Network Sniffer in Python

🔍 Introduction

A **network sniffer** is a tool used to **capture and analyze network traffic**. It helps in **monitoring data packets**, debugging network issues, and understanding network behavior.

In this presentation, we will explore a Python-based **network sniffer** using **Scapy**, which allows us to inspect network packets in detail.

⚙️ How It Works

The Python script performs the following actions:

1. **Captures network packets** using `sniff()` from Scapy.
2. **Extracts relevant details** such as:
   - Source and Destination IP addresses
   - Transport protocol (TCP, UDP, ICMP)
   - Port numbers (for TCP/UDP packets)
   - Packet payload (actual data transmitted)
3. **Prints detailed explanations** for each packet field.
4. **Displays the full packet structure** using Scapy's `packet.show()` function.
5. Runs **indefinitely** until manually stopped (Ctrl + C).

🛠️ Code Breakdown

1️⃣ Import Required Libraries

```python
from scapy.all import *
from scapy.layers.inet import IP, TCP, UDP, ICMP
```

- `scapy.all` provides full Scapy functionality.
- `IP, TCP, UDP, ICMP` are specific network protocols we will analyze.

2️⃣ Define a Function to Process Packets

```
def explain_packet(packet):
    print("\n📦 New Packet Captured:")
```

- `packet_callback()` is triggered when a new packet arrives.

3️⃣ Extract IP Layer Details

```
if IP in packet:
    src_ip = packet[IP].src
    dst_ip = packet[IP].dst
    ttl = packet[IP].ttl
    proto = packet[IP].proto
    print(f"   🌐 Source IP: {src_ip} → Destination IP: {dst_ip}")
    print(f"   🔢 TTL (Time To Live): {ttl} (Limits packet travel time)")
    print(f"   📡 Protocol Number: {proto}")
```

- Retrieves **source and destination IP addresses**.
- Extracts **TTL (Time To Live)** which prevents infinite packet looping.
- Identifies the **protocol number** (e.g., 6 for TCP, 17 for UDP).

4️⃣ Extract Transport Layer Details

For TCP Packets:

```
if TCP in packet:
    src_port = packet[TCP].sport
    dst_port = packet[TCP].dport
    flags = packet[TCP].flags
    print(f"   🔵 Protocol: TCP (Reliable, connection-oriented)")
    print(f"   🚪 Source Port: {src_port} → Destination Port: {dst_port}")
    print(f"   🚩 TCP Flags: {flags}")
```

- Extracts **port numbers**.
- Shows **TCP flags**, which define connection states (e.g., SYN, ACK, FIN).

For UDP Packets:

```
elif UDP in packet:
    src_port = packet[UDP].sport
    dst_port = packet[UDP].dport
    print(f"🟢 Protocol: UDP (Fast, connectionless)")
    print(f"🚪 Source Port: {src_port} → Destination Port: {dst_port}")
```

- UDP is a **connectionless protocol**, commonly used for streaming and DNS.

For ICMP Packets:

```
elif ICMP in packet:
    print(f"⚡ Protocol: ICMP (Used for network diagnostics, like ping)")
```

- ICMP is mainly used for **ping requests and network troubleshooting**.

5️⃣ Extract and Display Payload Data

```
payload_data = packet.payload
if len(payload_data) > 0:
    print(f"📄 Payload (Data inside the packet): {payload_data}")
else:
    print("🚫 No Payload (Control packet)")
```

- Displays the **actual data transmitted**.

6️⃣ Show Full Packet Details

```
print("📝 Full Packet Details:\n")
packet.show()
print("=" * 80)
```

- Prints the **entire packet structure** using `packet.show()`.

🚀 Running the Sniffer

```
print("🚀 Starting Network Sniffer... Press Ctrl+C to stop.")
sniff(prn=explain_packet)
```

- `sniff()` listens for packets and passes them to `explain_packet()`.
- Runs until manually stopped (Ctrl + C).

🖥️ Sample Output

```
📦 New Packet Captured:
   🌐 Source IP: 192.168.1.2  →  Destination IP: 8.8.8.8
   🔢 TTL: 64 (Limits packet travel time)
   📡 Protocol Number: 6 (TCP)
   🔵 Protocol: TCP (Reliable, connection-oriented)
   🚪 Source Port: 51432  →  Destination Port: 443 (HTTPS)
   🚩 TCP Flags: PA (Push, Acknowledgment)
   📄 Payload: b'GET / HTTP/1.1\r\nHost: google.com\r\n\r\n'
   📑 Full Packet Details:
## Full Scapy packet structure ##
================================================================================
```