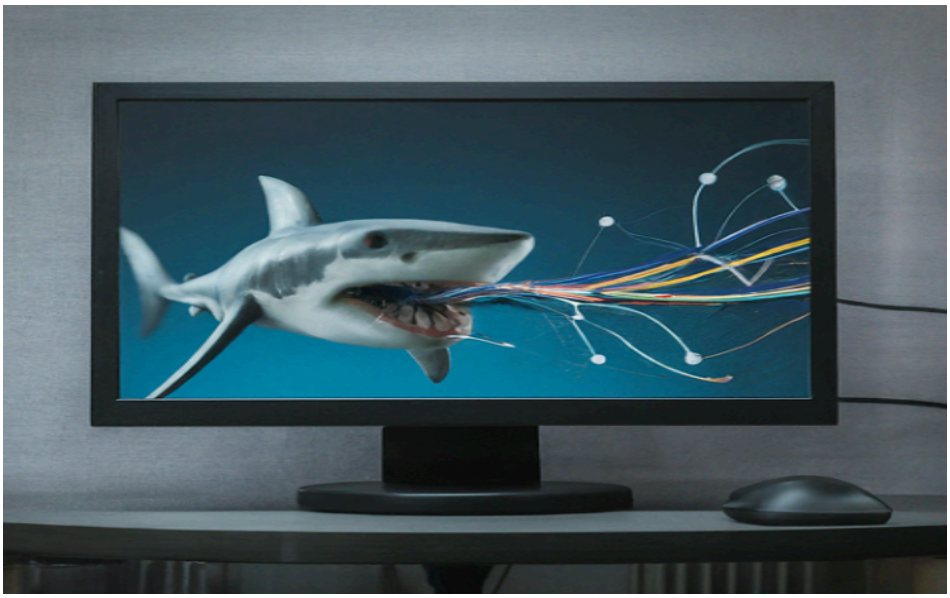
PCAP Analysis Python Program

Felix



“Mmm packets”
or
“Using deadly wildlife to interpret internet traffic”

Table of contents

1. Introduction

This report details the creation and investigation of a PCAP Analysis Program, a Python-based program designed to process and analyze packet capture (PCAP) files from wireshark. The program offers a basic solution for a user to examine network traffic data efficiently. By using various Python libraries and SQLite database functionality, the tool provides insights into packet information, protocol distribution, and top talkers within a network.

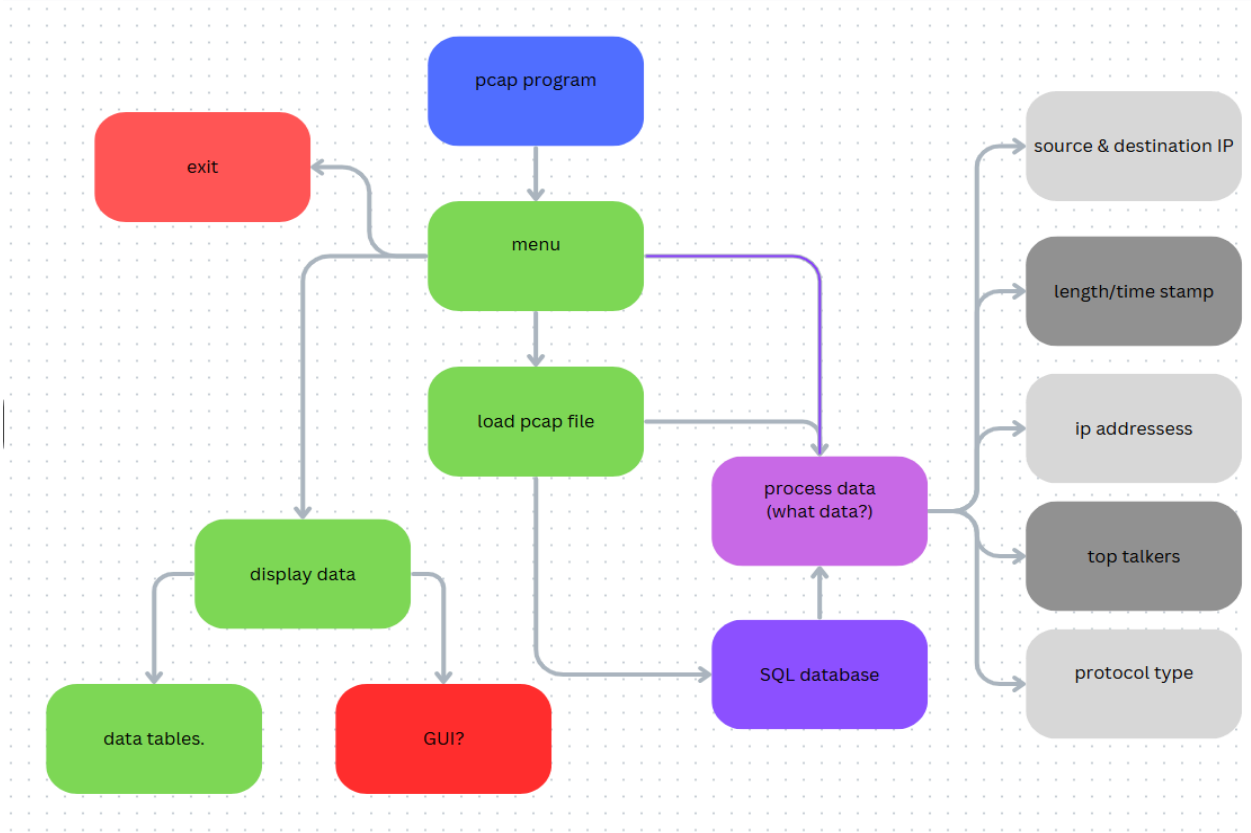
What is wireshark and why should we be interested in what information it can show us? Wireshark is a widely-used network protocol analyzer that allows users to capture and interactively browse the traffic running on a computer network. It provides deep insight into hundreds of protocols and is an essential tool for network troubleshooting, analysis, and cybersecurity. Inside wireshark we can find a lot of information about packets. Packet data is crucial in cybersecurity as it contains detailed information about network communications. By analyzing packet data, we can identify malicious activities, detect vulnerabilities, and monitor network performance. The objective of our program is to create a basis from which we can easily tabulate, display, organize and interpret the information saved in PCAP files containing our network information.

