REAL-TIME NETWORK PACKET SNIFFER WITH ANOMALY DETECTION

(WITH LIVE TRAFFIC IN GUI WITH MATPLOTLIB + TKINTER)

SUBMITTED BY

SAMPENGALA VIJAYA KUMAR

REAL-TIME NETWORK PACKET SNIFFER WITH ANOMALY DETECTION

INTRODUCTION

In modern computer networks, data is transmitted as small units called packets. Monitoring and analyzing these data packets is crucial for ensuring network performance, detecting security threats, and troubleshooting connectivity issues. Packet sniffing refers to the process of capturing and inspecting these packets as they travel across a network. This technique is widely used by network administrators and cybersecurity professionals to gain insights into network traffic and identify anomalies or unauthorized activities.

A packet sniffer is a software or hardware tool that intercepts, logs, and analyzes data packets transmitted over a network. It operates by placing the network interface into a mode—often called promiscuous mode—that allows it to capture all packets passing through, regardless of their intended destination. Packet sniffers provide detailed visibility into network communications, including IP addresses, port numbers, protocols used, and other metadata critical for network management and security.

While packet sniffing is an invaluable tool for network optimization and threat detection, it also poses potential privacy risks if used maliciously. Hence, responsible and authorized use of packet sniffers is essential. This project focuses on building a real-time packet sniffer using Python, aimed at capturing traffic, logging it efficiently, detecting suspicious patterns such as port scans and flooding, and providing visual feedback through a user-friendly graphical interface.

Through this work, I gain hands-on experience in network programming, database management, and cybersecurity practices, creating a practical tool that aids in continuous network monitoring and proactive defense.

ABSTRACT

The real-time network packet sniffer project aims to develop a robust tool for capturing and analyzing network traffic, designed primarily for effective network monitoring and security threat detection. Utilizing the Python programming language along with key libraries such as Scapy for packet capture, SQLite for persistent storage, and Matplotlib combined with Tkinter for dynamic visualization, this system provides a comprehensive solution for real-time network traffic analysis.

The project captures packets passing through a network interface, extracting critical header information such as source and destination IP addresses, ports, protocols, packet size, and TCP flags. Captured data is logged efficiently into an SQLite database, enabling both live and historical data analysis. Built-in anomaly detection algorithms identify suspicious behaviour like port scanning and flooding, which are typical indicators of cyber-attacks or network misuse.

Upon detection of threshold breaches, the system triggers alerts via logging mechanisms and optional email notifications to support real-time incident response. The project also features a user-friendly graphical interface that provides continuous visual feedback on traffic protocol distributions and volume, thereby enhancing situational awareness for network administrators.

This packet sniffer tool addresses key challenges in network security by providing flexible monitoring capability, real-time alerting, and insightful visualization. Its modular design and use of open-source technologies make it an extensible platform suited for educational purposes and practical cybersecurity applications alike.

STEP BY STEP PROCEDURE

- **Install Python**

I downloaded and installed Python 3.7+ from the official website and made sure to check the option to add Python to my system PATH. This allowed me to run Python commands from the terminal easily.

- **Install Visual Studio Code**

I downloaded and installed Visual Studio Code (VS Code), and then installed the Microsoft Python extension from the Extensions marketplace. This gave me useful features like syntax highlighting, code completion, and debugging support for Python.

- **Create and Open My Project Folder**

I created a folder named PacketSniffer on my computer where I keep all my project files organized.

In VS Code, I opened this folder using File > Open Folder, so all my code and resources load together in one workspace.

- **Install Required Packages**

With the virtual environment activated, I installed the necessary Python libraries:

```
##text

pip install scapy matplotlib
```

- ***Scapy*** for capturing packets.
- ***Matplotlib*** for creating live traffic visualizations.

I didn't need to install ***SQLite3*** or ***Tkinter*** because they came bundled with Python.

- **Add and Organize My Project Files**

I added my source code files (***init_db.py***, ***sniffer.py, Live_traffic_gui.py)*** to the PacketSniffer folder I created.

