ABSTRACT

Packet sniffer is a tool used to intercept and analyze network traffic, capturing individual data packets as they traverse a network. This technique allows users, whether administrators or malicious actors, to monitor and potentially access data intended for other network users. In a hub-based network, where all devices receive all network traffic, packet sniffing is straightforward since packets are broadcast to all connected devices. However, in a switched network, where packets are selectively delivered only to their intended destinations, packet sniffing requires more advanced methods like ARP spoofing or port mirroring to capture traffic not directly destined for the sniffer's device. To counteract packet sniffing, network administrators employ AntiSniff techniques, such as traffic analysis and detection of known packet sniffing tools, to safeguard network security and privacy.

Table of Contents

LISTS OF FIGURES

CHAPTER 1.

INTRODUCTION

A tool called a Packet Sniffer can monitor every piece of data traveling through its linked network. It is essentially akin to wiretapping, as it can be inserted into computer networks and covertly monitor their information flow.

Packet sniffers, or network analyzers, are tools that capture and analyze network traffic. They operate by intercepting data packets as they traverse a network, decoding them to reveal their contents. Historically, Ethernet networks utilized a bus topology with coaxial or twisted pair cables, allowing devices to communicate over shared wires. Data transmission followed the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol, akin to waiting for a turn to speak at a noisy party.

Each network node possesses a unique Media Access Control (MAC) address, typically only examining packets meant for its address. However, in promiscuous mode, a network card examines all traffic, enabling the sniffer to monitor all data packets on the network. This feature is crucial for Intrusion Detection Systems (IDS), which detect and report network intrusions, playing a vital role in an organization's information security policy.

Packet sniffers are essential for network security, providing real-time monitoring [1], 24/7 surveillance, advanced protocol analysis, and in-depth packet decoding. They facilitate a comprehensive view of network activities, allowing for detailed packet-level scrutiny and diagnostics, ensuring the security and integrity of digital communications.

1.1 Objective of the Study

The objectives of studying packet sniffers are:

- **Network Monitoring:** To provide real-time surveillance of network traffic, ensuring all data packets are properly captured and analyzed.
- **Troubleshooting:** To identify and resolve network issues by analyzing packet data, which can reveal the source of connectivity problems.
- **Security:** To detect and prevent security threats by monitoring for suspicious or malicious activity within the network traffic.
- **Performance Optimization:** To assess network performance and identify potential bottlenecks, ensuring efficient data transmission and network operations.

1.2 Future Research

The project concludes with a summary of the findings and some suggestions for future research**.**

- Research the type of packet sniffer that you need for your particular situation. Make sure it will meet your needs and be compatible with your system.
- Configure the packet sniffer according to your specific needs. This may involve setting up filters, enabling logging, or other tasks.
- Monitor the network activity using the packet sniffer to detect any suspicious activity or potential security threats.
- Analyze the data collected by the packet sniffer. This may require additional software or special tools to interpret the data.
- Create a report containing the results of the analysis and any recommendations for further action.

CHAPTER 2.
LITERATURE REVIEW

2.1 Introduction

This chapter discusses the background literature associated with the issues of the design and implementation of effective honeypots and honeynets.

2.2 Previous Research

Pallavi Asrodia et al [2] in their paper have given more focus on the basics of packet sniffing. They have given the knowledge about the various principles on which the packet sniffer works, what are different approaches, and how these approaches work. They have focused on the working principles of the packet sniffer which is used for the analysis of network traffic.

Dr. Dayanand Lal N et al [3] in their paper have focused on the development of a security tool named 'Secret Credentials Packet Sniffer. All the secret credentials flowing into the network are sniffed by this security tool. Secret Credentials include mainly usernames, passwords, cookies, etc. They have used various networking protocols where packet sniffers are used.

Ogbu N. Henry et al [4] in their paper they have used a LAN packet sniffer to observe the network traffic for internet security. They have provided internet traffic monitoring using a sniffer at the Local Area Network servers for protection purposes. They have used static Internet Protocol (IP) for implementation. The ethernet cable of category 6, the Dlink 16-port switch of windows 8, and some other Local Area Network devices were used to deploy the Local Area Network. For analyzing and capturing the IP traffic they have used Wireshark.

Wireshark et al [5] is a protocol analyzer that helps us supervise our network at a microscopic level. It gives information on many protocols and the protocols are still being added every day. It helps us to inspect the packet in real-time and is also multiplatform, i.e., it can run on Windows, macOS, Linux, Solaris, NetBSD, and many other Operating Systems. It is user-friendly and provides a GUI to look at the captured data. Decryption support is provided for various protocols, like IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2 It allows us to apply coloring rules to the packet list for quick, intuitive analysis. This paper reviews the existing literature on packet sniffers and their use in network security. Packet sniffers are programs that capture, analyze, and reconstruct network traffic on a local area network (LAN) or a wide area network (WAN). Packet sniffers can be used for malicious purposes, such as stealing data or launching attacks. However, these programs can also be used for legitimate purposes, such as network monitoring and troubleshooting. The paper begins with a discussion of the core features of packet sniffers and their use in network security.

CHAPTER 3.
SNIFFING REQUIRMENTS

3.1 Requirement Analysis

The project requires Python 3 to execute the backend .py file, ensuring the packet sniffer's core functionality. Node.js is essential for developing the user interface, which will display real-time packet capture results on a web platform. For analyzing the captured packets, Wireshark is necessary, providing in-depth inspection and diagnostic capabilities. Together, these technologies form the backbone of the packet sniffer, enabling efficient packet capturing, real-time data presentation, and comprehensive analysis for network monitoring and security.

3.2 Protocols Used to Sniffing

- **TCP**

TCP, or Transmission Control Protocol, is a core network protocol that ensures reliable data transmission between devices. In packet sniffing, TCP packets are intercepted and analyzed to maintain data integrity and order. TCP's three-way handshake—SYN, SYN-ACK, ACK—establishes a connection, allowing data exchange [6]. Sniffers capture these TCP packets, using tools like Wireshark for real-time analysis, aiding in network troubleshooting and security. TCP's error-checking mechanisms and sequence control make it integral to packet sniffing, providing a detailed view of network communication for monitoring and diagnostics.

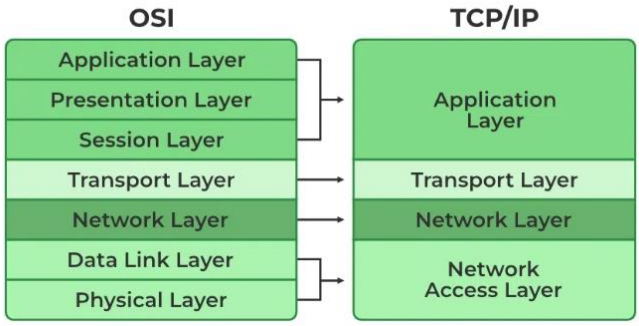


Fig.1: TCP Diagram

Fig.1: Shows the working principle of TCP.

- **IPV4**

IPv4, or Internet Protocol version 4, is the foundational protocol for routing traffic across the internet. In packet sniffing, IPv4 packets are intercepted to analyze network traffic [7]. Each packet contains source and destination IP addresses, allowing sniffers to track where data is coming from and going to. This is crucial for network diagnostics and security, as it helps identify potential threats and performance issues. By examining these packets, administrators can ensure network integrity and efficiency, making IPv4 a

key element in maintaining a secure and stable digital environment.

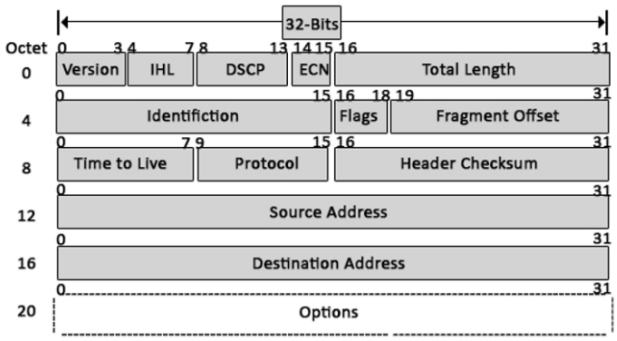


Fig.2: IPV4 Diagram

Fig.2: Shows the working principle of IPV4.

- **ICMP**

ICMP, or Internet Control Message Protocol [8], is used for sending error messages and operational information in IP networks. In packet sniffing, ICMP packets are captured to diagnose network issues. These packets, which include types like Echo Request and Echo Reply, are used by tools like ping to check network connectivity. By analyzing ICMP traffic, network administrators can detect and troubleshoot problems such as unreachable hosts or network congestion, making ICMP a valuable tool for maintaining network health and performance.

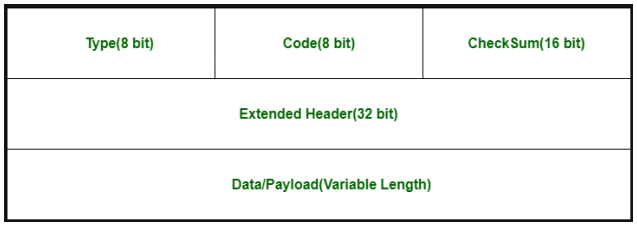


Fig.3: ICMP Diagram

Fig.3: Shows the working principle of ICMP.