2. Project Outline

The PCAP Analysis Program will be structured as a command-line interface application with the following main components:

1. PCAP file loading
2. Basic packet information processing
3. Protocol distribution analysis
4. Top talkers identification
5. Database creation and management
6. Results storage and display

2.1 Project Design



In the above mind map we have devised a basic lay out for our program. We will use several libraries to accomplish our task for accessing and interpreting our wireshark data. We will use pyshark to access our .pcapng file and for our database we will use a simple sql database in the form of sqlite3.

3. Libraries and Functions

3.1 Libraries

The program utilizes several Python libraries to achieve its functionality:

1. **pyshark**: A Python wrapper for tshark, allowing the program to read and parse PCAP files. It provides an interface to access packet information without requiring direct interaction with the lower-level pcap files.
2. **sqlite3**: This library enables the program to interact with SQLite databases. It's used to create tables, insert data, and retrieve information for display.
3. **tabulate**: A library that facilitates the creation of formatted tables for displaying results in the console. It enhances the readability of the output by presenting data in a structured, easy-to-read format.

3.2 Key Functions

3.2.1 load_pcap(file_path)

This function is responsible for loading the PCAP file specified by the user. It utilizes the pyshark library to create a FileCapture object, which allows for packet-by-packet analysis of the PCAP file. The function includes error handling in the form of “try” and “except” blocks, to manage exceptions that may occur during file loading, such as file not found or permission issues.

```python
def load_pcap(file_path):
    try:
        print(f"Loading PCAP file: {file_path}")
        capture = pyshark.FileCapture(file_path)
        print("PCAP file loaded successfully.")
        return capture
    except Exception as e:
        print(f"Error loading PCAP file: {e}")
        return None
```

3.2.2 process_basic_info(capture)

This function extracts basic information from each packet in the capture. It iterates through the packets, focusing on those with IP layer information. For each relevant packet, it extracts the source IP, destination IP, transport layer protocol, packet length, and timestamp. This information is crucial for understanding the general characteristics of the network traffic.

```python
def process_basic_info(capture):
    print("Processing basic packet information...")
    basic_info = []
    for packet in capture:
        if 'IP' in packet:
            basic_info.append({
                'src_ip': packet.ip.src,
                'dst_ip': packet.ip.dst,
                'protocol': packet.transport_layer,
                'length': packet.length,
                'timestamp': packet.sniff_timestamp
            })
    print("Basic packet information processed.")
    return basic_info
```

3.2.3 process_protocol_distribution(capture)

This function analyzes the protocol distribution within the captured traffic. It examines the highest layer protocol of each packet and maintains a count of occurrences for each protocol type. This analysis provides insights into the types of applications and services being used on the network.

```
def process_protocol_distribution(capture):
    print("Processing protocol distribution...")
    protocols = {}
    for packet in capture:
        if hasattr(packet, 'highest_layer'):
            protocol = packet.highest_layer
            protocols[protocol] = protocols.get(protocol, 0) + 1
    print("Protocol distribution processed.")
    return protocols
```

hasattr(packet, 'highest_layer'): This checks if the packet object has an attribute named highest_layer. This attribute typically represents the highest protocol layer that the packet belongs to.
protocols[protocol] = protocols.get(protocol, 0) + 1: Updates the count of the protocol in the protocols dictionary. If the protocol is not already in the dictionary, protocols.get(protocol, 0) returns 0, and then 1 is added to it.

