

NETWORK TRAFFIC ANALYSIS

Table of Contents :

- Introduction
- Methodology
- Sample Code
 - Packet Sniffing
 - Visualization
 - Packet Analysis
 - Select The Packet Type
- Execution
- Conclusion
- References

Introduction :

Network traffic analysis is a crucial task for understanding network behavior, identifying potential security threats, and optimizing network performance. This project aims to develop a tool that captures network packets, analyzes their characteristics, and visualizes the traffic patterns. The tool allows users to filter packets based on various criteria and provides insights into the network traffic.

Methodology :

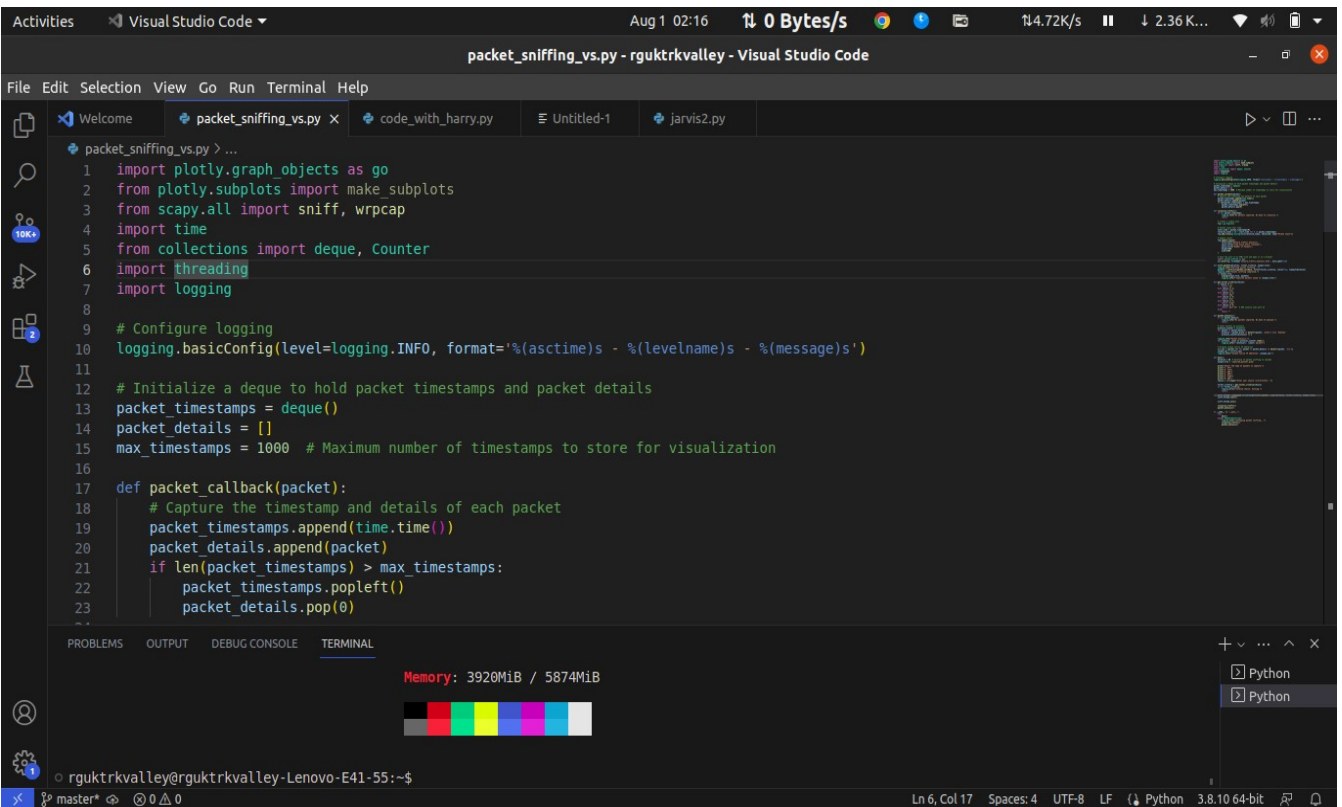
The project utilizes the following libraries and techniques:

1. **Scapy:** A powerful Python library for network packet manipulation and analysis. It is used for capturing and filtering network packets.
2. **Plotly:** A data visualization library for creating interactive and customizable plots. It is used to visualize the captured network traffic.
3. **Deque and Counter:** Data structures from the `collections` module used for efficient storage and analysis of packet timestamps and details.
4. **Multithreading:** Allows concurrent execution of packet sniffing and visualization to provide a responsive user experience.
5. **Logging:** Enables structured logging for informative output and error handling during the execution of the program.