Step 2: Initialize the SQLite Database Schema

Code Used:

```
##python
import sqlite3

def init_db():
    conn = sqlite3.connect("packets.db")
    cur = conn.cursor()
    cur.execute('''
        CREATE TABLE IF NOT EXISTS packets (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp TEXT,
            src_ip TEXT,
            dst_ip TEXT,
            src_port INTEGER,
            dst_port INTEGER,
            protocol TEXT,
            length INTEGER,
            flags TEXT
        )
    ''')
    conn.commit()
    conn.close()

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

Run this script once to create the database.

Step 3: Capture Packets and Log Headers

Code Used: (This code captures live packets and logs their details into the database.)

```
##python
from scapy.all import sniff, IP, TCP, UDP
import sqlite3
from datetime import datetime


def log_to_db(pkt):
    if IP in pkt:
        conn = sqlite3.connect("packets.db")
        cur = conn.cursor()
        src_ip = pkt[IP].src
        dst_ip = pkt[IP].dst
        proto = {6: 'TCP', 17: 'UDP'}.get(pkt[IP].proto, str(pkt[IP].proto))
        length = len(pkt)
        src_port = dst_port = flags = None
        if TCP in pkt:
            src_port = pkt[TCP].sport
            dst_port = pkt[TCP].dport
            flags = str(pkt[TCP].flags)
        elif UDP in pkt:
            src_port = pkt[UDP].sport
            dst_port = pkt[UDP].dport
        cur.execute('''
            INSERT INTO packets (timestamp, src_ip, dst_ip, src_port, dst_port, protocol, length, flags)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', (datetime.now().isoformat(timespec="seconds"), src_ip, dst_ip, src_port, dst_port, proto, length, flags))
        conn.commit()
        conn.close()


sniff(prn=log_to_db, store=0)
```

Step 4: Detect Anomalies (Port Scanning, Flooding)

Code Used (to be placed in the same script):

Add ***detect_anomaly(pkt)*** at the end of ***log_to_db(pkt).***

```
##python
from collections import defaultdict, deque
import time

scan_window = 10
scan_threshold = 20
flood_threshold = 100
ports_by_src = defaultdict(lambda: deque())
packets_by_src = defaultdict(lambda: deque())

def detect_anomaly(pkt):
    now = time.time()
    if IP in pkt:
        src_ip = pkt[IP].src
        if TCP in pkt:
            dport = pkt[TCP].dport
            ports_by_src[src_ip].append((now, dport))
            while ports_by_src[src_ip] and now - ports_by_src[src_ip][0][0] > scan_window:
                ports_by_src[src_ip].popleft()
            if len(set(port for (_, port) in ports_by_src[src_ip])) > scan_threshold:
                print(f"ALERT: Port scan suspected from {src_ip}")
        packets_by_src[src_ip].append(now)
        while packets_by_src[src_ip] and now - packets_by_src[src_ip][0] > scan_window:
            packets_by_src[src_ip].popleft()
        if len(packets_by_src[src_ip]) > flood_threshold:
            print(f"ALERT: Flooding suspected from {src_ip}")

# Call detect_anomaly(pkt) inside log_to_db(pkt)
```

Run the Sniffer Script

In VS Code's terminal, run:

```
##text
python sniffer.py
```

- The script now listens for network packets.
- For each relevant packet, it will log a row in ***packets.db*** with all the desired information.

To stop capturing packets while your sniffer is running, use one of these standard methods:

- **Keyboard Interrupt (Most Common)**

While your packet sniffer script runs in the terminal (such as *python* ***sniffer.py***), press *Ctrl+C*
This sends a Keyboard Interrupt signal, and Scapy's ***sniff()*** will stop running and return control to the script or the command line.

- **2. Set a Capture Limit in Code (Optional)**

If you want your script to stop after capturing a certain number of packets, use the count argument:

```
##python
sniff(prn=log_to_db, store=0, count=100)   # Captures 100 packets, then stops
```

- **3. Use a Timeout**

If you want to stop automatically after a set time (e.g., 30 seconds):

```
##python
sniff(prn=log_to_db, store=0, timeout=30)
```

- The sniffer will stop capturing after 30 seconds.

Step 5: Storing Data and Displaying Traffic Summary via GUICode Used (GUI Script):

This script displays a live-updating protocol breakdown chart.

