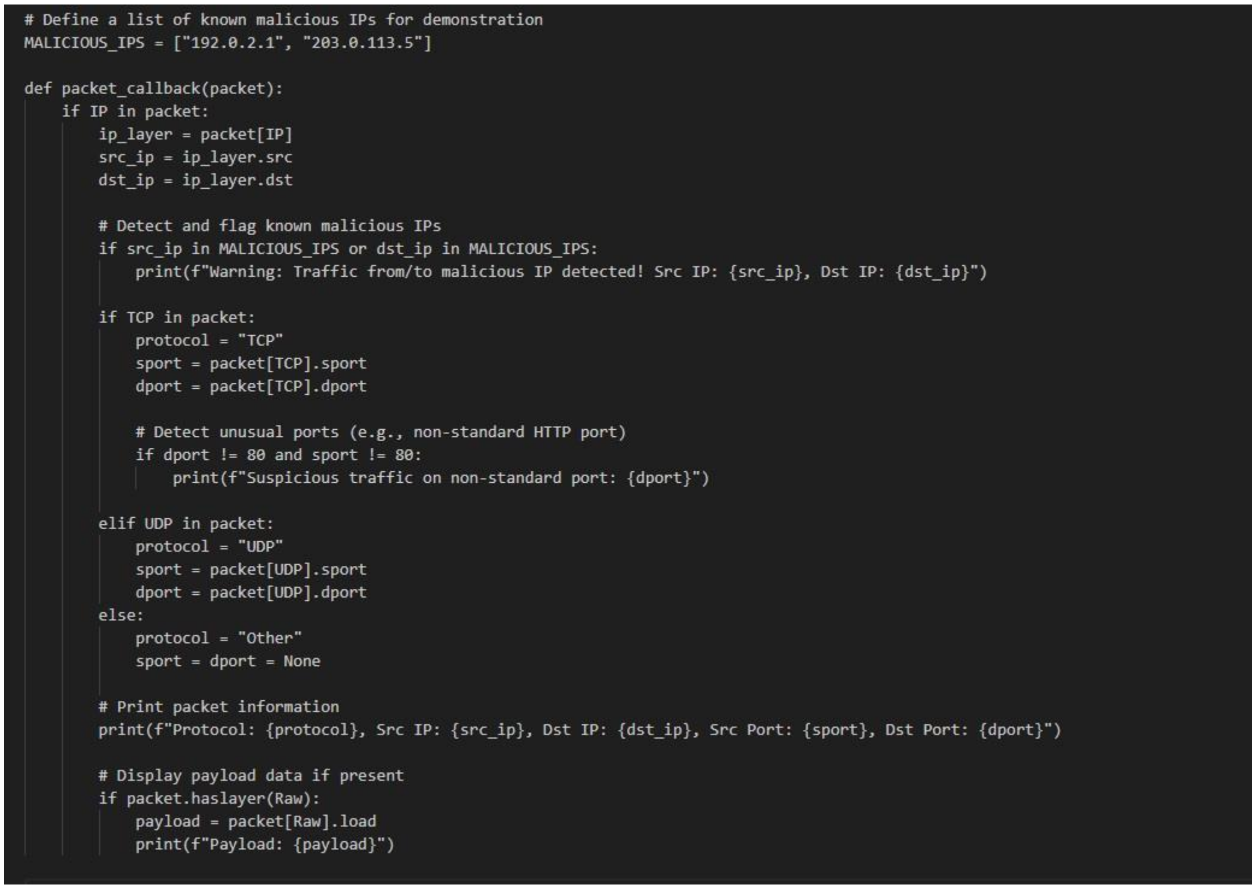
```python
# network_monitor.py
from scapy.all import sniff, IP, TCP, UDP

def packet_callback(packet):
    # Check if the packet has an IP layer
    if IP in packet:
        ip_layer = packet[IP]
        src_ip = ip_layer.src
        dst_ip = ip_layer.dst

        # Check protocol type and display relevant information
        if TCP in packet:
            protocol = "TCP"
            sport = packet[TCP].sport
            dport = packet[TCP].dport
        elif UDP in packet:
            protocol = "UDP"
            sport = packet[UDP].sport
            dport = packet[UDP].dport
        else:
            protocol = "Other"
            sport = dport = None

        # Print packet information
        print(f"Protocol: {protocol}, Src IP: {src_ip}, Dst IP: {dst_ip}, Src Port: {sport}, Dst Port: {dport}")

        # Display payload data if present
        if packet.haslayer(Raw):
            payload = packet[Raw].load
            print(f"Payload: {payload}")

# Start sniffing
if __name__ == "__main__":
    print("Starting network traffic monitoring...")
    sniff(prn=packet_callback, store=0)
```

2. Analyze Traffic for Suspicious Activity

You can extend the packet_callback function to include logic for identifying suspicious or malicious traffic. For example:

- **Unusual Ports**: Detect connections on non-standard ports.
- **Anomalous IPs**: Flag traffic from known malicious IPs (could use threat intelligence feeds).
- **Payload Patterns**: Analyze payloads for signatures of common attacks or malware.

Figure 2. Python



```python
# Define a list of known malicious IPs for demonstration
MALICIOUS_IPS = ["192.0.2.1", "203.0.113.5"]

def packet_callback(packet):
    if IP in packet:
        ip_layer = packet[IP]
        src_ip = ip_layer.src
        dst_ip = ip_layer.dst

        # Detect and flag known malicious IPs
        if src_ip in MALICIOUS_IPS or dst_ip in MALICIOUS_IPS:
            print(f"Warning: Traffic from/to malicious IP detected! Src IP: {src_ip}, Dst IP: {dst_ip}")

        if TCP in packet:
            protocol = "TCP"
            sport = packet[TCP].sport
            dport = packet[TCP].dport

            # Detect unusual ports (e.g., non-standard HTTP port)
            if dport != 80 and sport != 80:
                print(f"Suspicious traffic on non-standard port: {dport}")

        elif UDP in packet:
            protocol = "UDP"
            sport = packet[UDP].sport
            dport = packet[UDP].dport
        else:
            protocol = "Other"
            sport = dport = None

        # Print packet information
        print(f"Protocol: {protocol}, Src IP: {src_ip}, Dst IP: {dst_ip}, Src Port: {sport}, Dst Port: {dport}")

        # Display payload data if present
        if packet.haslayer(Raw):
            payload = packet[Raw].load
            print(f"Payload: {payload}")
```