* Sample Code:

• Packet Sniffing :

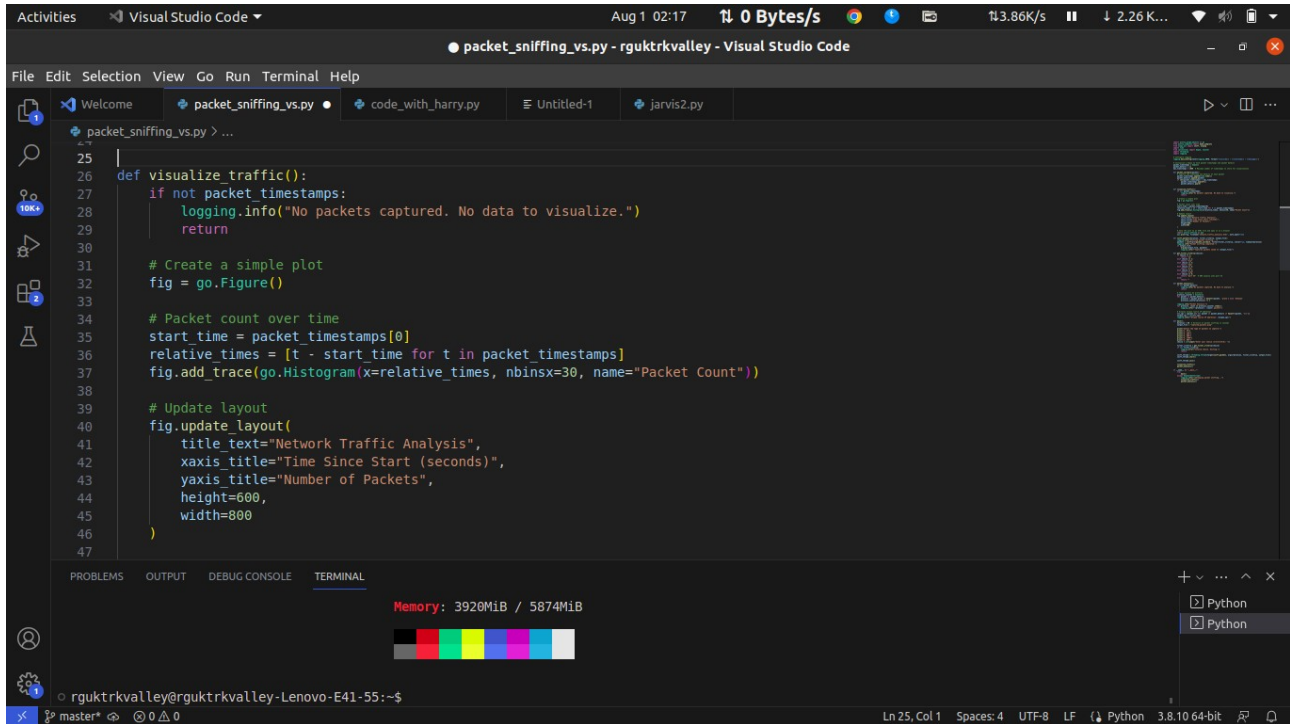
The `sniff_packets` function captures packets based on user-defined filters and saves them to a file. The `packet_callback` function is used to process each packet, capturing its timestamp and details.



```
packet_sniffing_vs.py > ...
1 import plotly.graph_objects as go
2 from plotly.subplots import make_subplots
3 from scapy.all import sniff, wrpcap
4 import time
5 from collections import deque, Counter
6 import threading
7 import logging
8
9 # Configure logging
10 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
11
12 # Initialize a deque to hold packet timestamps and packet details
13 packet_timestamps = deque()
14 packet_details = []
15 max_timestamps = 1000 # Maximum number of timestamps to store for visualization
16
17 def packet_callback(packet):
18     # Capture the timestamp and details of each packet
19     packet_timestamps.append(time.time())
20     packet_details.append(packet)
21     if len(packet_timestamps) > max_timestamps:
22         packet_timestamps.popleft()
23         packet_details.pop(0)
```

• Visualization :

The `visualize_traffic` function creates a histogram of packet counts over time using Plotly. This visual representation helps in understanding the traffic pattern.