- **UDP**

UDP, or User Datagram Protocol [9], is a connectionless protocol used for sending data without establishing a reliable connection. In packet sniffing, UDP packets are captured to analyze network traffic. Unlike TCP, UDP does not guarantee delivery, making it faster but less reliable. Sniffers capture these packets to monitor network activity, troubleshoot issues, and detect security threats. By analyzing UDP traffic, network administrators can gain insights into the flow of data and ensure the network's efficient operation, despite UDP's lack of error-checking and recovery features.

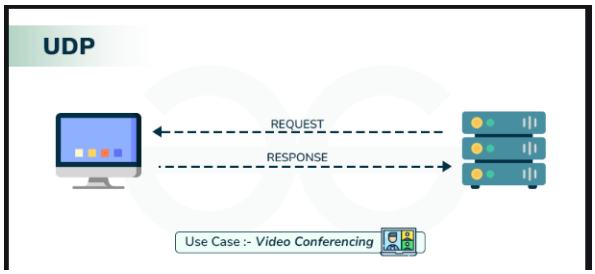


Fig.4: UDP Diagram

Fig.4: Shows the working principle of UDP.

- **HTTP**

HTTP, or Hypertext Transfer Protocol, is the foundation for data communication on the web [10]. In packet sniffing, HTTP packets are captured to analyze web traffic. This protocol operates at the application layer, transferring data between web servers and clients. Sniffers intercept these HTTP packets to monitor and log web activity, useful for network troubleshooting and security assessments. Since HTTP is not encrypted, sniffers can easily read the data, making it crucial to use HTTPS for secure communication to protect against potential eavesdropping in packet sniffing scenarios.

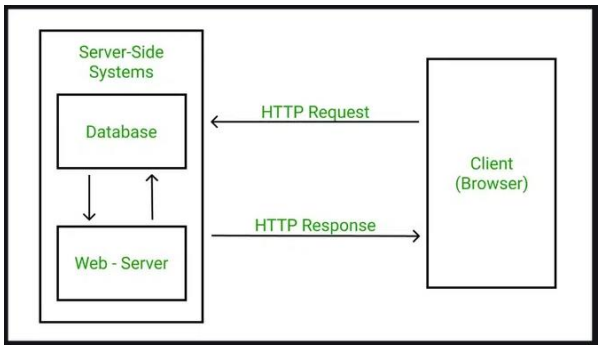


Fig.5: HTTP Diagram

Fig.5: Shows the working principle of HTTP.

- **FTP**

FTP, or File Transfer Protocol, is used for transferring files across a network [11]. In packet sniffing, FTP traffic can be intercepted to analyze file transfers. Since FTP does not encrypt data, credentials and file contents can be captured in plain text by sniffers. This makes FTP vulnerable to eavesdropping, highlighting the importance of using secure alternatives like FTPS or SFTP for encrypted transfers, which protect against sniffing attacks by obscuring data from unauthorized viewers. Understanding and mitigating FTP's security risks are crucial for maintaining data confidentiality in network communications.

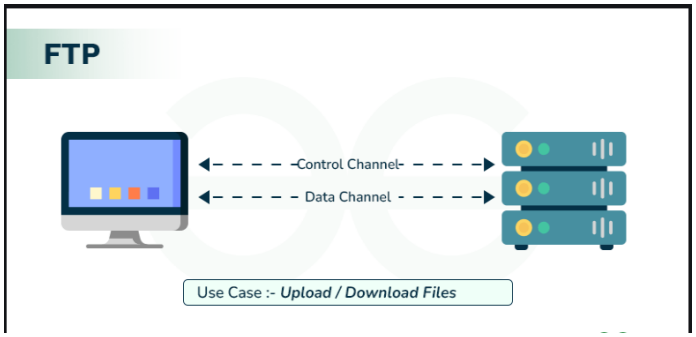


Fig.6: FTP Diagram

Fig.6: Shows the working principle of FTP.

- **ARP**

The Address Resolution Protocol (ARP) is essential for network communication, translating IP addresses to MAC addresses [12]. In packet sniffing, ARP packets can be captured to reveal the physical address

associated with a specific IP on the network. This is crucial for network mapping and security analysis, as it helps in identifying devices and potential vulnerabilities within a network. By monitoring ARP traffic, network administrators can detect anomalies like ARP spoofing, ensuring the network's integrity and preventing man-in-the-middle attacks. Thus, ARP plays a vital role in maintaining secure and efficient network operations.

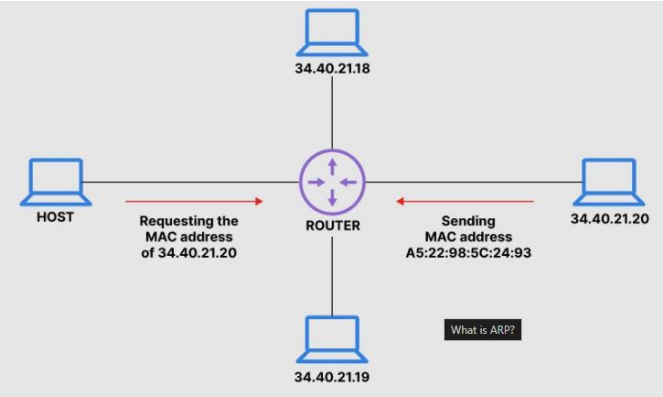


Fig.7: ARP Diagram

Fig.7: Shows the working principle of ARP.

- **TELNET**

Telnet is a network protocol used to provide bi-directional interactive text-oriented communication using a virtual terminal connection [13]. In packet sniffing, unencrypted Telnet traffic can be intercepted, revealing session information including credentials and commands. Telnet operates over TCP/IP networks and uses port 23 by default. Due to its lack of encryption, Telnet is susceptible to eavesdropping, making it insecure for transmitting sensitive data. For secure communication, protocols like SSH are recommended as they provide encryption, which protects against packet sniffing.

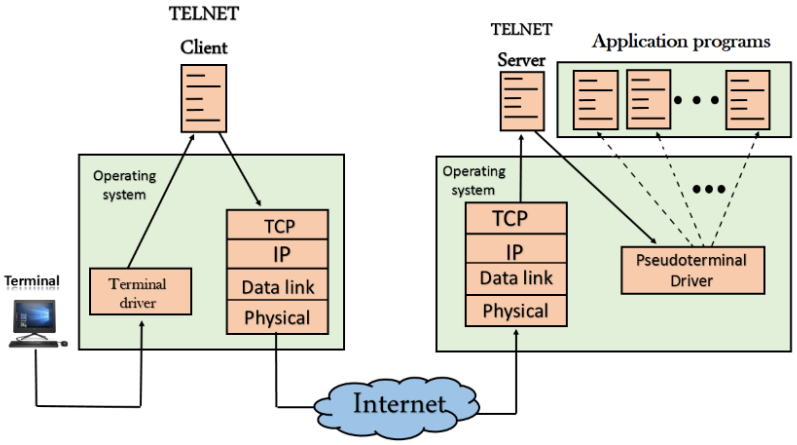


Fig.8: TELNET Diagram

Fig.8: Shows the working principle of TELNE.

- **SNMP**

SNMP, or Simple Network Management Protocol, is used for network management and monitoring [14]. In packet sniffing, SNMP traffic can be captured to analyze network performance and device health. It typically uses UDP ports 161 for SNMP messages and 162 for SNMP traps. Sniffers can intercept these

packets to observe network activity, identify issues, and ensure proper network functionality. SNMP's simplicity makes it a target for sniffing, as it can reveal device information and network structure, emphasizing the need for secure configurations and monitoring practices.

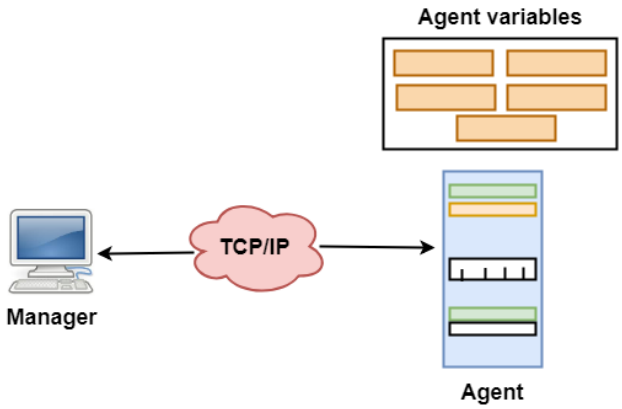


Fig.9: SNMP Diagram

Fig.9: Shows the working principle of SNMP.

- **DHCP**

DHCP, or Dynamic Host Configuration Protocol, automates the assignment of IP addresses and other network configurations to devices on a network. In packet sniffing, DHCP traffic can be monitored to observe the process of devices requesting and receiving network configuration. This includes the DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, and DHCPACK stages of the DHCP process. By capturing these packets, one can analyze how IP addresses are distributed and managed within a network, aiding in network troubleshooting and monitoring for unauthorized DHCP servers or other security threats.