3.2.4 process_top_talkers(capture)

The top talkers function identifies the most active IP addresses in the captured traffic. It counts the number of packets associated with each IP address, both as a source and destination. The function then sorts these counts to identify the top 10 most active IP addresses, which can be crucial for identifying potential security issues or understanding network usage patterns.

```
def process_top_talkers(capture):
    print("Processing top talkers...")
    ip_counts = {}
    for packet in capture:
        if 'IP' in packet:
            src_ip = packet.ip.src
            dst_ip = packet.ip.dst
            ip_counts[src_ip] = ip_counts.get(src_ip, 0) + 1
            ip_counts[dst_ip] = ip_counts.get(dst_ip, 0) + 1
    top_talkers = sorted(ip_counts.items(), key=lambda x: x[1], reverse=True)[:10]
    print("Top talkers processed.")
    return top_talkers
```

ip_counts[src_ip] = ip_counts.get(src_ip, 0) + 1: Updates the count of packets sent by the source IP address. If the source IP is not already in the dictionary, ip_counts.get(src_ip, 0) returns 0, and then 1 is added to it.

ip_counts[dst_ip] = ip_counts.get(dst_ip, 0) + 1: Updates the count of packets received by the destination IP address in the same manner.

top_talkers = sorted(ip_counts.items(), key=lambda x: x[1], reverse=True): Sorts the items in the ip_counts dictionary by the count of packets (the second item in each tuple) in descending order.

reverse=True: This parameter sorts the list in descending order.

[:10]: Selects the top 10 IP addresses with the highest packet counts.

lambda x: x[1]: This is a lambda function. A lambda function is a small anonymous function defined using the lambda keyword. It can have any number of arguments but only one expression.

x: The argument to the lambda function. In this case, x represents each item (a tuple) in the ip_counts.items() list.

x[1]: The expression that the lambda function evaluates. Here, x[1] refers to the second element of the tuple, which is the count of packets for a particular IP address.

3.2.5 create_database()

This function sets up the SQLite database structure for storing the analyzed data. It creates three tables: basic_info, protocol_distribution, and top_talkers. The use of SQLite allows for efficient storage and retrieval of the processed information, enabling further analysis and long-term storage of results.

```
def create_database():
    print("Creating/connecting to SQLite database...")
    conn = sqlite3.connect('pcap_analysis.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS basic_info
                 (src_ip TEXT, dst_ip TEXT, protocol TEXT, length INTEGER, timestamp TEXT)''')
    c.execute('''CREATE TABLE IF NOT EXISTS protocol_distribution
                 (protocol TEXT, count INTEGER)''')
    c.execute('''CREATE TABLE IF NOT EXISTS top_talkers
                 (ip_address TEXT, packet_count INTEGER)''')
    conn.commit()
    print("Database ready.")
    return conn
```

sqlite3.connect('pcap_analysis.db'): This function call connects to an SQLite database named pcap_analysis.db. If the database does not exist, it will be created.

conn: This variable holds the connection object to the database.

conn.cursor(): This method creates a cursor object, which is used to execute SQL commands.

c: This variable holds the cursor object.

c.execute: This method executes an SQL command.

CREATE TABLE IF NOT EXISTS: This SQL command creates a table named x if it does not already exist.

conn.commit(): This method commits the current transaction, saving all changes made to the database.

The create_database function connects to or creates an SQLite database named pcap_analysis.db. It then creates three tables (basic_info, protocol_distribution, and top_talkers) if they do not already exist. The function commits the changes to the database and returns the connection object.

3.2.6 store_results(conn, data, table_name)

This function is responsible for storing the processed data into the appropriate SQLite database tables. It uses SQL INSERT statements to populate the tables with the analyzed information, ensuring that the results are persistently stored for future reference and analysis.

```
def store_results(conn, data, table_name):
    print(f"Storing results in {table_name} table...")
    c = conn.cursor()
    if table_name == 'basic_info':
        c.executemany('INSERT INTO basic_info VALUES (?,?,?,?,?)',
                      [(d['src_ip'], d['dst_ip'], d['protocol'], d['length'], d['timestamp']) for d in data])
    elif table_name == 'protocol_distribution':
        c.executemany('INSERT INTO protocol_distribution VALUES (?,?)', data.items())
    elif table_name == 'top_talkers':
        c.executemany('INSERT INTO top_talkers VALUES (?,?)', data)
    conn.commit()
    print(f"Results stored in {table_name} table.")
```

c.executemany: This method executes an SQL command multiple times.

'INSERT INTO basic_info VALUES (?,?,?,?,?)': This SQL command inserts values into the basic_info table.

[(d['src_ip'], d['dst_ip'], d['protocol'], d['length'], d['timestamp']) for d in data]: This list comprehension creates a list of tuples, each containing the values for src_ip, dst_ip, protocol, length, and timestamp from the data dictionary.

'INSERT INTO protocol_distribution VALUES (?,?)': This SQL command inserts values into the protocol_distribution table.

data.items(): This method returns a view object that displays a list of a dictionary's key-value tuple pairs, which are inserted into the table.

'INSERT INTO top_talkers VALUES (?,?)': This SQL command inserts values into the top_talkers table.

data: This is expected to be a list of tuples, each containing an IP address and its packet count.

The store_results function stores data in a specified table within an SQLite database. It takes three parameters: the database connection object (conn), the data to be stored (data), and the name of the table (table_name). The function uses conditional statements to determine which table to insert the data into and uses the executemany method to insert multiple rows of data.