```
##python
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import sqlite3

class LiveTrafficApp:
    def __init__(self, master):
        self.master = master
        self.fig, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(self.fig, master)
        self.canvas.get_tk_widget().pack()
        self.update_chart()

    def update_chart(self):
        conn = sqlite3.connect("packets.db")
        cur = conn.cursor()
        cur.execute("SELECT protocol, COUNT(*) FROM packets GROUP BY protocol")
        results = cur.fetchall()
        conn.close()
        protocols, counts = zip(*results) if results else ([], [])
        self.ax.clear()
        self.ax.bar(protocols, counts)
        self.ax.set_title("Live Protocol Breakdown")
        self.canvas.draw()
        self.master.after(1000, self.update_chart)
root = tk.Tk()
app = LiveTrafficApp(root)
root.mainloop()
```

Step 6: Send alert on threshold breach (via email/log).

To send an alert when an anomaly threshold is breached (such as port scanning or flooding), we can use either logging (print to console or file) or send an alert email using Python's ***smtplib.***

1. Log Alerts: Console or File

Print to Console (Basic)

Add this inside your anomaly detection logic:

```
##python
print(f"ALERT: Port scan suspected from {src_ip}")
print(f"ALERT: Flooding suspected from {src_ip}")
```

Write to a Log File

Add this code to log alerts persistently:

```
##python
def log_alert(message):
    with open("alerts.log", "a") as log_file:
        log_file.write(f"{datetime.now().isoformat()} - {message}\n")
```

Then call *log_alert()* whenever an anomaly is detected:

```
##python
log_alert(f"Port scan suspected from {src_ip}")
log_alert(f"Flooding suspected from {src_ip}")
```

2. Send Alert Emails (with smtplib)

Add this code when threshold is breached:

```
##python
import smtplib
from email.mime.text import MIMEText

def send_email_alert(message, subject="Network Alert"):
    sender = "your_email@gmail.com"
    receiver = "recipient_email@gmail.com"
    password = "your_email_password_or_app_password"

    msg = MIMEText(message)
    msg["Subject"] = subject
    msg["From"] = sender
    msg["To"] = receiver

    try:
        with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
            server.login(sender, password)
            server.sendmail(sender, receiver, msg.as_string())
        print("Alert email sent!")
    except Exception as e:
        print("Error sending email:", e)
```

Call this function when an anomaly is detected:

```
##python
send_email_alert(f"Port scan suspected from {src_ip}")
send_email_alert(f"Flooding suspected from {src_ip}")
```

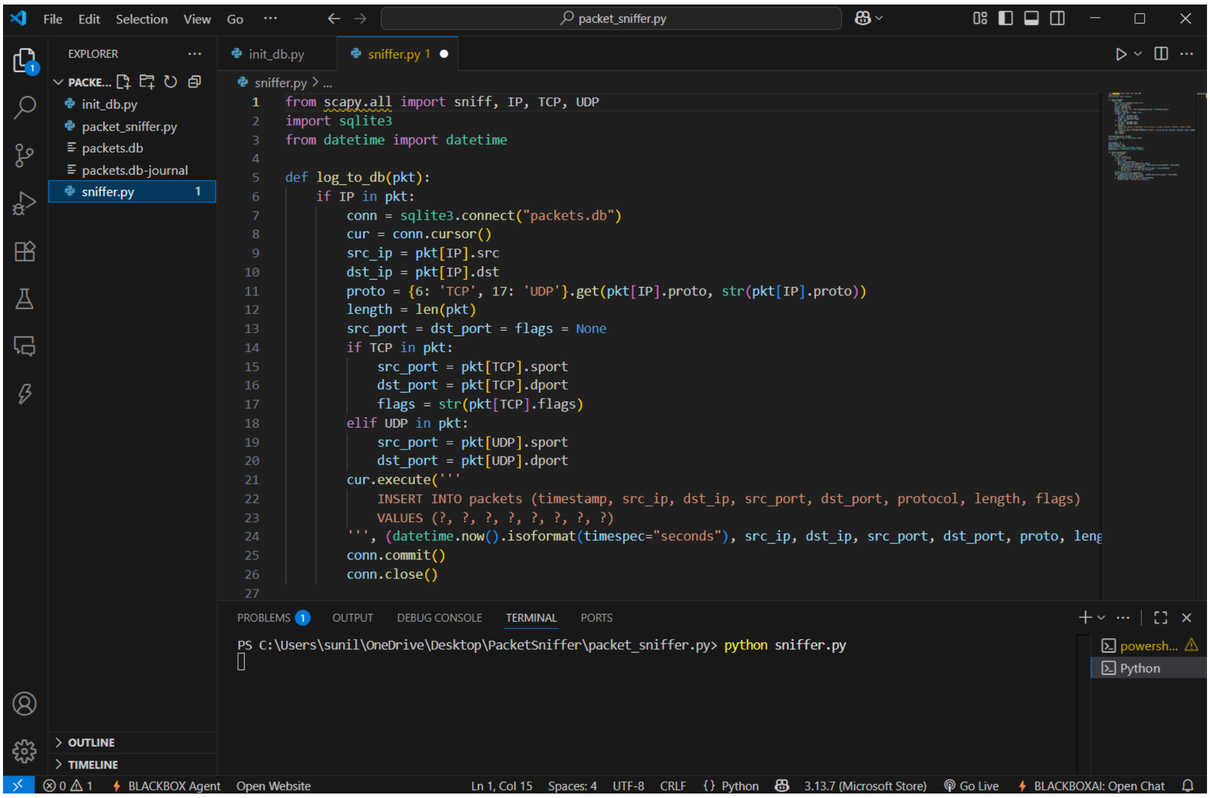
3. How to Integrate with Anomaly Detection

In your anomaly detection function, after confirming the threshold is breached:

```
##python
if len(set(port for (_, port) in ports_by_src[src_ip])) > scan_threshold:
    alert_msg = f"Port scan suspected from {src_ip}"
    print(alert_msg)
    log_alert(alert_msg)
    send_email_alert(alert_msg)
```

PROOF OF CONCEPT:

1. sniffer.py



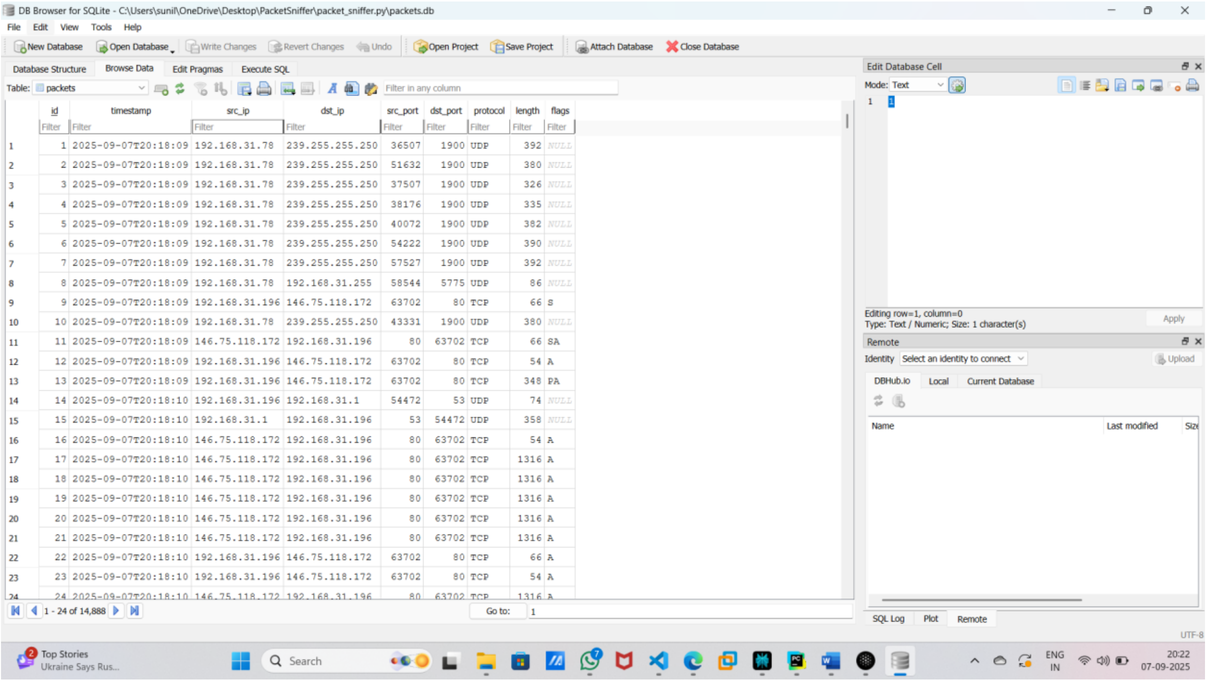
```python
from scapy.all import sniff, IP, TCP, UDP
import sqlite3
from datetime import datetime

