T.C

İSTANBUL OKAN UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Network Intrusion Detection System

Graduation Project

PROGRAM:

Computer Engineering

Prepared By:
Yazid Omran 200212315

Advisor:
Prof. Dr. Pınar YILDIRIM

İSTANBUL

2024

“All the information in this project was obtained and presented in accordance with academic rules and ethical principles; I also declare that I make all references that do not originate from this study, as required by these rules and principles. "

Yazid Omran

SUMMARY

GRADUATION PROJECT

NETWORK INTRUSION DETECTION SYSTEM(IN-SIGHT)

In-Sight is a network monitoring and analysis tool developed using Python, utilizing powerful libraries such as Scapy for packet capturing and processing, and PyShark for packet analysis. These technologies and libraries were chosen for their versatility, reliability, and ease of use, enabling efficient development and robust functionality. With this combination of tools, In-Sight provides network administrators and security analysts with a comprehensive solution for monitoring, analyzing, and securing network traffic, ultimately enhancing the overall security posture of their network infrastructure.

Keywords: Network monitoring, Analysis tool, Python, Scapy, PyShark

ACKNOWLEDGMENT

Our respected professor, who generously offered their interest and assistance throughout the planning, research, execution, and development of this thesis, drew upon their extensive knowledge and expertise to guide and inform our work according to scientific principles. We are immensely grateful to Associate Professor Pınar Yıldırım for her invaluable support.

Contents

List of Figures

1.INTRODUCTION

1.1 Project Definition and Goals

Our project, "In-Sight," aims to create a user-friendly tool for monitoring and understanding network traffic. We want to help network administrators and security experts better protect their networks by giving them clear insights into what's happening on their systems.

1.2 Statement of Problem

In today's world, cyber threats are getting more sophisticated, making it harder to keep networks safe. Many existing tools for monitoring networks aren't always up to the task of spotting these threats in time. We're building "In-Sight" to bridge this gap and provide a smarter solution for keeping networks secure.

1.3 Mission & Vision

Our mission is simple: to make network security easier and more effective. We want "In-Sight" to be the go-to tool for anyone who needs to keep an eye on their network's health. By giving users real-time updates on network activity and potential threats, we're helping them stay one step ahead of cyber attackers.

1.4 Similar Projects

While there are other tools out there for monitoring networks, we believe "In-Sight" stands out for its ease of use and powerful features. We've designed it to be intuitive and customizable, so it can adapt to the unique needs of each user.

2. Literary Research

2.1 Python Programming Language

Figure 1:Python Logo

Python's popularity in the realm of cybersecurity and networking stems from its versatility and extensive ecosystem of libraries. Beyond networking, Python finds applications in web development, data analysis, artificial intelligence, and more. Its readability and simplicity make it an excellent choice for both beginners and experienced developers alike.

2.2 Scapy

Scapy isn't just limited to packet manipulation – it's a powerful tool for network exploration, attacks, and security auditing. Its interactive shell allows for quick prototyping and testing of network protocols. Additionally, Scapy's ability to handle a wide range of network layers and protocols makes it invaluable for understanding complex network architectures.

Figure 2:Scapy Logo

2.3 PyShark

Figure 3:PyShark Logo

PyShark's integration with Wireshark gives it access to a vast array of protocol dissectors, allowing for comprehensive packet analysis. Its Pythonic interface makes it easy to automate packet capture and analysis tasks, saving time and effort for network administrators and security professionals. PyShark's compatibility with other Python libraries enhances its utility for building custom network monitoring solutions.

2.4 PySimpleGUI

Figure 4:PySimpleGUI Logo

PySimpleGUI's cross-platform compatibility ensures that our tool's interface remains consistent across different operating systems. Its simplicity belies its power – PySimpleGUI enables rapid development of GUI applications without sacrificing customization options.

With PySimpleGUI, developers can focus on creating intuitive interfaces without getting bogged down in complex code.

2.5 Packet Capturing and Analysis

Figure 5:In-Sight Logo

Capturing and analyzing network packets is like peeking into the inner workings of a network. It reveals valuable insights into network traffic and potential security issues. Using Python libraries like Scapy and PyShark, we can dive deep into packet data and understand network behavior better.

3.Methodology

In developing our network security application, we employed a meticulous and multifaceted approach. This section outlines the steps we took, the tools we used, and the processes we implemented to ensure our application is robust, user-friendly, and effective in detecting network threats.

3.1 Data Capture and Preprocessing

Our first step was to capture network traffic efficiently. We chose the Scapy library for its versatility and powerful packet manipulation capabilities. The `get_wifi_interface` function was essential in identifying the appropriate Wi-Fi interface for capturing packets. Once the packets were captured, they were processed by the `pkt_process` function, which serves as the backbone of our data preprocessing.

The `pkt_process` function meticulously decodes the packet header information, extracting crucial details such as source and destination IP addresses, protocol type (TCP, UDP, etc.), port numbers, and timestamps. This information forms the basis for all subsequent analyses. To make the data more readable, we utilized the `proto_name_by_num` function, which translates numerical protocol codes into human-readable formats like "TCP" or "UDP."

