**Network Sniffer Explanation**

**Introduction**

A network sniffer is a tool used to capture and analyze network packets. This Python-based sniffer leverages the Scapy library to extract key information such as IP addresses, ports, timestamps, and packet sizes. It also supports filtering based on specific protocols (TCP/UDP) and target IPs.

**Code Breakdown**

**1. Import Required Libraries**

**from scapy.all import sniff, IP, TCP, UDP**

**from datetime import datetime**

* scapy.all → Imports functions needed for network packet capture.
* sniff → Captures packets.
* IP, TCP, UDP → Extracts IP and transport layer details.
* datetime → Adds timestamps to logs.

**2. Define Filters (Optional)**

**TARGET\_IP = None # Set to an IP address (e.g., "192.168.1.1") to capture only that IP, or None for all**

**PROTOCOL\_FILTER = ["TCP", "UDP"] # Choose ["TCP"], ["UDP"], or both**

* TARGET\_IP → If set, only packets from/to this IP are logged.
* PROTOCOL\_FILTER → Controls which protocols are captured.

**3. Function to Process and Log Packets**

**def log\_packet(packet):**

**with open("packet\_log.txt", "a", encoding="utf-8") as f:**

* Opens packet\_log.txt in append mode to store logs.
* Uses UTF-8 encoding to avoid character encoding issues.

**if IP in packet:**

**timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Capture timestamp**

**packet\_size = len(packet) # Get packet size**

**src\_ip = packet[IP].src**

**dst\_ip = packet[IP].dst**

* Checks if the packet contains an IP header.
* Extracts the timestamp and packet size.
* Retrieves source and destination IP addresses.

**if TARGET\_IP and (src\_ip != TARGET\_IP and dst\_ip != TARGET\_IP):**

**return # Skip packets that don't match**

* If TARGET\_IP is set, ignore packets that don’t match.

**log = f"[{timestamp}] Packet Size: {packet\_size} bytes\n"**

**log += f"Source: {src\_ip} → Destination: {dst\_ip}\n"**

* Formats the log entry with timestamp, packet size, and IP addresses.

**if TCP in packet and "TCP" in PROTOCOL\_FILTER:**

**log += f"Protocol: TCP | Src Port: {packet[TCP].sport} → Dst Port: {packet[TCP].dport}\n"**

**elif UDP in packet and "UDP" in PROTOCOL\_FILTER:**

**log += f"Protocol: UDP | Src Port: {packet[UDP].sport} → Dst Port: {packet[UDP].dport}\n"**

**else:**

**return # Skip packet if it doesn't match the protocol filter**

* Checks if the packet is TCP or UDP and logs corresponding details.
* Skips packets that do not match the selected protocol filter.

**log += "-" \* 60 + "\n"**

**f.write(log)**

**print(log.strip()) # Print in real-time**

* Adds a separator for readability.
* Writes log entry to file and prints it in real time.

**4. Start Sniffing**

**sniff(prn=log\_packet, store=False)**

* Captures packets and processes them with log\_packet.
* store=False prevents storing packets in memory (saves resources).

**Summary**

This network sniffer captures and logs real-time network traffic, extracting essential details such as source and destination IPs, ports, packet size, and timestamps. It supports filtering by protocol (TCP/UDP) and target IP to focus on specific network traffic. The logged data is stored in a text file for later analysis, making it useful for security monitoring, debugging, and network analysis. The implementation ensures efficiency by avoiding unnecessary memory storage and enabling real-time output for immediate insights.