def log_to_db(pkt):
    if IP in pkt:
        conn = sqlite3.connect("packets.db")
        cur = conn.cursor()
        src_ip = pkt[IP].src
        dst_ip = pkt[IP].dst
        proto = {6: 'TCP', 17: 'UDP'}.get(pkt[IP].proto, str(pkt[IP].proto))
        length = len(pkt)
        src_port = dst_port = flags = None
        if TCP in pkt:
            src_port = pkt[TCP].sport
            dst_port = pkt[TCP].dport
            flags = str(pkt[TCP].flags)
        elif UDP in pkt:
            src_port = pkt[UDP].sport
            dst_port = pkt[UDP].dport
        cur.execute('''
            INSERT INTO packets (timestamp, src_ip, dst_ip, src_port, dst_port, protocol, length, flags)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', (datetime.now().isoformat(timespec="seconds"), src_ip, dst_ip, src_port, dst_port, proto, leng
        conn.commit()
        conn.close()

```

```
PS C:\Users\sunil\OneDrive\Desktop\PacketSniffer\packet_sniffer.py> python sniffer.py
```

2. *packets.db* **Screenshot of DB browser (SQLite)**


DB Browser for SQLite - C:\Users\sunil\OneDrive\Desktop\PacketSniffer\packet_sniffer.py\packets.db
File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database
Database Structure Browse Data Edit Pragmas Execute SQL
Table: packets
Filter in any column
Edit Database Cell
Mode: Text
Editing row=1, column=0
Type: Text / Numeric; Size: 1 character(s)
Apply
Remote
Identity Select an identity to connect
Upload
DBHub.io Local Current Database
Name Last modified Size
SQL Log Plot Remote
1 - 24 of 14,888
Go to: 1
Top Stories
Ukraine Says Rus...
Search
ENG IN
20:22
07-09-2025

	id	timestamp	src_ip	dst_ip	src_port	dst_port	protocol	length	flags
1	1	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	36507	1900	UDP	392	NULL
2	2	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	51632	1900	UDP	380	NULL
3	3	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	37507	1900	UDP	326	NULL
4	4	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	38176	1900	UDP	335	NULL
5	5	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	40072	1900	UDP	382	NULL
6	6	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	54222	1900	UDP	390	NULL
7	7	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	57527	1900	UDP	392	NULL
8	8	2025-09-07T20:18:09	192.168.31.78	192.168.31.255	58544	5775	UDP	86	NULL