3.2.7 display_results(conn)

The display_results function provides a user interface for viewing the stored analysis results. It allows users to select which type of information they want to view (basic info, protocol distribution, or top talkers) and presents the data in a formatted table using the tabulate library.

```
def display_results(conn):
    while True:
        print("\nSelect a table to display:")
        print("1. Basic Info")
        print("2. Protocol Distribution")
        print("3. Top Talkers")
        print("4. Return to main menu")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            table_name = 'basic_info'
        elif choice == '2':
            table_name = 'protocol_distribution'
        elif choice == '3':
            table_name = 'top_talkers'
        elif choice == '4':
            return
        else:
            print("Invalid choice. Please try again.")
            continue

        c = conn.cursor()
        c.execute(f'SELECT * FROM {table_name}')
        rows = c.fetchall()
        headers = [description[0] for description in c.description]
        print(tabulate(rows, headers=headers, tablefmt='grid'))
```

The display_results function allows the user to select and display data from different tables in an SQLite database. It presents a menu with options, takes user input, and based on the input, retrieves and displays the data from the selected table using the tabulate library for formatting.
c.execute(f'SELECT * FROM {table_name}'): This method executes an SQL command to select all rows from the specified table.
c.fetchall(): This method fetches all rows from the result of the executed SQL query and stores them in the variable rows.

headers = [description[0] for description in c.description]: This list comprehension extracts the column headers from the cursor's description attribute and stores them in the variable headers

print(tabulate(rows, headers=headers, tablefmt='grid')): This line uses the tabulate function to format and print the rows and headers in a grid format.

3.2.8 Main Menu

The main menu of the PCAP Analysis Program serves as the central hub for interacting with the various functionalities of the application. It provides users with a straightforward interface to load PCAP files, process packet data, display results, and visualize findings.

```python
def main_menu():
    capture = None
    conn = create_database()

    while True:
        print("\n===== PCAP Analysis Program =====")
        print("1. Load PCAP file")
        print("2. Process basic packet info")
        print("3. Process protocol distribution")
        print("4. Process top talkers")
        print("5. Display results")
        print("6. Visualize results")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == '1':
            file_path = input("Enter the path to the PCAP file: ")
            if capture:
                close_capture(capture)  # Close the previous capture if any
            capture = load_pcap(file_path)
        elif choice in ['2', '3', '4']:
            if capture is not None:
                if choice == '2':
                    data = process_basic_info(capture)
                    store_results(conn, data, 'basic_info')
                elif choice == '3':
                    data = process_protocol_distribution(capture)
                    store_results(conn, data, 'protocol_distribution')
                elif choice == '4':
                    data = process_top_talkers(capture)
                    store_results(conn, data, 'top_talkers')
            else:
                print("Please load a valid PCAP file first.")
        elif choice == '5':
            display_results(conn)
        elif choice == '6':
            visualize_results(conn)  # New function for visualization
        elif choice == '7':
            print("Exiting program. Goodbye!")
            close_capture(capture)
            break
        else:
            print("Invalid choice. Please try again.")

    conn.close()
```

Load PCAP file: This option allows the user to load a PCAP file for analysis. The user is prompted to enter the path to the PCAP file. If a capture is already loaded, it will be closed before loading the new file.

Process basic packet info: This option processes the basic information of the packets in the loaded PCAP file. It extracts details such as source IP, destination IP, protocol, length, and timestamp, and stores this information in the basic_info table of the SQLite database.

Process protocol distribution: This option analyzes the protocol distribution within the loaded PCAP file. It counts the occurrences of each protocol and stores the results in the protocol_distribution table of the SQLite database.

Process top talkers: This option identifies the top talkers in the network traffic captured in the PCAP file. It counts the number of packets sent and received by each IP address and stores the top talkers in the top_talkers table of the SQLite database.

Display results: This option allows the user to display the results stored in the database. The user can choose to display data from the basic_info, protocol_distribution, or top_talkers tables. The results are formatted and displayed using the tabulate library.

Visualize results: This option provides a way to visualize the results of the analysis. It is intended to call a function that generates visual representations of the data stored in the database. (using matplotlib library)

Exit: Exits the program.

The main menu continuously loops, allowing the user to perform multiple actions until they choose to exit the program.

4. PCAP File Handling and Analysis Process

4.1 Wireshark Packet Capture Fundamentals

Wireshark is a powerful network protocol analyzer that captures and displays the contents of network packets. When capturing traffic, Wireshark records detailed information about each packet, including:

- Timestamp
- Source and destination IP addresses
- Protocol information
- Packet length
- Detailed protocol-specific information

4.2 Saving PCAP Files

When saving captures from Wireshark, the software creates a PCAP (Packet Capture) file. This file format preserves all the captured packet data, including full packet contents and metadata. The PCAP file format is designed to store network traffic data efficiently, allowing for later analysis without loss of information.