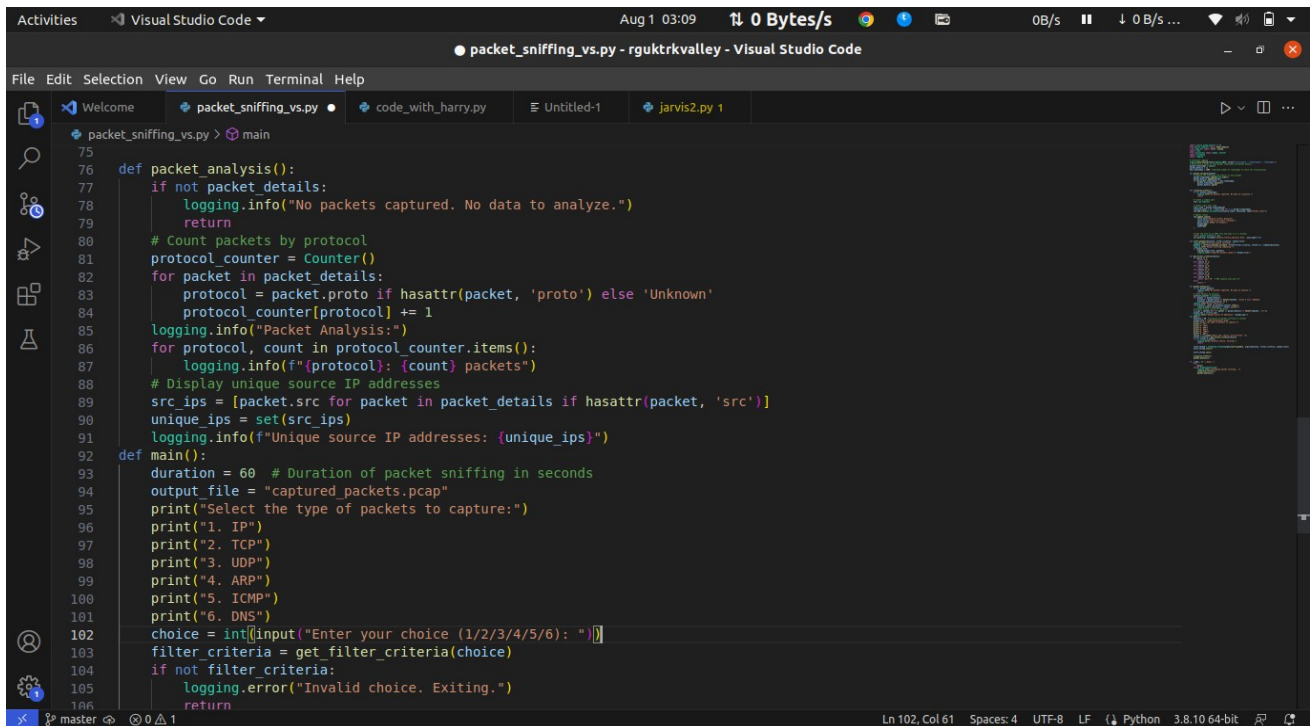
```
25 |
26 | def visualize_traffic():
27 |     if not packet_timestamps:
28 |         logging.info("No packets captured. No data to visualize.")
29 |         return
30 |
31 |     # Create a simple plot
32 |     fig = go.Figure()
33 |
34 |     # Packet count over time
35 |     start_time = packet_timestamps[0]
36 |     relative_times = [t - start_time for t in packet_timestamps]
37 |     fig.add_trace(go.Histogram(x=relative_times, nbinsx=30, name="Packet Count"))
38 |
39 |     # Update layout
40 |     fig.update_layout(
41 |         title_text="Network Traffic Analysis",
42 |         xaxis_title="Time Since Start (seconds)",
43 |         yaxis_title="Number of Packets",
44 |         height=600,
45 |         width=800
46 |     )
47 |
```