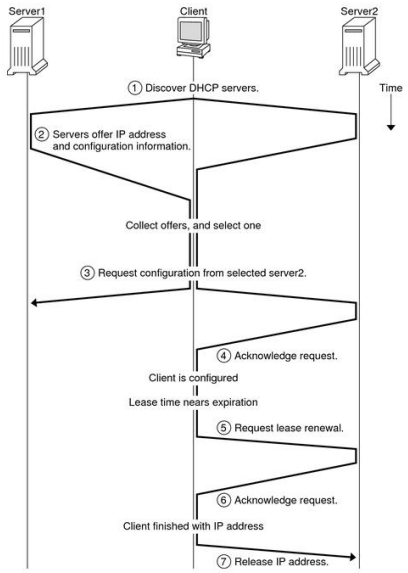


Fig.10: DHCP Diagram

Fig.10: Shows the working principle of DHCP.

3.3 Proposed Model

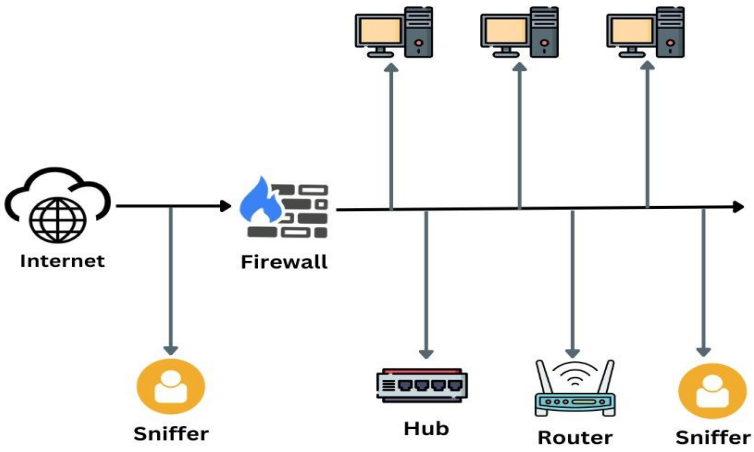


Fig.11: Diagram of Packet Sniffer

Fig.11: Shows the packet sniffer, capturing traffic from devices connected to a router, analyzing data flow for network monitoring.

CHAPTER 4.
PROPOSED METHOD

4.1Configure Backend

Here's is The Algorithm of The Python Code:

Here we define,

Input X = Gathering Packets from the ethernet.

Output Y = Giving output of catching packet's details in a readable format for further future analysis after using separate filters.

Step 1 - Input X.

Step 2 - Initialize Socket: Create a raw socket to capture all incoming packets.

Step 3 - Capture Loop: Continuously read packets from the network interface.

Step 4 - Parse Ethernet Frame: Extract the source and destination MAC addresses, as well as the protocol type.

Step 5 - Check Protocol Type: Determine if the packet is IPv4, ICMP, TCP, UDP, or ARP.

Step 6 - Parse Packet Content: Based on the protocol type, parse the packet to extract relevant information such as IP addresses, port numbers, and payload data.

Step 7 - Analyze Protocols:

- For IPv4 packets, extract version, header length, TTL, protocol, source and destination IP.
- For ICMP packets, extract type, code, and checksum.
- For TCP packets, extract source and destination ports, sequence and acknowledgment numbers, flags, and payload.
- For UDP packets, extract source and destination ports and payload.
- For ARP packets, extract hardware type, protocol type, hardware size, protocol size, opcode, sender MAC and IP, and target MAC and IP.

Step 8 - Decrypt Payload: If the payload is encrypted or encoded, attempt to decrypt or decode it to human-readable format.

Step 9 - Output Y

Step 10 - Write to File: Save the analyzed packed information to a text file for persistence.

Implementation:

Step 1:

Importing Struct and Socket module.

```
import socket
import struct
```

Fig.12: Import Module

Fig.12: Shows importing module.

Step 2:

Creating function "collect_and_show_output" function for showing output.

```
def collect_and_show_output(conn):
    file1 = open('./views/myfile.txt', 'w')
    while True:
        raw_data, addr = conn.recvfrom(65535)
        output = analyze_packet(raw_data)
        # print(output);
        file1.writelines(output)
    file1.close()
```

Fig.13: Creating Function “collect_and_show_output”

Fig.13: Shows “collect_and_show_output” function.

Step 3:

Creating function “analyze_packet” for adding protocols to collect packets.

```
def analyze_packet(raw_data):
    output = ""
    dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)

    if eth_proto == 8:  # IPv4 protocol
        version, header_length, ttl, proto, src_ip, dst_ip, data = ipv4_packet(data)

        if proto == 6:  # TCP protocol
            src_port, dest_port, sequence, acknowledgment, flags, payload = tcp_segment(data)

            if payload:
                output += "\nTCP Packet:\n"
                output += f"Source IP: {src_ip}, Destination IP: {dst_ip}\n"
                output += f"Source Port: {src_port}, Destination Port: {dest_port}\n"
                output += "Decoded Payload:\n"
                try:
                    output += decrypt_payload(proto, payload) + "\n"  # Decrypt payload for analysis
                except UnicodeDecodeError:
                    output += "Unable to decode payload. Raw data:\n"
                    output += str(payload) + "\n"
```

Fig.14: Creating Function “analyze_packet”

Fig.14: Shows “analyze_packet” function.

Step 4:

Creating functions “decrypt_payload” for decrypt payloads in normal text form.

```
def decrypt_payload(proto, payload):
    # Add decryption logic based on the protocol type
    if proto == 6:  # TCP protocol
        # Check if payload contains HTTP or HTTPS data
        if b'HTTP' in payload:
            # Decrypt HTTP payload
            decrypted_payload = decrypt_http(payload)
            return decrypted_payload
        elif b'HTTPS' in payload:
            # Decrypt HTTPS payload
            decrypted_payload = decrypt_https(payload)
            return decrypted_payload
        elif b'FTP' in payload:
            # Decrypt FTP payload
            decrypted_payload = decrypt_ftp(payload)
            return decrypted_payload
        elif b'Telnet' in payload:
            # Decrypt Telnet payload
```

Fig.15: Creating Function “decrypt_payload”

Fig.15: Shows “decrypt_payload” function.

Step 5:

Collecting all output from main function.

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

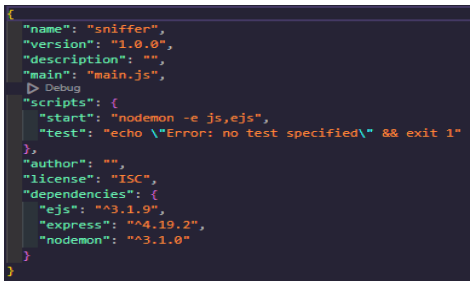
Fig.16: Collecting Data

Fig.16: Shows Collecting Data.

4.2Configure Frontend

Step 1:

Importing packages of Node.js for Web user interface.



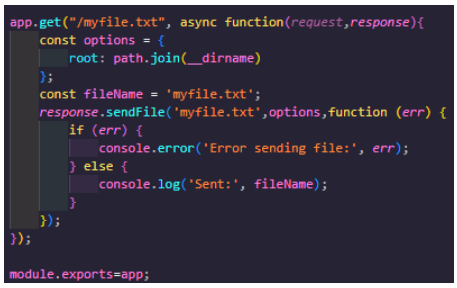
```
{
  "name": "sniffer",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  ▷ Debug
  "scripts": {
    "start": "nodemon -e js,ejs",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ejs": "^3.1.9",
    "express": "^4.19.2",
    "nodemon": "^3.1.0"
  }
}
```

Fig.17: Import Packages

Fig.17: Shows importing packages.

Step 2:

Converting output of the Python's code in a "myfile.text" file for showing outputs in Web.



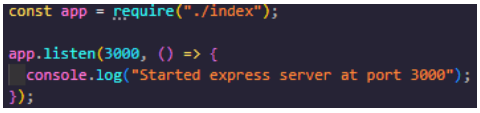
```
app.get("/myfile.txt", async function(request,response){
    const options = {
        root: path.join(__dirname)
    };
    const fileName = 'myfile.txt';
    response.sendFile('myfile.txt',options,function (err) {
        if (err) {
            console.error('Error sending file:', err);
        } else {
            console.log('Sent:', fileName);
        }
    });
});

module.exports=app;
```

Fig.18: Converting output in "myfile.txt"

Fig.18: Shows "myfile.txt" in Node.js.

Step 3:

Run the python code in a local host server for working properly in real-time.



```
const app = require("./index");

app.listen(3000, () => {
  console.log("Started express server at port 3000");
});
```

Fig.19: Connecting Local host Server

Fig.19: Shows local host server at port 3000.