Image removed

4.3 Using PCAP Files in the Program

The PCAP Analysis Program uses the pyshark library to read and process these PCAP files. When a user loads a PCAP file into the program:

1. The `load_pcap` function creates a pyshark FileCapture object.
2. This object provides an interface to iterate through each packet in the file.
3. The program can then access various attributes of each packet, such as IP addresses, protocols, and timestamps.

```
===== PCAP Analysis Program =====
1. Load PCAP file
2. Process basic packet info
3. Process protocol distribution
4. Process top talkers
5. Display results
6. Visualize results
7. Exit
Enter your choice (1-7): 1
Enter the path to the PCAP file: C:\pythonshark\pythonshark1\pcap1.pcapng
Loading PCAP file: C:\pythonshark\pythonshark1\pcap1.pcapng
PCAP file loaded successfully.
```

Loading the pcap file.

4.4 Data Extraction and Processing

```
===== PCAP Analysis Program =====
1. Load PCAP file
2. Process basic packet info
3. Process protocol distribution
4. Process top talkers
5. Display results
6. Visualize results
7. Exit
Enter your choice (1-7): 5

Select a table to display:
1. Basic Info
2. Protocol Distribution
3. Top Talkers
4. Return to main menu
Enter your choice (1-4): 2
```

The program extracts specific information from each packet:

- In process_basic_info, it pulls out source and destination IP addresses, protocol, length, and timestamp.
- process_protocol_distribution focuses on the highest layer protocol of each packet.
- process_top_talkers counts packet occurrences for each IP address.

This extracted data is then processed and aggregated to provide meaningful insights into the network traffic.

Here we can see choice 2

“Protocol Distribution”

5. Utility of Traffic Analysis

Analyzing network traffic through tools like this PCAP Analysis Program serves several important purposes in cybersecurity and network management:

1. **Security Monitoring**: Identifying unusual patterns or unexpected protocols can help detect potential security threats or breaches.
2. **Network Performance**: Understanding protocol distribution and top talkers can highlight bottlenecks or inefficiencies in network usage.
3. **Compliance**: For industries with strict data handling regulations, this analysis can help ensure that network traffic complies with required standards.
4. **Troubleshooting**: When network issues occur, having detailed packet information can be crucial in identifying the root cause.
5. **Capacity Planning**: By understanding traffic patterns, network administrators can make informed decisions about infrastructure upgrades or changes.

6. Information Displayed in Tables

6.1 Basic Info Table

The tables created using tabulate provide detailed information on individual packets, including:

- Source IP: The originating IP address of the packet.
- Destination IP: The intended recipient's IP address.
- Protocol: The transport layer protocol used (e.g., TCP, UDP).
- Length: The size of the packet in bytes.
- Timestamp: The exact time the packet was captured.

Image removed

This information is used for understanding the flow of traffic between different network entities or hosts.
It appears there is some error in the timestamp column of the table.

6.2 Protocol Distribution Table

```
Select a table to display:
1. Basic Info
2. Protocol Distribution
3. Top Talkers
4. Return to main menu
Enter your choice (1-4): 2
+--------------+---------+
| protocol     |   count |
+==============+=========+
| DATA         |     339 |
+--------------+---------+
| RTCP         |     122 |
+--------------+---------+
| TLS          |      29 |
+--------------+---------+
| TCP          |      31 |
+--------------+---------+
| _WS.MALFORMED |       5 |
+--------------+---------+
| STUN         |      48 |
```

This table shows the frequency of different protocols in the captured traffic:

- Protocol: The name of the protocol

  (e.g., HTTP, DNS, TLS).

- Count: The number of packets using this protocol.

This distribution helps in understanding the types of applications and services being used on the network, which is valuable for security analysis and network optimization.

6.3 Top Talkers Table

The Top Talkers table identifies the most active IP addresses in the network:

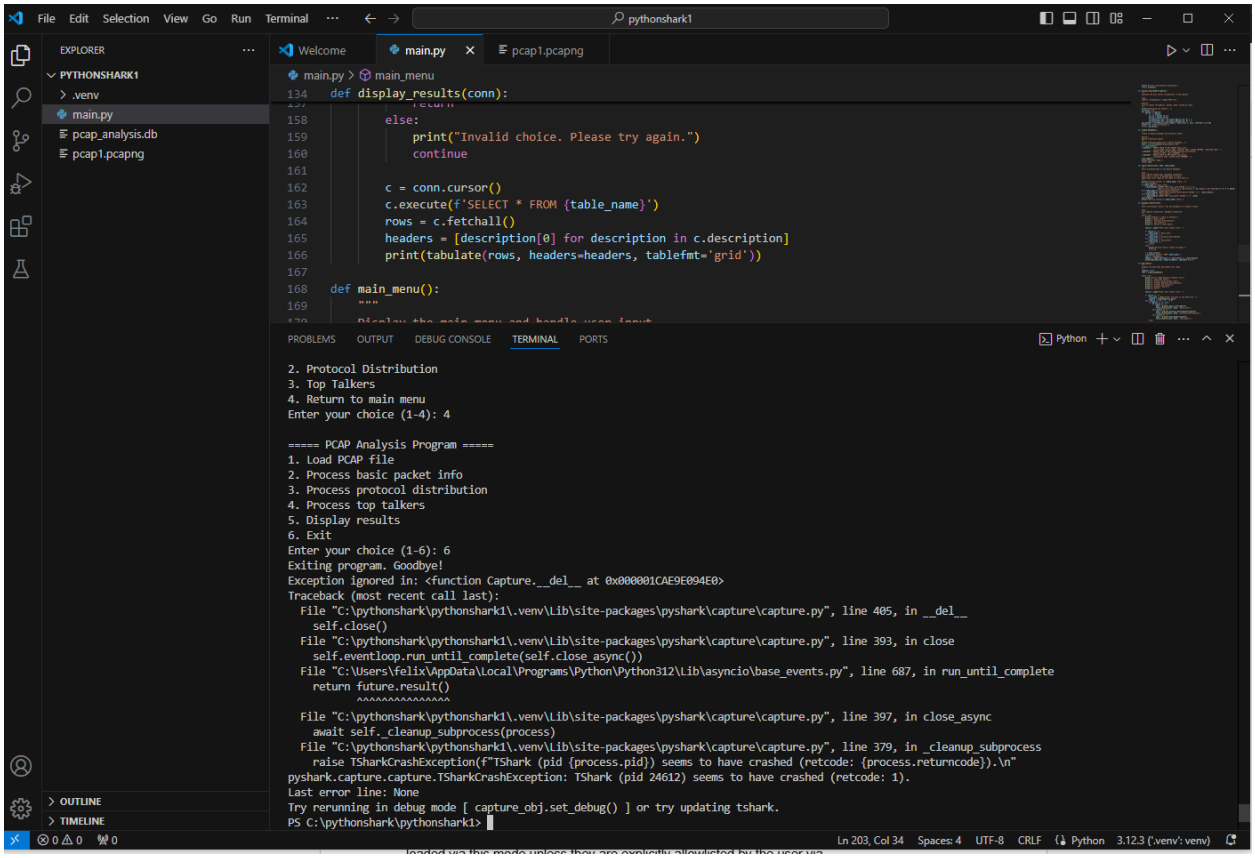
- IP Address: The IP address of the network entity.
- Packet Count: The total number of packets associated with this IP address.

This information is used for identifying potential security issues, such as data exfiltration attempts or compromised systems, as well as understanding which systems are generating the most network traffic.

Image removed

7. Debug

Bug Description: Upon exiting the PCAP Analysis Program, the following exception is observed in the terminal output:



This bug identified in the program causes an exception to be raised when the program exits. The error occurs specifically during the cleanup process when the program attempts to close the TShark subprocess utilized by the `pyshark` library. This issue results in an uncaught exception being printed to the terminal, indicating that TShark has crashed.

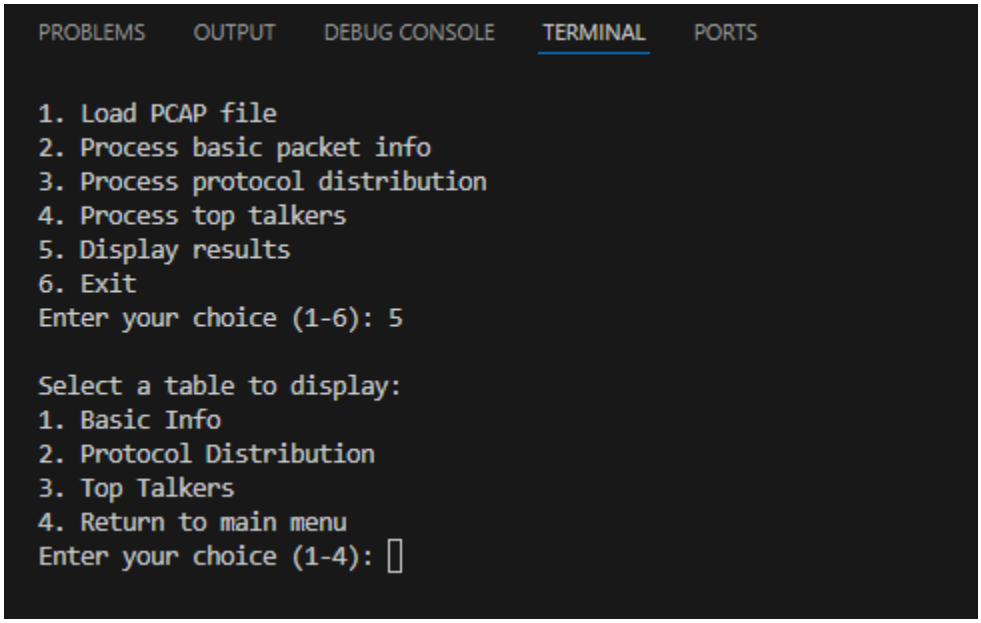
Resolution: Ensure the connection to the sql database is closed when the user exits the program.

```
        elif choice == '7':
            print("Exiting program. Goodbye!")
            close_capture(capture) ##close the connection before ending program.
            break
        else:
            print("Invalid choice. Please try again.")
```

```python
## closes the capture object to free up resources
def close_capture(capture):
    """Close the capture object to clean up resources."""
    if capture:
        try:
            print("Closing capture object...")
            capture.close()
        except Exception as e:
            print(f"Error closing capture: {e}")
```

Older versions:

When selecting menu function 5 to display results the program does not display results and the program displays the menu.