Memory: 3920MiB / 5874MiB

rguktrkvalley@rguktrkvalley-Lenovo-E41-55:~\$

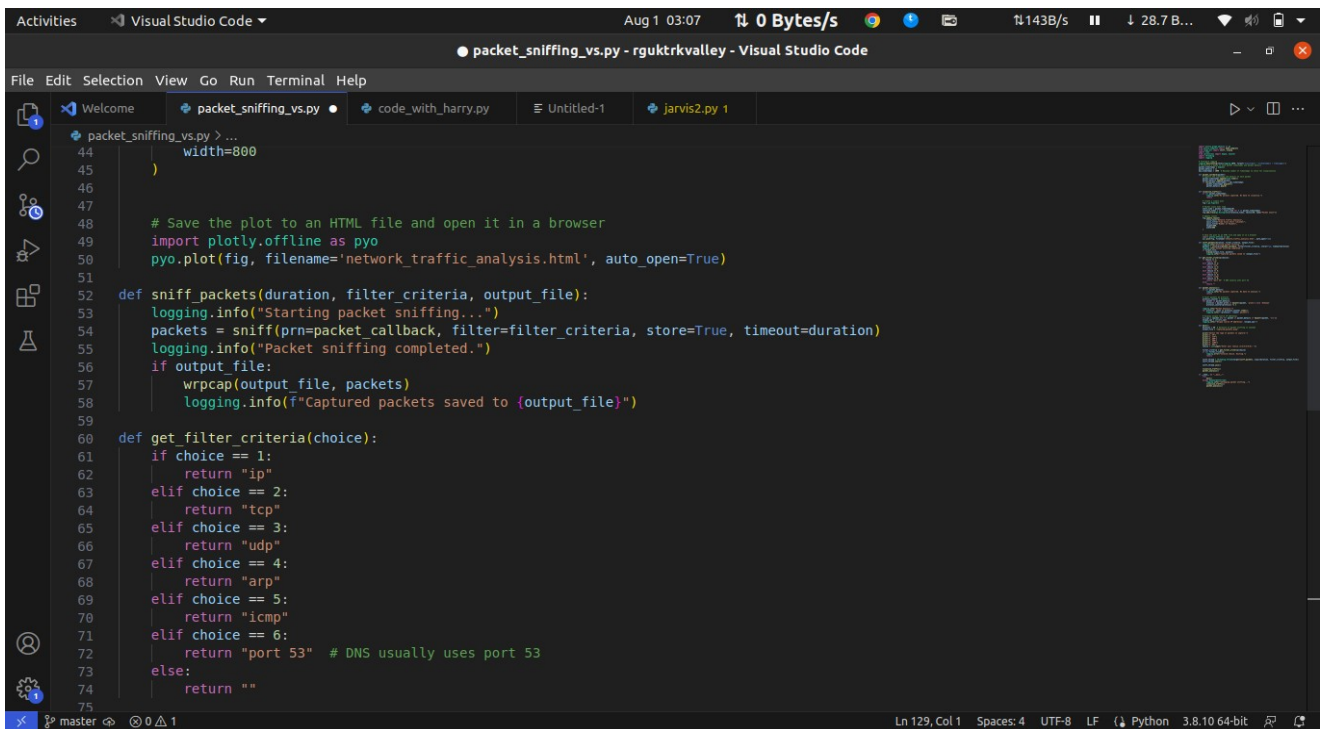
• Packet Analysis :

The `packet_analysis` function performs a basic analysis of the captured packets, counting packets by protocol and listing unique source IP addresses.