Step 5:

Design web interface using HTML, Node.js for showing the output in a proper form and form of analyzing.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.tailwindcss.com"></script>
    <link type="style" src="../style.css">
    <title>Packet Sniffer</title>
    <script>
        function startscript(){
            var el=document.getElementById('sniff_button');
            if(el.innerHTML=='Start Script'){
                el.innerHTML='Stop Script';
                exec('sudo python3 -u "/home/xakep/Desktop/pingle/packet.py"');
```

Fig.20: Design Web Interface

Fig.20: Shows design Web interface using HTML.

CHAPTER 5.

DATA ANALYSIS AND RESULTS

5.1 Web Interface

The web interface features a “Start Sniff” switch, which, when activated, initiates the real-time collection of network packets. This interactive element allows users to begin monitoring network traffic with ease, providing immediate visual feedback as packets are captured and displayed on the interface. It serves as a user-friendly control for starting and stopping the packet sniffing process, making network analysis accessible even to those with minimal technical expertise. Allowing for comprehensive threat analysis. This information empowers proactive security measures, preventing potential attacks and bolstering the overall resilience of the system, ensuring a robust defense against evolving cybersecurity threats.

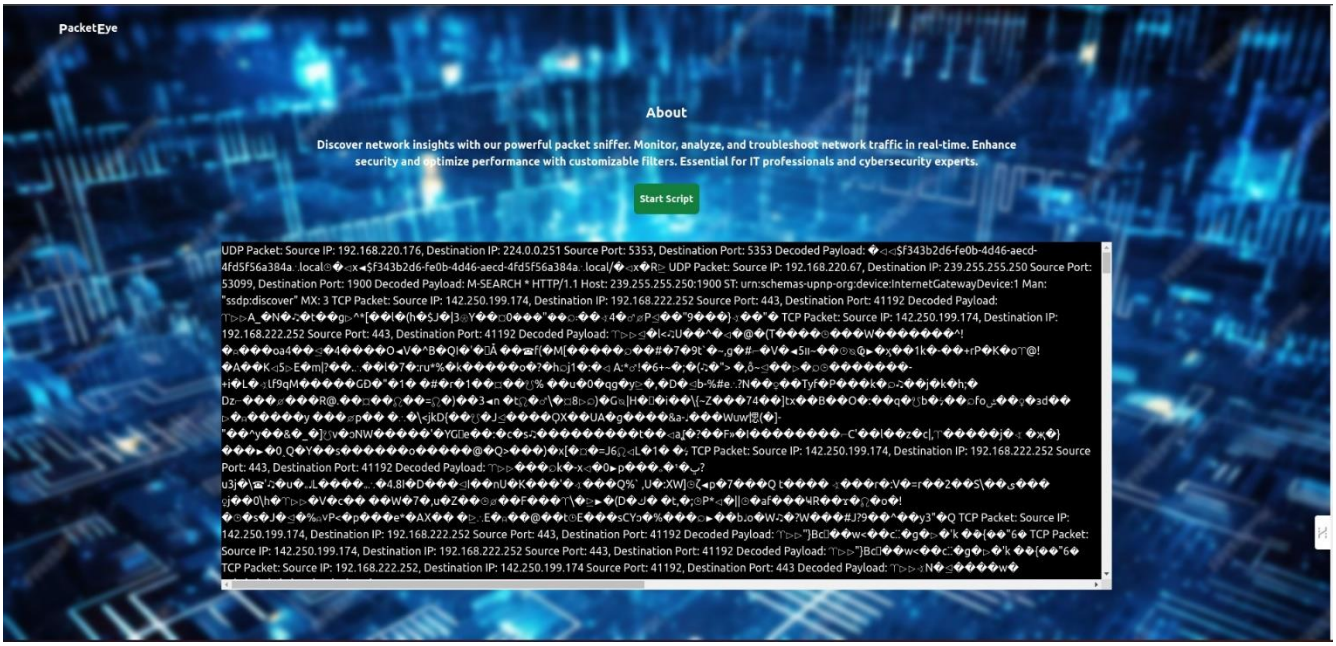


Fig.21: Web Interface

Fig.21: Shows main web page in which left side shows its logo, above about of packet sniffier and collecting packets in the real-time as we start the process.

5.2 Data Analysis

In our data analysis of **"Packet sniffer"** project, we utilized a packet sniffer to capture 222,758 packets on local Ethernet. Our methodology involved analyzing these captured packets to discern their transfer protocols and dissect all traffic contained within the capture file. Additionally, we created visual representations of TCP streams within the capture file. This comprehensive approach enabled us to gain insights into network activity, identify communication patterns, and assess the efficiency and security of data transmission protocols.

Displaying all the traffic:

The visualization of all traffic within a capture file is depicted through a graph, showcasing the packets transmitted per second, measured in bytes or bits. Utilizing bytes and time scaling, this graph provides a clear representation of the analyzed network traffic. On the x-axis, time is delineated in seconds, while the y-axis quantifies the number of packets per tick. This graphical representation aids in understanding the flow and intensity of network activity over time, allowing for insights into peak periods of data transmission, patterns of communication, and potential anomalies. By observing fluctuations in packet counts against the temporal backdrop, analysts can discern trends, identify irregularities, and make informed decisions regarding network optimization and security measures**.**

Packets/1 sec:

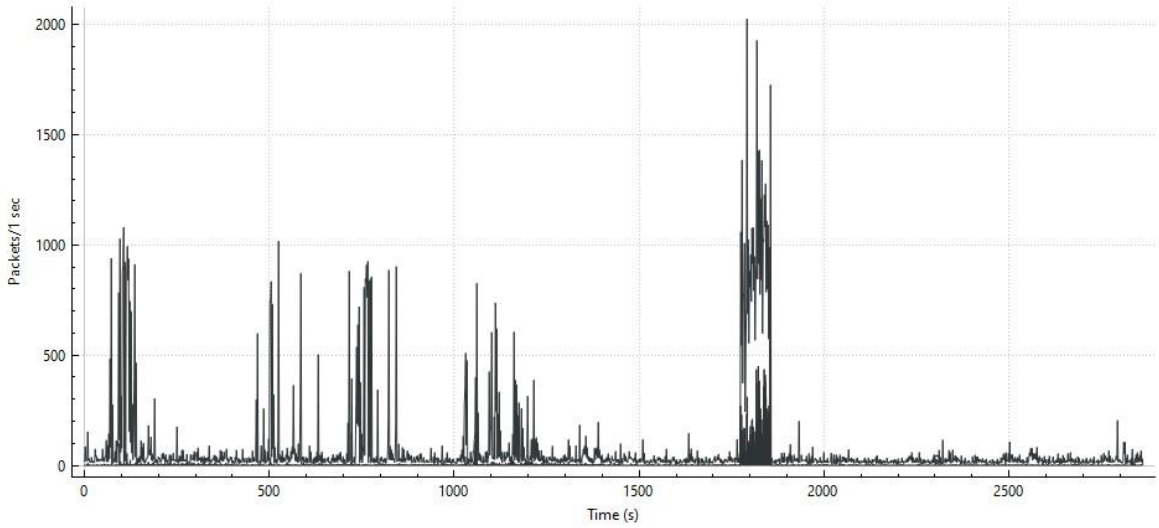


Fig.22: Packets/1 sec

In fig.22: the x-axis represents time in seconds, while the y-axis signifies the number of packets per tick. Specifically, the data on the y-axis indicates the rate of packets per second (Packets/1 second). This visualization method offers a straightforward means of interpreting the flow of network traffic over time. By tracking the fluctuations in packet count per second against the temporal scale, analysts can gain valuable insights into the dynamics of data transmission within the network. Such insights include identifying periods of high or low activity, detecting patterns in communication, and highlighting

potential anomalies or irregularities in traffic behavior.

Packets/5 sec:

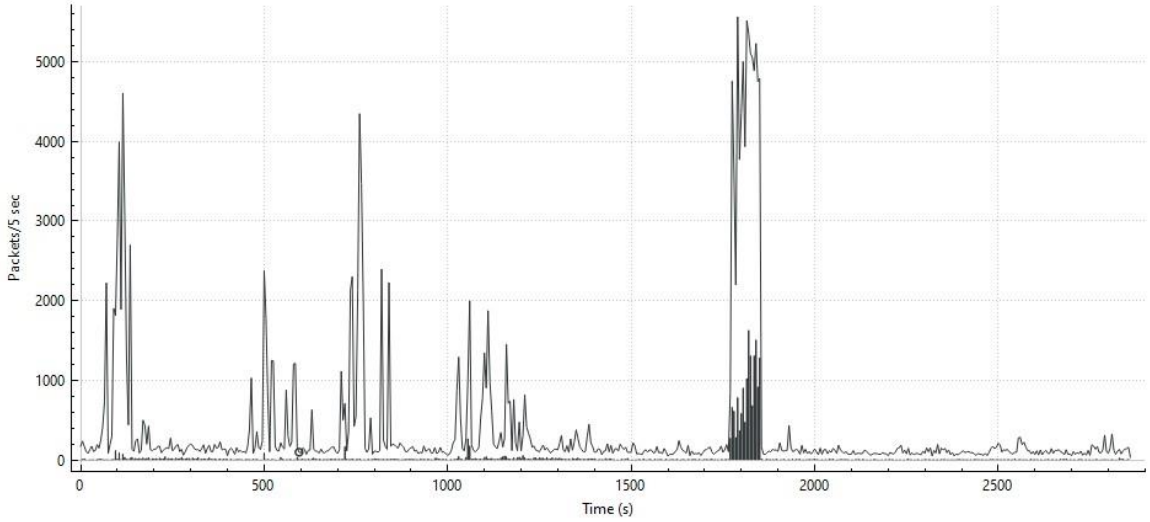


Fig.23: Packets/5 sec