9	9	2025-09-07T20:18:09	192.168.31.196	146.75.118.172	63702	80	TCP	66	S
10	10	2025-09-07T20:18:09	192.168.31.78	239.255.255.250	43331	1900	UDP	380	NULL
11	11	2025-09-07T20:18:09	146.75.118.172	192.168.31.196	80	63702	TCP	66	SA
12	12	2025-09-07T20:18:09	192.168.31.196	146.75.118.172	63702	80	TCP	54	A
13	13	2025-09-07T20:18:09	192.168.31.196	146.75.118.172	63702	80	TCP	348	PA
14	14	2025-09-07T20:18:10	192.168.31.196	192.168.31.1	54472	53	UDP	74	NULL
15	15	2025-09-07T20:18:10	192.168.31.1	192.168.31.196	53	54472	UDP	358	NULL
16	16	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	54	A
17	17	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A
18	18	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A
19	19	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A
20	20	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A
21	21	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A
22	22	2025-09-07T20:18:10	192.168.31.196	146.75.118.172	63702	80	TCP	66	A
23	23	2025-09-07T20:18:10	192.168.31.196	146.75.118.172	63702	80	TCP	54	A
24	24	2025-09-07T20:18:10	146.75.118.172	192.168.31.196	80	63702	TCP	1316	A

The 2^{nd} Screenshot, shows the database structure and packet data within your SQLite database (***packets.db***) captured by network sniffer.

Database Details from Screenshot

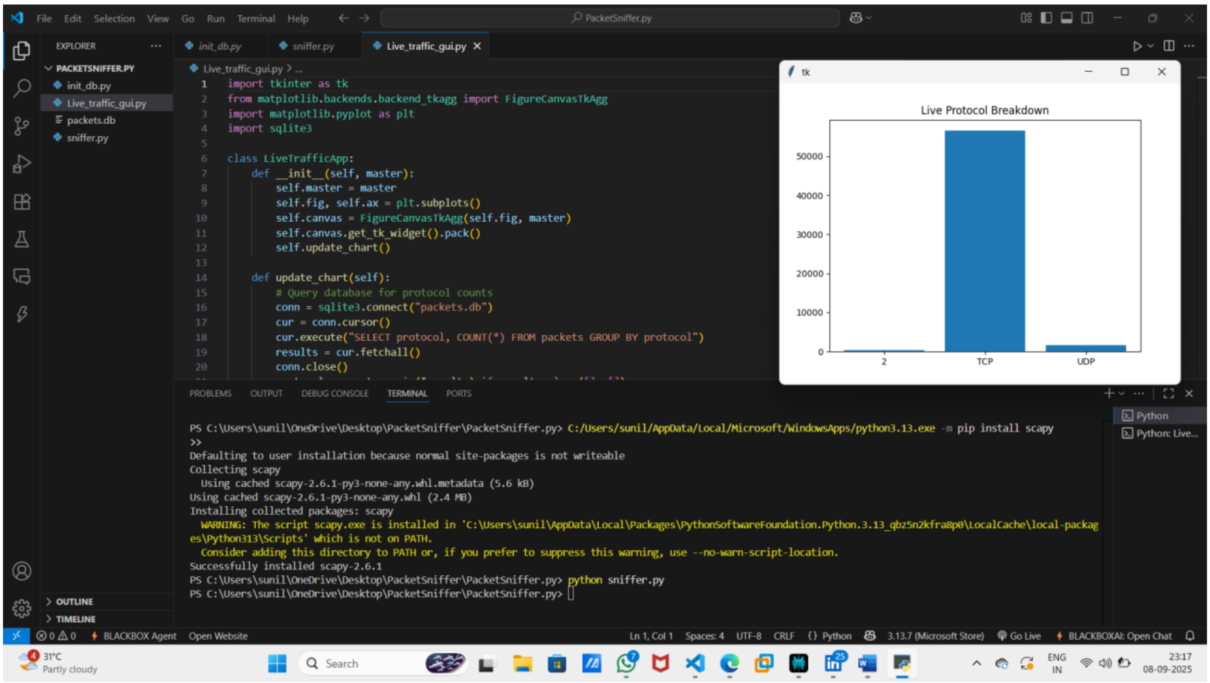
- The SQLite table packets stores each captured packet with fields like:
  - id (unique row index)
  - timestamp (packet capture time)
  - src_ip (source IP)
  - dst_ip (destination IP)
  - src_port and dst_port (TCP/UDP ports)
  - protocol (protocol type—TCP/UDP)
  - length (packet size in bytes)
  - flags (TCP flags if present)
- The image reflects active packet captures, mostly for UDP protocol, with TCP packets including flags such as S, SA, PA, and A.
- Data is automatically updated and includes thousands of records (indicated by navigation at the bottom).

Contextual Integration

This database snapshot demonstrates that Python sniffer correctly logs real, granular traffic data—including metadata essential for security monitoring—into a persistent and queryable structure.
It enables:

- Efficient querying of traffic trends or anomalies
- Storing evidence for forensic/network analysis
- Supporting live dashboard visualizations and security alerts

3.Live Traffic Summary (GUI)