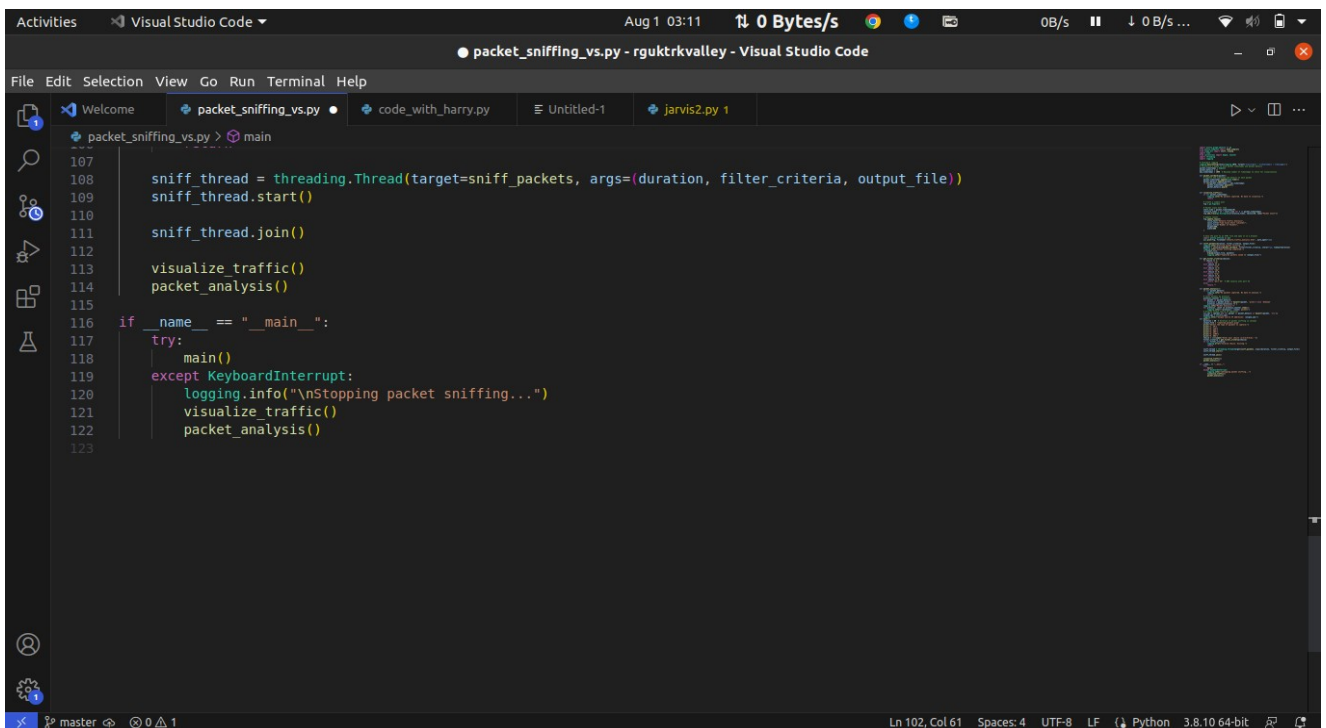


```
75
76 def packet_analysis():
77     if not packet_details:
78         logging.info("No packets captured. No data to analyze.")
79         return
80     # Count packets by protocol
81     protocol_counter = Counter()
82     for packet in packet_details:
83         protocol = packet.proto if hasattr(packet, 'proto') else 'Unknown'
84         protocol_counter[protocol] += 1
85     logging.info("Packet Analysis:")
86     for protocol, count in protocol_counter.items():
87         logging.info(f"{protocol}: {count} packets")
88     # Display unique source IP addresses
89     src_ips = [packet.src for packet in packet_details if hasattr(packet, 'src')]
90     unique_ips = set(src_ips)
91     logging.info(f"Unique source IP addresses: {unique_ips}")
92 def main():
93     duration = 60 # Duration of packet sniffing in seconds
94     output_file = "captured_packets.pcap"
95     print("Select the type of packets to capture:")
96     print("1. IP")
97     print("2. TCP")
98     print("3. UDP")
99     print("4. ARP")
100    print("5. ICMP")
101    print("6. DNS")
102    choice = int(input("Enter your choice (1/2/3/4/5/6): "))
103    filter_criteria = get_filter_criteria(choice)
104    if not filter_criteria:
105        logging.error("Invalid choice. Exiting.")
106        return
```

*** Select the Packet Type:** When prompted, choose the type of packets you want to capture (e.g., IP, TCP, UDP, ARP, ICMP, DNS).