In fig.23: we aim to enhance specificity and ease of analysis by presenting packet data in a more refined manner. To achieve this, we've increased clarity by aggregating packet counts over 5-second intervals. On the x-axis, time is measured in seconds, while the y-axis denotes the number of packets per tick. Specifically, the data on the y-axis represents the rate of packets per 5-second period (Packets/5 seconds). This approach allows for a more granular view of network traffic patterns, facilitating a deeper understanding of data transmission dynamics. By observing fluctuations in packet counts over these larger intervals, analysts can better discern trends, identify peak periods of activity, and detect any abnormalities with greater clarity.

Packets/ 1 min:

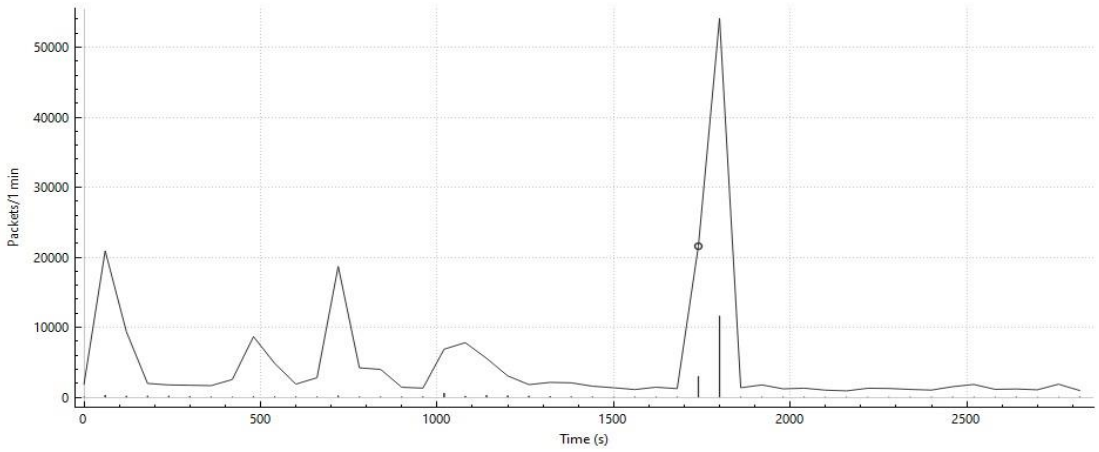


Fig.24: Packets/1 min

In fig.24: we have improved visibility by enhancing clarity through a reduction in granularity. Specifically, we have aggregated packet data over 1-minute intervals to provide a more comprehensive and readable representation. Time is depicted on the x-axis in seconds, while the y-axis indicates the number of packets per tick. Notably, the data on the y-axis now reflects the rate of packets transmitted per 1-minute period (Packets/1 minute). This adjustment allows for a broader overview of network traffic trends and patterns, facilitating easier interpretation and analysis.

By observing packet counts over these larger intervals, analysts can gain valuable insights into overall network activity, identify long-term trends, and pinpoint any significant fluctuations in data transmission. This graph serves as a valuable tool for enhancing the analysis and comprehension of network traffic behaviors, aiding in the optimization and management of network resources and security measures.

Transmission Control Protocol (TCP) Stream:

A TCP stream graph visually depicts the flow of data within a TCP connection recorded in a capture file. It illustrates how data was transmitted over time in a single direction of the TCP connection. Each stream in the graph is assigned a unique identifier, starting from 1 for the first stream, 2 for the second, and so forth. A stream encompasses a set of related TCP packets, usually commencing with the 3-way handshake, followed by data transfer, and concluding with session termination.
In this graph, the x-axis typically represents time, indicating the progression of data transmission events over time intervals. On the other hand, the y-axis represents TCP sequence numbers, which denote the sequential order of data packets within each TCP stream. This graphical representation offers a clear visualization of the chronological sequence of data exchange within TCP connections, aiding in the analysis of network behaviors, performance optimization, and troubleshooting of communication issues.

Time Sequence:

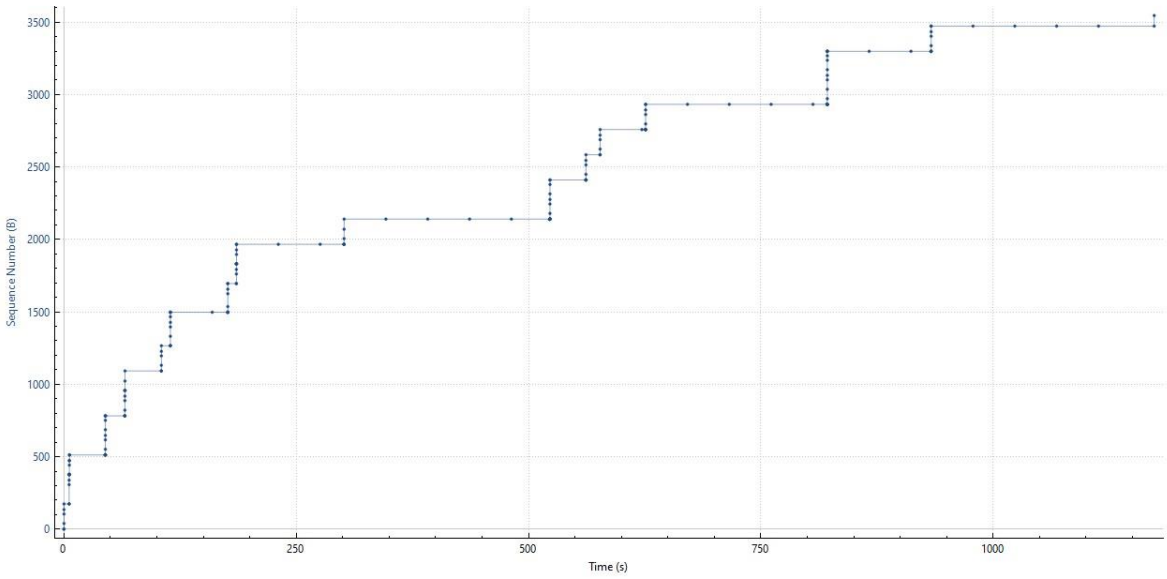


Fig.25: Time Sequence of TCP

In fig.25: the time sequence graph, that depicts TCP sequence numbers over time, with sequence numbers in bytes on the y-axis and time in seconds on the x-axis. The graph showcases the progression of data transmission within TCP connections, offering insights into the order and volume of transmitted data.

Additionally, blue lines positioned above the main line indicate selective acknowledgments, marking the receipt of specific data segments by the recipient.

Throughput:

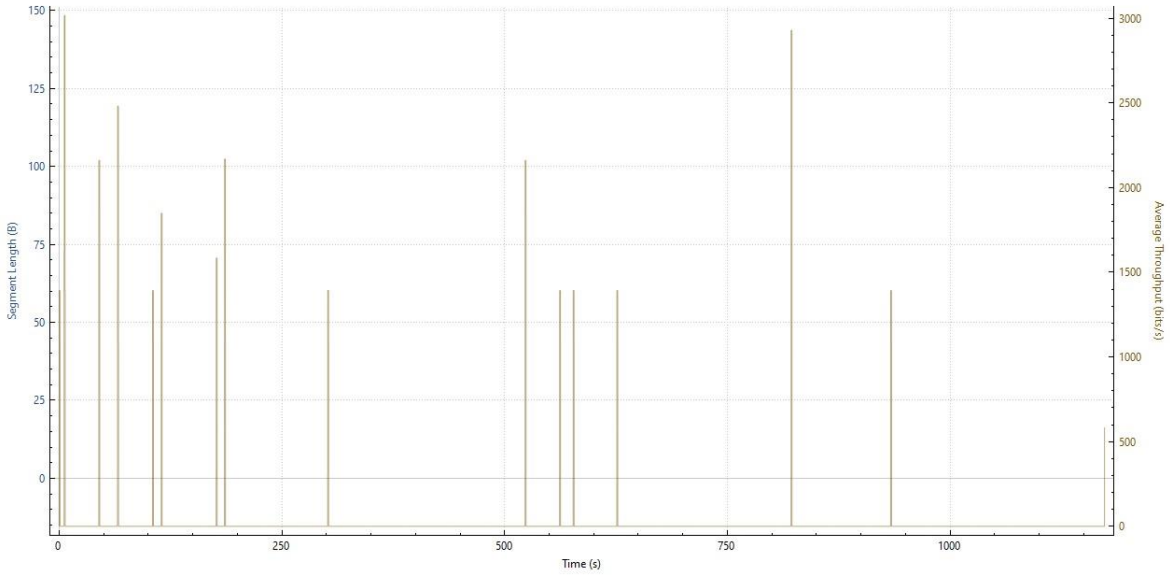


Fig.26: Throughput of TCP

Fig.26: Showcasing average throughput and goodput. The y-axis on the left side represents segment length in bytes, while the x-axis indicates time in seconds. Meanwhile, the y-axis on the right side represents average throughput in bits per second. This graphical representation provides a comprehensive view of data transmission efficiency, with segment length illustrating the volume of data transferred, and average throughput highlighting the rate of successful data delivery. By correlating segment length with throughput metrics, analysts can assess network performance and optimize data transmission processes effectively.

Analyzing Packet Warning Signals:

Analyzing the source of packet transfers to understand underlying issues and assess their potential impact on system functionality. This investigation aims to preemptively address vulnerabilities and implement measures to bolster network resilience and security. Insights gained will inform proactive strategies to mitigate risks and optimize system performance, ensuring a robust and reliable network infrastructure for sustained operation.