To resolve this issue, add an elif statement for choice 5 that calls the display_results() function:

Realization: The program does in fact display the results as intended , it is just a sub menu. I did have a similar problem and thought it was a recurring problem. Upon investigation I realized the program was actually functioning correctly. I simply did not read the output correctly.

User Experience Issue

While the program is functioning correctly, the user interface could be improved to make the flow of information more clear and intuitive. The transition from the main menu to the submenu for displaying results, and then back to the main menu, may not be sufficiently distinct for all users.

This lack of clarity could lead to problems like user confusion about whether their input was registered. Misinterpretation of program behavior. Potential overlooking of important data or options.

To improve this programs user experience we could enhanced Visual Separation:

Add clear visual separators between different menu levels and outputs. For example:

==========

| PCAP Analysis Program |

==========

Improving the User interface and adding a GUI would be valuable additions to the program.

8. Improvements and Afterthoughts

Currently, the program has separate menu options for processing basic packet info, protocol distribution, and top talkers. To streamline the user experience and improve efficiency, these functions could be consolidated into a single "Process PCAP file" option. This would allow for a more comprehensive analysis in a single step, reducing the need for multiple user interactions.

```python
def main_menu():
    capture = None
    conn = create_database()

    while True:
        print("\n===== PCAP Analysis Program =====")
        print("1. Load PCAP file")
        print("2. Process basic packet info")
        print("3. Process protocol distribution")
        print("4. Process top talkers")
        print("5. Display results")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ")
```

Options 2-3-4 can be condensed into a single operation.

The consolidation could be achieved by creating a new function that calls each of the existing processing functions sequentially and returns their results. This new function could then be invoked from a single menu option.

Additional Data Extraction

To enhance the program's analytical capabilities, we could add functions to extract more detailed information from the PCAP file. Some potential additions include:

HTTP Request Analysis: Implement a function to extract and analyze HTTP requests, including methods, URLs, and headers. This would provide insights into web traffic patterns and potential security issues.
DNS Query Analysis: Create a function to identify and analyze DNS queries and responses. This could help in detecting unusual domain lookups or potential DNS-based attacks.

SSL/TLS Certificate Information: Develop functionality to extract and display SSL/TLS certificate details from encrypted traffic. This would be valuable for identifying potentially malicious or misconfigured secure connections.

Security

To enhance the security of the SQLite database, we could implement measures to prevent SQL injection attacks. This could be achieved by modifying the store_results function to use parameterized queries instead of string concatenation when constructing SQL statements. Additionally, input validation could be implemented to ensure that table names and other user inputs are sanitized before being used in database operations. This would help protect against potential SQL injection vulnerabilities.