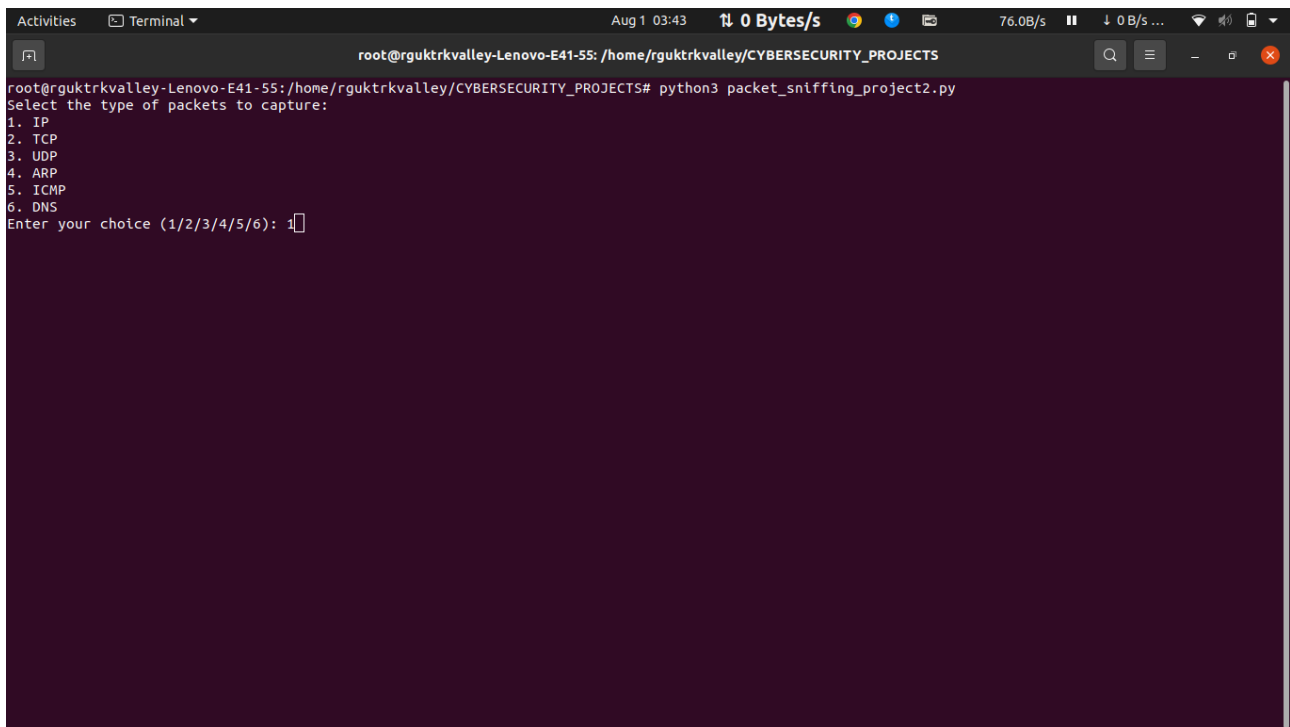
```
44         width=800
45     )
46
47     # Save the plot to an HTML file and open it in a browser
48     import plotly.offline as pyo
49     pyo.plot(fig, filename='network_traffic_analysis.html', auto_open=True)
50
51
52 def sniff_packets(duration, filter_criteria, output_file):
53     logging.info("Starting packet sniffing...")
54     packets = sniff(prn=packet_callback, filter=filter_criteria, store=True, timeout=duration)
55     logging.info("Packet sniffing completed.")
56     if output_file:
57         wrpcap(output_file, packets)
58         logging.info(f"Captured packets saved to {output_file}")
59
60 def get_filter_criteria(choice):
61     if choice == 1:
62         return "ip"
63     elif choice == 2:
64         return "tcp"
65     elif choice == 3:
66         return "udp"
67     elif choice == 4:
68         return "arp"
69     elif choice == 5:
70         return "icmp"
71     elif choice == 6:
72         return "port 53" # DNS usually uses port 53
73     else:
74         return ""
75
```



```
107     sniff_thread = threading.Thread(target=sniff_packets, args=(duration, filter_criteria, output_file))
108     sniff_thread.start()
109
110     sniff_thread.join()
111
112     visualize_traffic()
113     packet_analysis()
114
115
116 if __name__ == "__main__":
117     try:
118         main()
119     except KeyboardInterrupt:
120         logging.info("\nStopping packet sniffing...")
121         visualize_traffic()
122         packet_analysis()
123
```

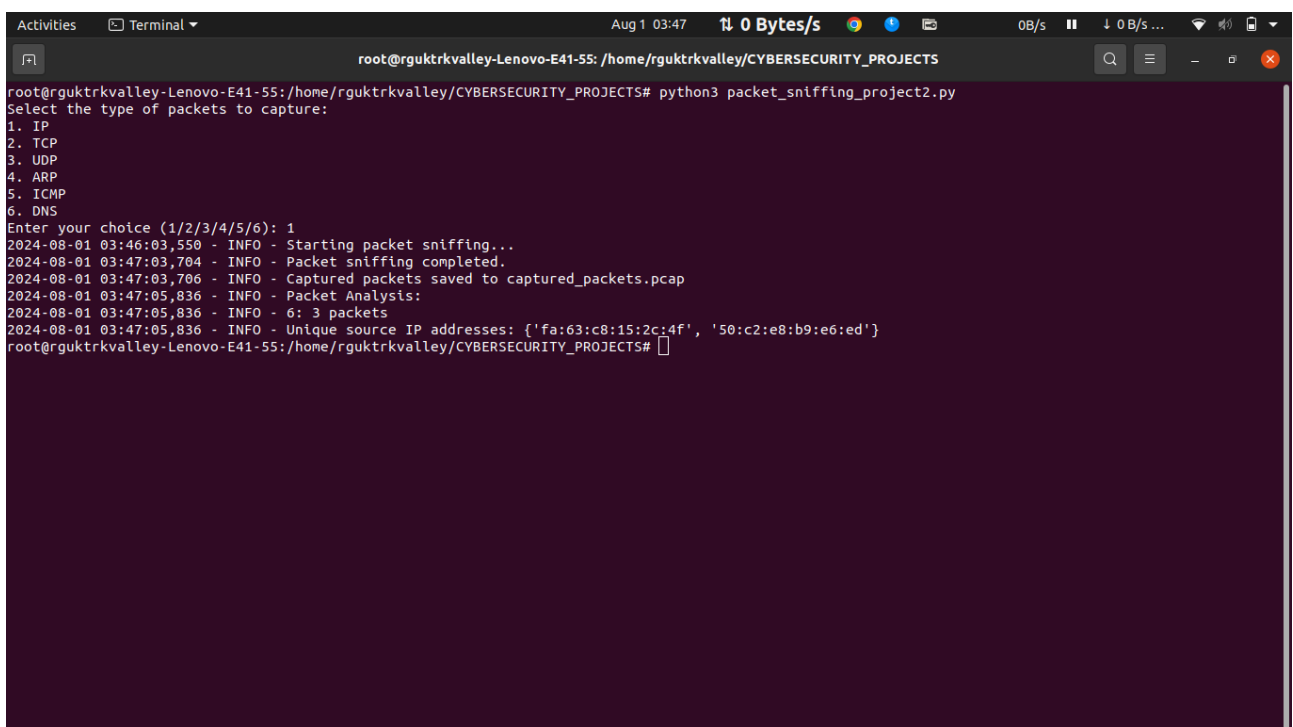
* Execution :