125409 DNS: DNS query retransmission

125407 1741.81…	GrandstreamN_d4:68:f4	Broadcast	Ethernet	564 ✓	✓	Ethernet II
125408 1741.81…	GrandstreamN_d4:69:2c	Broadcast	Ethernet	564 ✓	✓	Ethernet II
125409 1741.95…	192.168.222.252	192.168.220.1	DNS	91 ✓	✓	Standard query 0x7c79 A settings-win.data.microsoft.com
125410 1742.12…	192.168.222.235	224.0.0.251	MDNS	408 ✓	✓	Standard query response 0x0000 SRV, cache flush 0 0 5666
125411 1742.12…	fe80::db26:cba9:e481:5e39	ff02::fb	MDNS	428 ✓	✓	Standard query response 0x0000 SRV, cache flush 0 0 5666

Fig.27 125409 DNS

In fig.27: the row highlighted in black, The Domain Name System (DNS) protocol queries are displayed, indicating warning levels within the total packet count.

We conducted a detailed analysis, revealing specific packet details within the selective data.

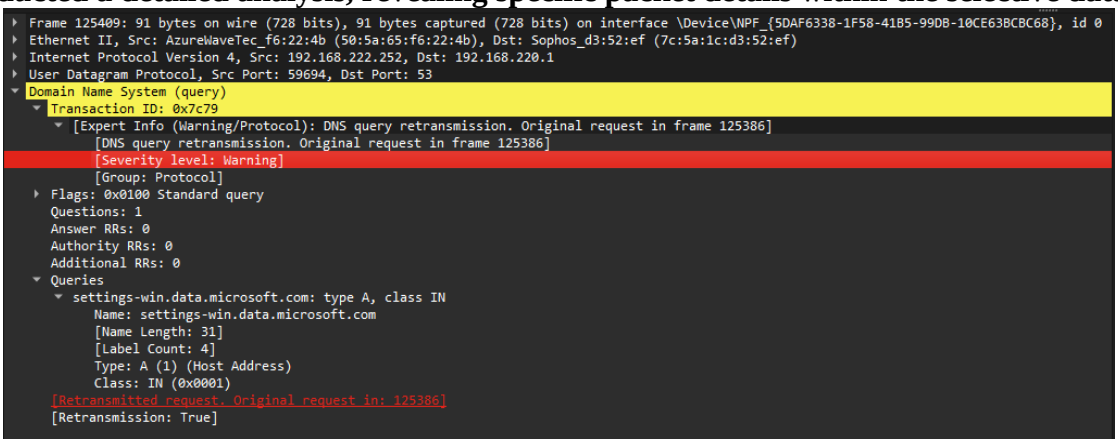


Fig.28: 125409 Packet Details

In fig.28: we readily identify selective packet details including source and destination IP and port, transaction ID, warning levels, and flags. Such visibility exposes network vulnerabilities, potentially exploited by malicious actors. This underscores the importance of addressing weaknesses to fortify network security and thwart potential threats.

92705 TCP: TCP Zero Window segment

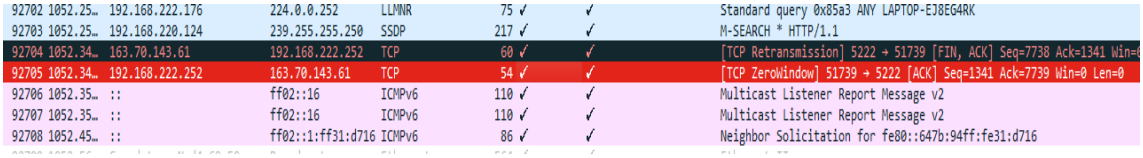


Fig.29: 92705 TCP

In fig.29: the row highlighted in red, Transmission Control Protocol (TCP) Zero Window are displayed, indicating warning levels within the total packet count.

We conducted a detailed analysis, revealing specific packet details within the selective data.

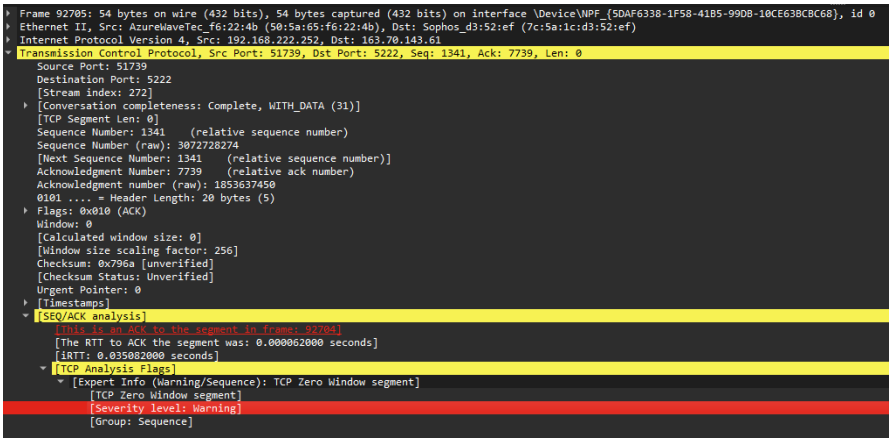


Fig.30: 92705 Packet Details

In fig.30: we can readily observe key details of selective packets, including source and destination IP and port, transaction ID, warning levels, acknowledgement number, header length, TCP analysis and its flags. This vulnerability in the network could be exploited by malicious individuals, highlighting the need for robust security measures.

69481 HTTP: HTTP/1.1 204 No Content

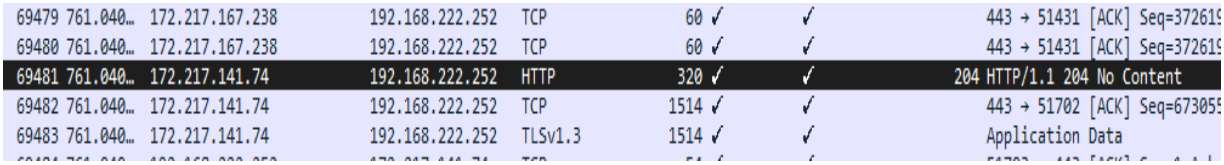


Fig.31: 69481 HTTP

In fig.31: the row highlighted in black, Unencrypted the Hypertext Transfer Protocol (HTTP) detected over encrypted port, could indicate a dangerous misconfiguration.

We conducted a detailed analysis, revealing specific packet details within the selective data.

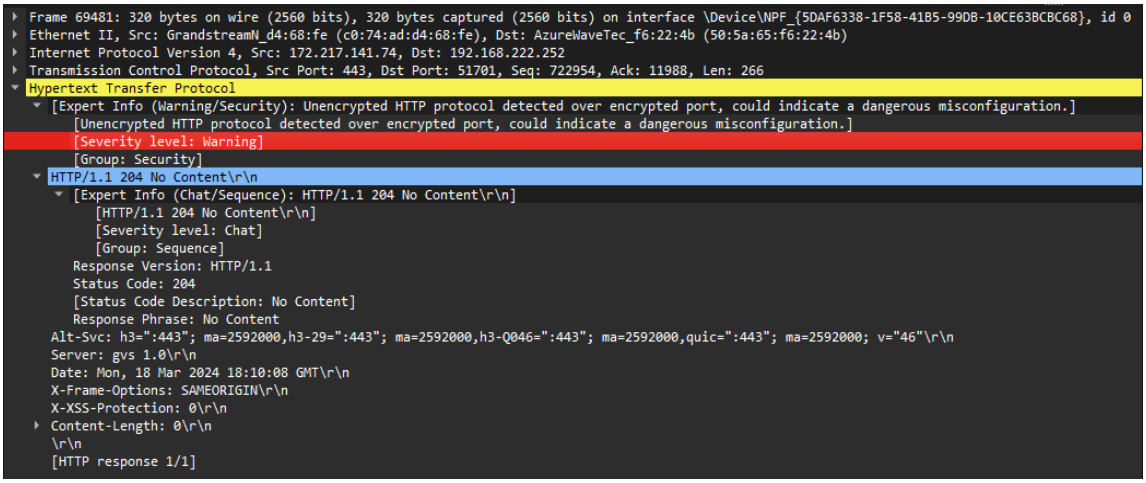


Fig.32: 69481 Packet Details

In fig.32: we examine selective packet details including bytes on wire and capture, interface specifics, source and destination MAC addresses, port numbers, sequence and acknowledgement, length, packet response, and X-XSS-Protection. The unencrypted HTTP protocol poses vulnerability, enabling attackers to exploit and cause harm.

9330 TCP: Out-Of-Order segment

Fig.33: 9330 TCP

In fig.33: the row highlighted in red, suspected Out-Of-Order segment detected over Transmission Control Protocol (TCP), indicating warning levels within the total packet count.

We conducted a detailed analysis, revealing specific packet details within the selective data.

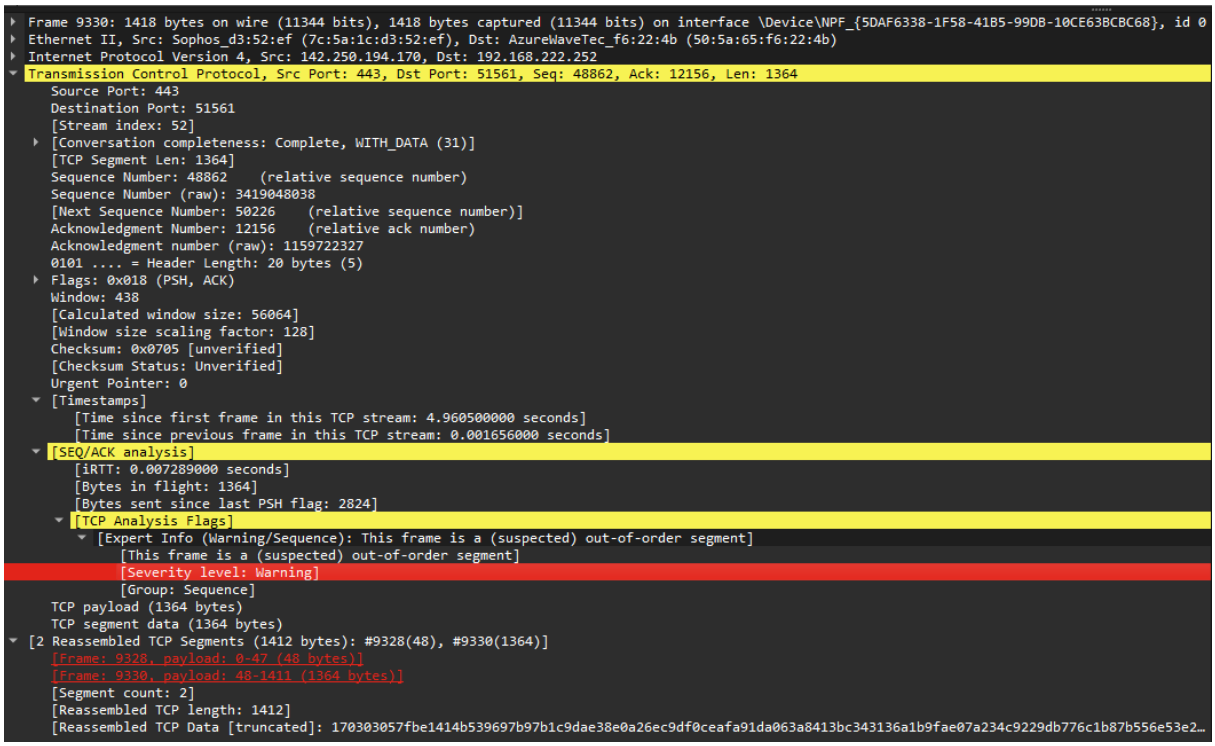


Fig.34:9330 Packet Details

In fig.34: we observe selective packet details including bytes on wire and capture, interface specifics, source and destination IP and port, sequence and acknowledgement, TCP segment length, flags, checksum, and timestamps. Suspected out-of-order segments and truncated TCP data payloads are identified, indicating potential anomalies. This vulnerability in the network could be exploited by malicious individuals, highlighting the need for robust security measures.

1650 TLS: Ignored Unknown Record

1648	56.949526	192.168.222.252	172.217.167.238	TLSv1.2	749 ✓	✓	Application Data
1649	56.949609	192.168.222.252	172.217.167.238	TLSv1.2	311 ✓	✓	Application Data
1650	56.949921	192.168.222.252	172.217.136.169	TLSv1.2	1514 ✓	✓	Ignored Unknown Record
1651	56.949921	192.168.222.252	172.217.136.169	TLSv1.2	957 ✓	✓	Ignored Unknown Record