```python
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import sqlite3

class LiveTrafficApp:
    def __init__(self, master):
        self.master = master
        self.fig, self.ax = plt.subplots()
        self.canvas = FigureCanvasTkAgg(self.fig, master)
        self.canvas.get_tk_widget().pack()
        self.update_chart()

    def update_chart(self):
        # Query database for protocol counts
        conn = sqlite3.connect("packets.db")
        cur = conn.cursor()
        cur.execute("SELECT protocol, COUNT(*) FROM packets GROUP BY protocol")
        results = cur.fetchall()
        conn.close()
```

```
PS C:\Users\sunil\OneDrive\Desktop\PacketSniffer\PacketSniffer.py> C:/Users/sunil/AppData/Local/Microsoft/WindowsApps/python3.13.exe -m pip install scapy
>>
Defaulting to user installation because normal site-packages is not writeable
Collecting scapy
  Using cached scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
Using cached scapy-2.6.1-py3-none-any.whl (2.4 MB)
Installing collected packages: scapy
  WARNING: The script scapy.exe is installed in 'C:\Users\sunil\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packag
es\Python313\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed scapy-2.6.1
PS C:\Users\sunil\OneDrive\Desktop\PacketSniffer\PacketSniffer.py> python sniffer.py
PS C:\Users\sunil\OneDrive\Desktop\PacketSniffer\PacketSniffer.py>
```

The 3rd screenshot provides a visual overview of project directory in VS Code, the source code for live traffic GUI, terminal commands, and a live GUI chart generated by Matplotlib and Tkinter.

- Project Organization:
  Project is neatly structured with files for database initialization (init_db.py), packet sniffing (***sniffer.py***), and GUI visualization (***Live_traffic_gui.py***). The ***packets.db*** SQLite file appears in the workspace, confirming successful data capture and storage.

- Live Traffic GUI:

  - The ***Live_traffic_gui.py*** code demonstrates how you fetch protocol counts from the database and plot them in real time using Tkinter and Matplotlib.

  - The running application window titled “Live Protocol Breakdown” displays a bar chart, visually summarizing captured protocol traffic (with TCP as the dominant protocol and UDP present).

  - This chart updates at regular intervals, automatically reflecting incoming network traffic.

Project Value as Demonstrated

This screenshot confirms that my project delivers on its objectives:

- Live packet capture with persistent logging to SQLite,
- Automated real-time visualization of network traffic,
- User-friendly monitoring with clear protocol breakdowns for rapid situational awareness.

The GUI chart is an important asset for this project report, showing how technical backend processing can be translated into actionable, accessible insights for analysts or administrators.

As seen in the provided screenshot, the developed GUI dynamically retrieves protocol statistics from the continuously growing SQLite packet log and visually presents real-time traffic trends. This feature streamlines anomaly detection and aids network administrators in quickly identifying traffic spikes, protocol imbalances, or potential threats.

CONCLUSION

This project successfully implements a real-time network packet sniffer equipped with anomaly detection, persistent data logging, and live graphical visualization. Through the use of Python’s Scapy library, network packets are captured continuously, enabling comprehensive monitoring of network traffic. Key packet attributes—including source and destination IPs, ports, protocols, packet length, and TCP flags—are systematically extracted and recorded in an SQLite database, providing a reliable and scalable backend for traffic analysis.

The integration of anomaly detection algorithms facilitates the identification of suspicious activities such as port scanning and flooding attacks by analyzing traffic patterns over specific time windows. Alerts generated by threshold breaches empower prompt awareness and reaction to potential security incidents, which is critical in minimizing network vulnerabilities.

Beyond the backend processing, the project features an interactive GUI built with Tkinter and Matplotlib. This interface dynamically presents live traffic summaries, allowing users to visualize the distribution of protocols and monitor traffic trends in real time. Such visualization enhances situational awareness and greatly assists in network troubleshooting and security monitoring.

Overall, this project demonstrates the transformative power of combining accessible, open-source technologies into an effective network monitoring solution. It provides a foundation that can be expanded with advanced detection techniques, support for additional protocols, integration with larger security platforms, or enhanced alerting capabilities. By implementing this system, I have gained hands-on experience with packet-level network analysis and the critical practices required for proactive cybersecurity defense.

This work serves as a meaningful step toward building comprehensive cybersecurity tools that are both practical and adaptable to real-world network environments.