* When the code is executed we will need to choose the type of packets we need to capture .



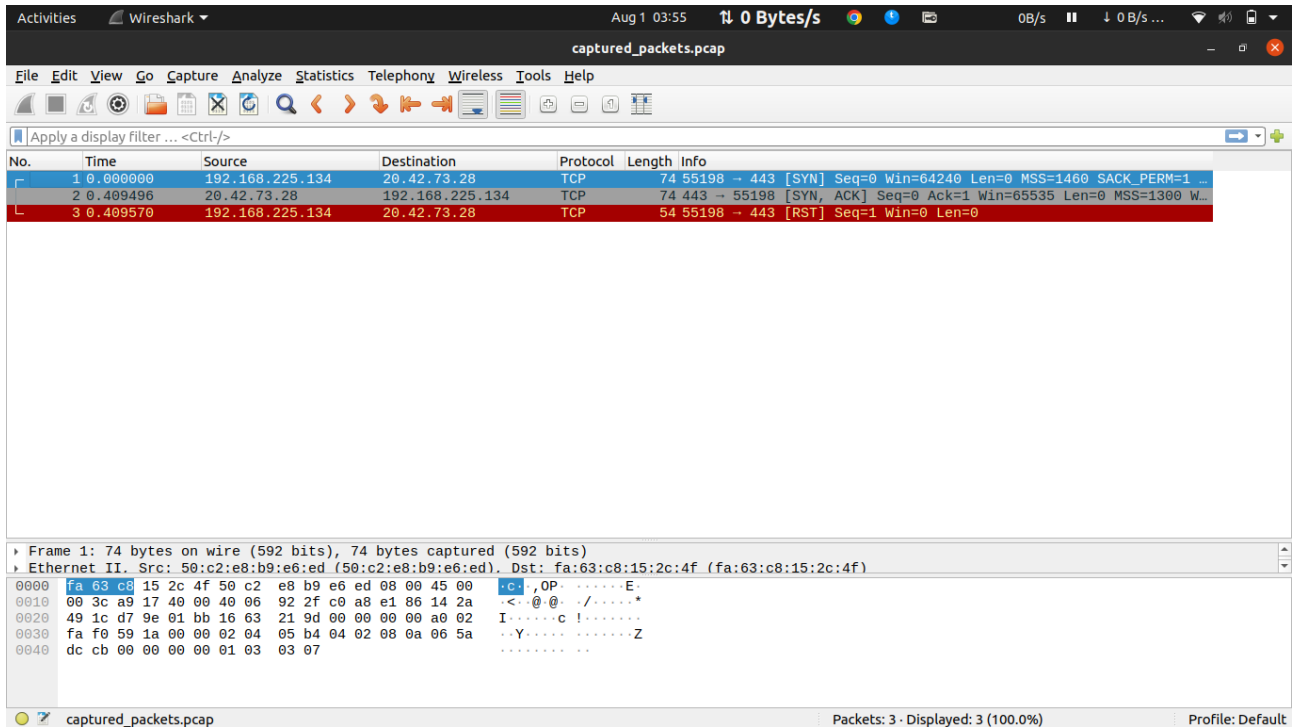
```
root@rguktrkvalley-Lenovo-E41-55: /home/rguktrkvalley/CYBERSECURITY_PROJECTS
root@rguktrkvalley-Lenovo-E41-55:/home/rguktrkvalley/CYBERSECURITY_PROJECTS# python3 packet_sniffing_project2.py
Select the type of packets to capture:
1. IP
2. TCP
3. UDP
4. ARP
5. ICMP
6. DNS
Enter your choice (1/2/3/4/5/6): 1
```

* In the above example I have taken the input as “1” where we will collect the packets of “IP”.