1652	56.949976	192.168.222.252	172.217.136.169	TLSv1.2	78 ✓	✓	Application Data

Fig.35: 1650 TLS

In fig.35: the row highlighted in black, Ignored Unknown Record detected over Transport Layer Security (TLS) protocol, could indicate a failed cryptographic communication between web applications and servers.

We conducted a detailed analysis, revealing specific packet details within the selective data.

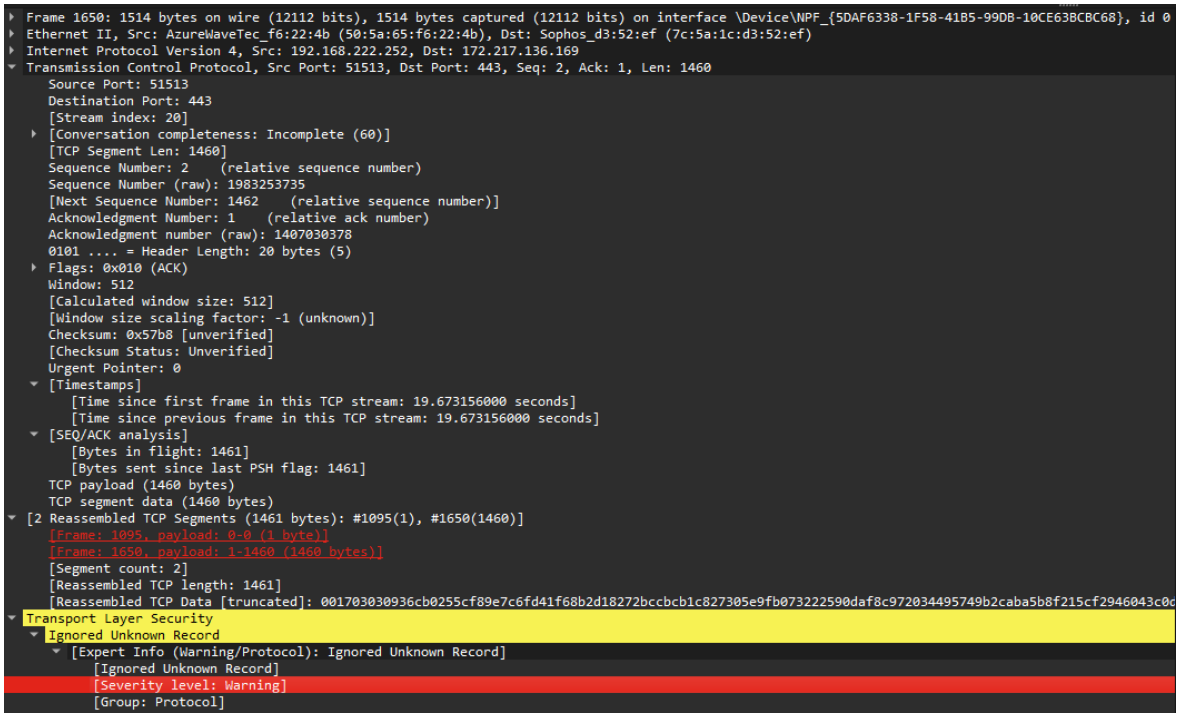


Fig.36: 1650 Packet Details

In fig.36: we examine selective packet details including bytes on wire and capture, interface specifics, source and destination MAC and port, sequence and acknowledgement, flags, checksum, timestamps, TCP payloads, and locations of reassembled TCP data truncation. These insights reveal communication failures and packet losses at specific points within the network.

1133 DHCPv6: WARNING: TLDs are rarely resolvable

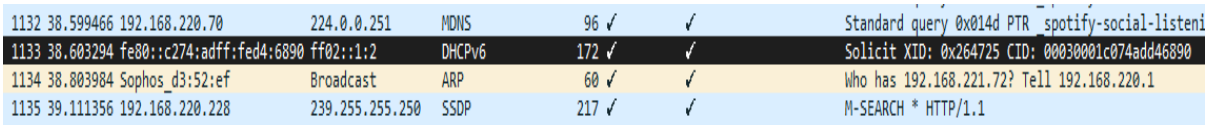


Fig.37: 1133 DHCPv6

In fig.37: the row highlighted in black, TLDs are rarely resolvable detected over Dynamic Host Configuration Protocol version 6 (DHCPv6), indicating warning levels within the total packet count.

We conducted a detailed analysis, revealing specific packet details within the selective data.

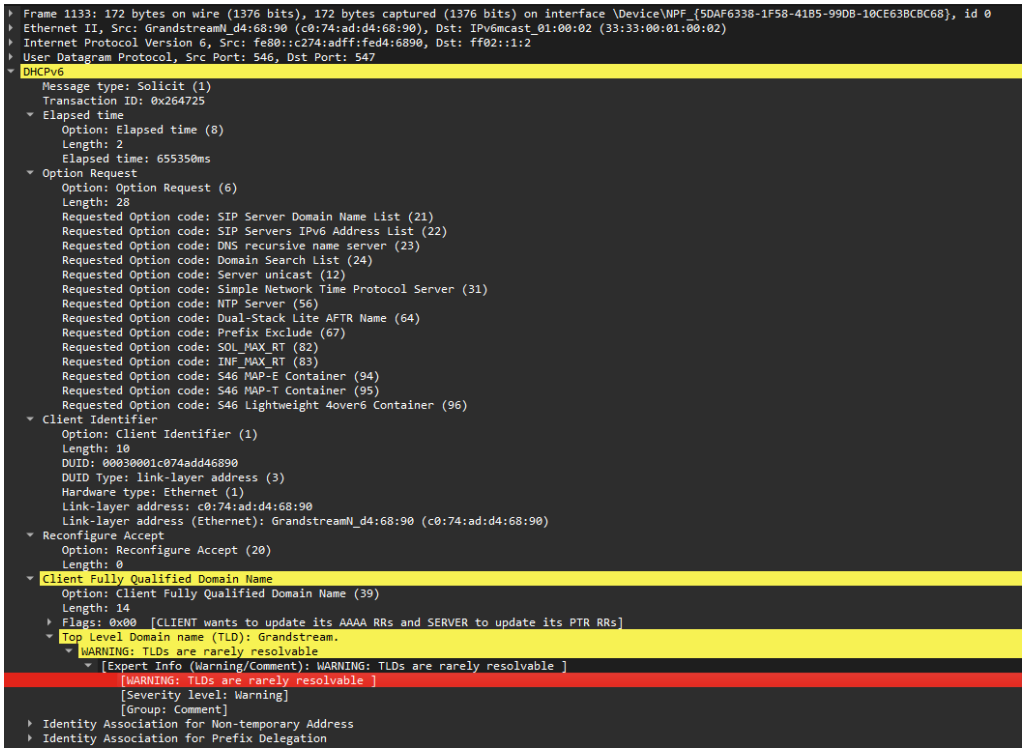


Fig.38: 1133 Packet Details

In fig.38: we analyze selective packet details including bytes on wire and capture, interface specifics, source and destination IPv6 and port, DHCPv6 data, elapsed time, client identifier, flags, and option requests. This reveals challenges with resolving top-level domains (TLDs) and domain name issues. This vulnerability in the network could be exploited by malicious individuals, highlighting the need for robust security measures.

151 MDNS: DNS response retransmission

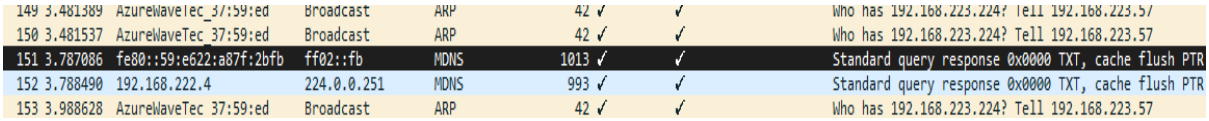


Fig.39: 151 MDNS

In fig.39: the row highlighted in black, DNS response retransmission detected over The Multicast Domain Name System (MDNS) protocol, indicating warning levels within the total packet count.

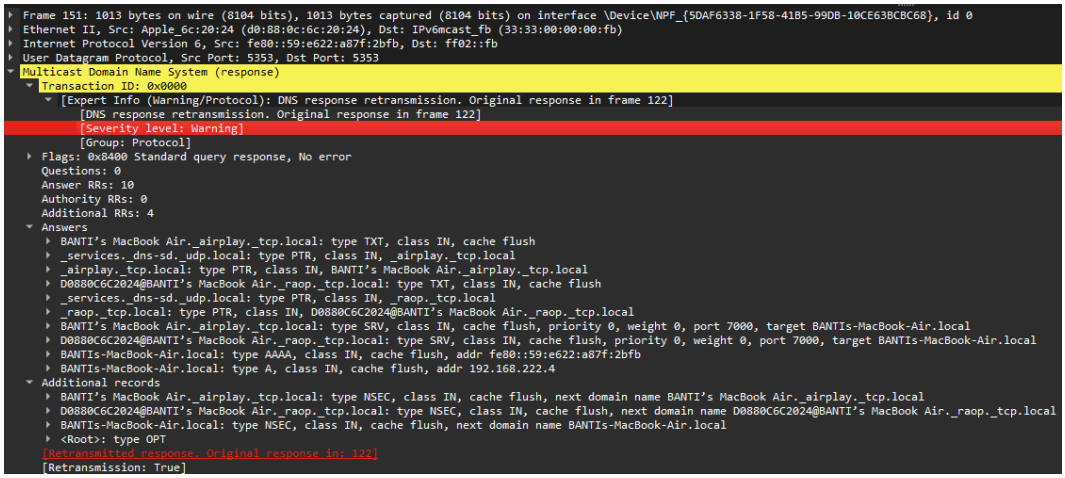


Fig.40: 151 Packet Details

In fig.40: we observe selective packet details, including bytes on wire and capture, interface specifics, source and destination IPv6 and port, flags, and additional data. This reveals DNS response retransmissions due to host non-response, potentially resulting in packet loss and delayed response that receipt in frame 122.

CHAPTER 6.

CONCLUSION & FUTURE WORK

6.1 Conclusion

This chapter summarizes the methodology of the experiments used in the report. Leveraging Python for backend packet sniffing and Node.js for frontend development presents a powerful combination for network monitoring applications. Python's extensive libraries like Struct and Socket offer robust capabilities for packet capture, analysis, and manipulation, making it ideal for implementing packet sniffing functionalities efficiently. With Python, developers can create custom scripts to extract specific data from network packets and integrate them into backend systems seamlessly. On the frontend, Node.js provides a lightweight and scalable environment for building responsive user interfaces. Its event-driven architecture and asynchronous capabilities enable real-time updates and interactive features, enhancing the user experience in network monitoring applications. By combining Python's backend prowess with Node.js's frontend agility, developers can deliver comprehensive and responsive network monitoring solutions tailored to diverse requirements.

6.2 Future Work

In future iterations of this packet sniffing project, several enhancements and expansions could be considered. One avenue for improvement is to enhance the packet analysis capabilities by incorporating machine learning techniques. This could involve developing models to automatically classify and detect anomalies in network traffic, improving the project's ability to identify suspicious activity. Additionally, implementing more advanced filtering and parsing mechanisms could make the packet sniffing process more efficient and targeted, reducing processing overhead. Furthermore, integrating support for additional protocols and network layers would broaden the project's scope and usefulness across diverse network environments. Finally, enhancing the project's user interface and visualization features could improve usability and provide more intuitive insights into network behavior. These future developments would contribute to making the packet sniffing project more sophisticated, effective, and user-friendly.