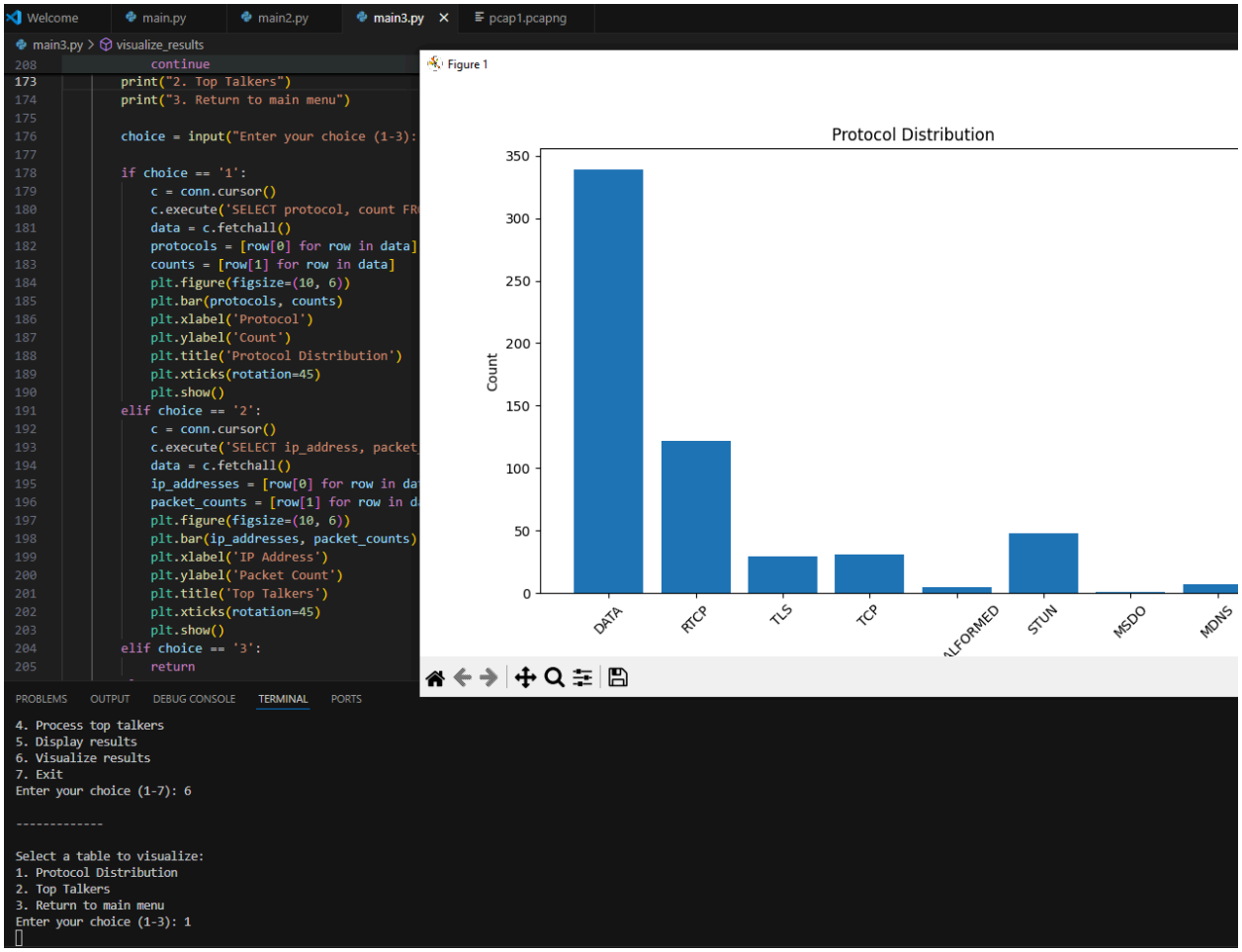
Visualising Data

Adding simple data visualization capabilities would enhance the program's ability to convey insights. This could be achieved by integrating a plotting library like matplotlib. Potential visualizations could include:

Bar charts showing protocol distribution.
Pie charts illustrating the proportion of traffic from top talkers.
Time series graphs displaying packet frequency over time.

These visualizations could be generated automatically after processing the PCAP file and saved as image files for easy viewing and sharing.

Here is a basic implementation of matplotlib graph showing protocol distribution.



User Interface

The current command-line interface of the PCAP Analysis Program, while functional, may present challenges for users less familiar with terminal-based applications. Implementing a Graphical User Interface (GUI) could significantly enhance the user experience, making the program more accessible.

Potential GUI features:

- File Browser: Implement a file browser dialog for easy selection of PCAP files, eliminating the need for users to manually type or paste file paths.
- Results Dashboard: Create a dashboard-style interface to display analysis results, including basic packet information, protocol distribution, and top talkers.
- Interactive Visualizations: Incorporate interactive charts and graphs that users can manipulate to explore data in more depth.
- Configuration Panel: Develop a settings panel where users can customize analysis parameters, output formats, and other program options.

- Export Functionality: Add buttons or menu items to easily export analysis results in various formats (e.g., CSV, PDF, or images of visualizations).

Adding a GUI would significantly increase the complexity of the project, potentially making it more difficult to maintain and update.

9. Conclusion

The program provides a solid foundation for analyzing network traffic, but some improvements would make it more useful. By combining menu options that perform similar tasks, such as the various processing steps, the user experience could be streamlined, reducing repetitive actions. Additionally, the program could benefit from more advanced analysis features, such as breaking down HTTP or DNS traffic, to offer a clearer view of potential security threats.

The program's modular structure and use of SQLite for data storage allows for easy maintenance and potential future expansions. The ability to process large volumes of packet data and present it in an easily digestible format makes this tool valuable for network administrators, security professionals, and students learning about network protocols and analysis.

Future enhancements to the program could include more advanced analysis features, graphical visualizations of the data, and integration with other security tools. As it stands, this program serves as a foundation for packet analysis and demonstrates the power of Python scripting in handling complex networking tasks.

10. User Guide

PCAP Analysis Program

This is a Python-based command-line tool designed to process and analyze PCAP files captured from network traffic. The program provides basic packet information, protocol distribution, and top talkers, storing the results in an SQLite database for easy retrieval and display.

Features
- **Load PCAP Files**: Analyze network traffic captured in Wireshark PCAP format.
- **Basic Packet Information**: View details like source IP, destination IP, protocol, length, and timestamp for each packet.
- **Protocol Distribution**: See the breakdown of different protocols used in the network traffic.
- **Top Talkers**: Identify the most active IP addresses in the captured network data.
- **Database Storage**: Results are stored in an SQLite database for future access and analysis.

- **Display Results**: View the processed results in neatly formatted tables using the Tabulate library.

Prerequisites
Before running the program, ensure the following Python libraries are installed:
- `pyshark` (for reading PCAP files)
- `sqlite3` (for database management)
- `tabulate` (for formatted table display)

You can install these libraries using `pip`:

pip install pyshark sqlite3 tabulate

11. Sources

16: Analyzing capture files in Python with PyShark (youtube.com)
https://www.youtube.com/watch?v=qxBKHsMhWKU

Sqlite: https://docs.python.org/3/library/sqlite3.html
https://github.com/python/cpython/tree/3.12/Lib/sqlite3/

Tabulate: https://github.com/astanin/python-tabulate

Wireshark: https://github.com/wireshark/wireshark/tree/master/doc

Pyshark: https://github.com/KimiNewt/pyshark

Sql injection:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html