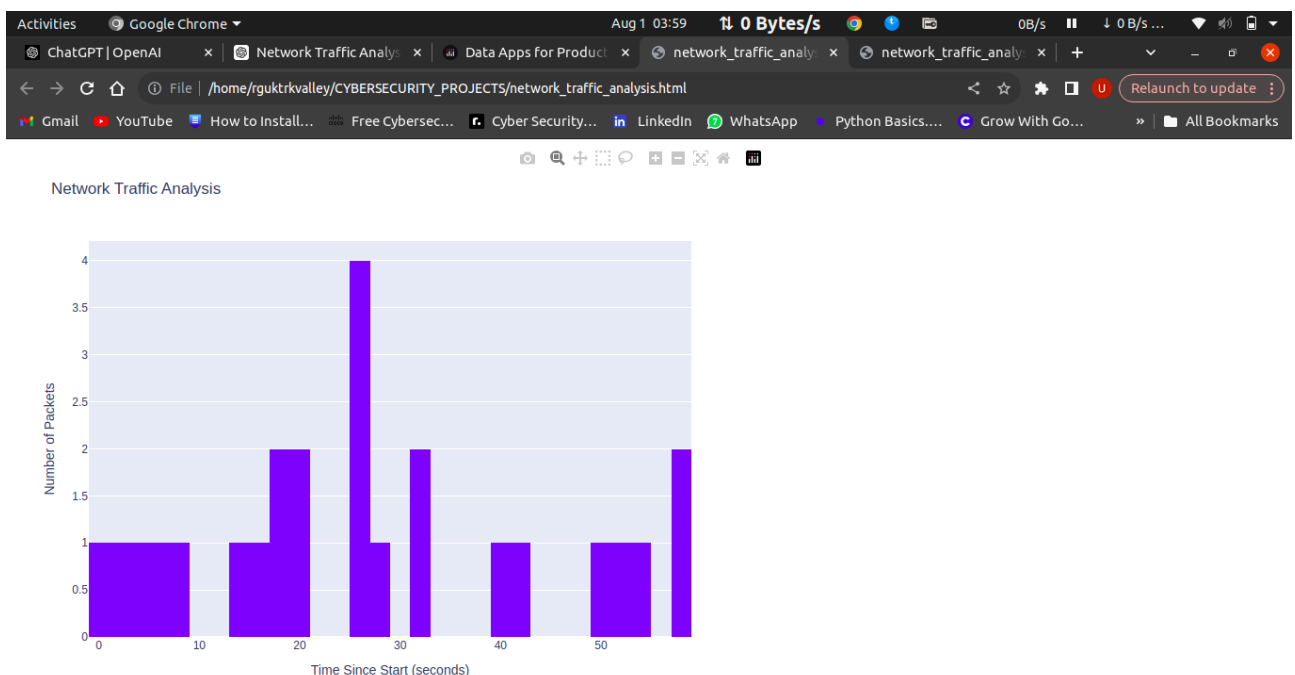


```
root@rguktrkvalley-Lenovo-E41-55: /home/rguktrkvalley/CYBERSECURITY_PROJECTS
root@rguktrkvalley-Lenovo-E41-55:/home/rguktrkvalley/CYBERSECURITY_PROJECTS# python3 packet_sniffing_project2.py
Select the type of packets to capture:
1. IP
2. TCP
3. UDP
4. ARP
5. ICMP
6. DNS
Enter your choice (1/2/3/4/5/6): 1
2024-08-01 03:46:03,550 - INFO - Starting packet sniffing...
2024-08-01 03:47:03,704 - INFO - Packet sniffing completed.
2024-08-01 03:47:03,706 - INFO - Captured packets saved to captured_packets.pcap
2024-08-01 03:47:05,836 - INFO - Packet Analysis:
2024-08-01 03:47:05,836 - INFO - 6: 3 packets
2024-08-01 03:47:05,836 - INFO - Unique source IP addresses: {'fa:63:c8:15:2c:4f', '50:c2:e8:b9:e6:ed'}
root@rguktrkvalley-Lenovo-E41-55:/home/rguktrkvalley/CYBERSECURITY_PROJECTS#
```

* As we can see the packets that are captured will be stored in “captured_packets.pcap”(The file can be viewed in Wireshark).



* The Visualization of the packets can be seen in the “network_traffic_analysis.html” file .



* Similarly we can capture the packets of all the other protocols giving their respective input.

*** Conclusion :**

This project provides a basic framework for capturing, visualizing, and analyzing network traffic. Future improvements could include real-time visualization updates, more advanced packet analysis features, and integration with other data sources.

*** References :**

- Plotly Documentation:
<https://plotly.com/python/>
- Scapy Documentation:
<https://scapy.readthedocs.io/>