APPENDIX

1. **Technical Specifications:** Provide a detailed breakdown of the hardware and software requirements needed to deploy and operate the packet sniffer.

2. **Installation Guide:** Step-by-step instructions on how to install and configure the packet sniffer software on different operating systems (e.g., Windows, Linux, macOS).

3. **User Manual:** A comprehensive guide for users on how to use the packet sniffer, including navigating the user interface, initiating packet captures, filtering traffic, and analyzing captured data.

4. **Packet Capture Examples:** Present examples of real-world packet captures obtained using the packet sniffer, showcasing various network protocols and traffic patterns.

5. **Data Analysis Techniques:** Explain the methodologies and algorithms employed by the packet sniffer to analyze captured network traffic, such as packet parsing, protocol detection, and traffic pattern recognition.

6. **Security Considerations:** Discuss potential security risks associated with using a packet sniffer, as well as best practices for securing the packet sniffer deployment and protecting sensitive information.

7. **Performance Optimization:** Tips and strategies for optimizing the performance of the packet sniffer, including resource utilization, memory management, and packet processing efficiency.

8. **Troubleshooting Guide:** Common issues and troubleshooting steps for diagnosing and resolving problems encountered during packet capture and analysis.

9. **References and Resources:** A curated list of references, documentation, and online resources for further reading on packet sniffing techniques, network analysis tools, and related topics.

10. **Appendix A:** Glossary: Definitions of key terms, acronyms, and technical jargon used throughout the documentation to aid readers' understanding.

REFERENCES

[1] *Ryan Spangler, "Packet Sniffing on Layer 2 Switched Local Area Networks", Packetwatch Research, December 2003.*

[2] Asrodia, Pallavi, and Hemlata Patel. "Network traffic analysis using a packet sniffer." International journal of engineering research and applications 2.3 (2012): 854-856. (n.d.-a).

[3] Nayak, Mr Parikshith, S. H. Brahmananda, and Mrs Sahana DS. "An Approach to Sniff Sensitive Information by Packet Sniffing." (n.d.-b).

[4] Ogbu, Henry N., and Moses Adah Agana. "Intranet Security using a LAN Packet Sniffer to Monitor Traffic." arXiv preprint arXiv:1910.10827 (2019). (n.d.-c).

[5] Wireshark:https://www.wireshark.org/docs/wsug_html_chunked/ChWorkDisplayFilterSection.html. (n.d.-d).

[6] D. Comer, Internetworking with TCP/IP, Vol. 1: Principles, Protocols, and Architecture, 5thedn., Prentice Hall, Upper Saddle River, NJ, 2005. (n.d.-b).

[7] Postel, J. (June 1978). "IEN 41, Internetwork Protocol Specification Version4" (PDF). The Internet Society. Archived from the original (PDF) on 16 May2019. Retrieved 16 December 2020. (n.d.-e).

[8] Postel, J. (ed.), "Internet Protocol - DARPA Internet ProgramProtocol Specification," RFC 791, USC/Information SciencesInstitute, September 1981. (n.d.-e).

[9] V. G. Cerf and R. E. Icahn, '"A protocol for packet network intercom-munication,"' ACM SIGCOMM Comput. Commun. Rev., vol. 35, no. 2,pp. 71–82, 2005. (n.d.-g).

[10] Fielding, R. et. al, Hypertext Transfer Protocol –HTTP/1.1, Request for Comments: 2616, InternetEngineering Task Force. (n.d.-c).

[11] E-Mail and Ftp (File Transfer Protocol) by L. Joyce Arnston, Kathy Berkemeyer, Ken Halliwell, Thomas Neuburger. (n.d.-c).

[12] Plummer, D.C. (1982) An Ethernet Address Resolution Protocol. RFC 826. (n.d.-g).

[13] Howard, V.J. et al., "An Interactive TerminalProtocl", EPSS Liazson Group, Study Group 2,HLP/CP(75)2, AERE Harwell, U.K. (n.d.-e).

[14] Peter Drake, Using SNMP to manage networks, IEEE Network Journal. (n.d.-h).