For HTTP packets, we went a step further. The `read_http` function performs a deep dive into the packet structure, identifying individual HTTP streams and extracting header information using the `get_http_headers` function. In certain cases, we employed the `extract_object` function to delve deeper into specific content types (e.g., JSON, XML), extracting object data and type based on the content headers.

3.2 Rule Definition and Management

Defining and managing security rules was a critical component of our project. These rules are stored in a text file, allowing users to customize the application's security posture. The `readrules` and `process_rules` functions work together to parse and interpret these rules.

Each rule specifies a protocol (e.g., TCP, UDP), source and destination IP addresses, and ports involved in potentially malicious communication patterns. Some rules also include keywords or patterns that should be present within the packet payload to trigger an alert. This approach allows users to tailor the application's detection capabilities to their specific security needs, providing flexibility and control.

3.3 Packet Analysis and Threat Detection

At the heart of our application lies the `pkt_process` function, responsible for the comprehensive analysis of each captured packet. This function matches packets against the defined security rules using the `check_rules_warning` function. It examines the protocol, source and destination IP addresses, ports, and payload content.

When a packet matches a rule, it is flagged as suspicious, prompting further investigation. The function then extracts essential details such as source and destination IP addresses, protocol type, timestamps, and packet size. In specific scenarios, we decode the payload of suspicious packets to gain deeper insights into the potential threat. This decoded payload is stored for further forensic analysis.

3.4 User Interface Design

Our user interface, created using PySimpleGUI, is designed to be both intuitive and powerful. It enables users to interact with the application efficiently and visualize the captured data clearly.

- **Packet Management:** Users can save captured packets for later analysis, which is crucial for revisiting suspicious activity or conducting in-depth forensic investigations.
- **Rule Management:** The interface allows users to refresh the list of defined security rules dynamically, ensuring the application remains up-to-date with the latest security requirements.
- **Stream Exploration:** Dedicated functions like `show_tcp_stream_openwin` and `show_http2_stream_openwin` enable users to load and view details of identified

TCP/HTTP2 streams, providing deeper insights into ongoing communication channels and potential vulnerabilities.

- **Data Visualization:** The interface organizes captured data into separate lists, including all captured packets, suspicious packets, and decoded payloads. This structured presentation helps users quickly identify and investigate suspicious activity.

3.5 Testing and Validation

Ensuring our application functions correctly and reliably was paramount. We adopted a comprehensive testing strategy:

- **Unit Testing:** Individual functions such as pkt_process and check_rules_warning were tested to verify their functionality and identify potential bugs in isolation.
- **Integration Testing:** We tested the overall application to ensure seamless integration between its various components—data capture, rule processing, analysis engine, and user interface. This involved capturing diverse network traffic scenarios and verifying the application's ability to detect threats.
- **Penetration Testing:** In advanced testing scenarios, we employed penetration testing tools to simulate real-world attack vectors. This allowed us to evaluate the application's ability to detect and respond to actual hacking attempts.

4.Results and Analysis

4.1 Rule Parsing and Suspicious Packet Detection

The system effectively parses rules from the rules file and categorizes them correctly, enabling precise matching against incoming network packets. The parsed components (protocol, IP addresses, ports, and messages) are stored in structured lists, which facilitate efficient rule matching. During testing, the system consistently identified packets that matched the predefined rules, flagging them as suspicious and logging relevant details.

4.2 Network Traffic Monitoring and Packet Capture

The system successfully identified the Wi-Fi interface on Windows platforms and initiated real-time packet capturing. The GUI displayed a comprehensive list of captured packets, highlighting those flagged as suspicious based on the parsed rules. Users were promptly notified of suspicious activity, with detailed packet summaries and decoded payloads presented for further inspection.

4.3 User Interface and Stream Analysis

The GUI, developed with PySimpleGUI, provided an intuitive platform for monitoring network traffic. Key observations include:

- **Packet Display:** Both general and suspicious packet lists were effectively maintained, offering clear visual cues for user attention.
- **Payload and Stream Inspection:** Decoded payloads of suspicious packets were displayed accurately, aiding in detailed threat analysis. The system's capability to display TCP and HTTP2 streams was validated, demonstrating its utility in identifying complex, multi-packet threats.
- **HTTP Object Extraction:** The extraction and display of HTTP objects from captured traffic proved useful in pinpointing specific threats embedded within HTTP streams.

4.4 Performance Metrics

During real-time packet capture and analysis, the system maintained high performance and responsiveness. The use of multithreading ensured that the GUI remained interactive while processing incoming network traffic. This efficiency was evident across various network scenarios, from low-traffic to high-traffic environments, where the system handled packet processing without significant delays or performance degradation.

4.5 Error Handling and Robustness

The system demonstrated robust error handling capabilities, logging errors for debugging purposes while presenting critical issues to the user via the GUI. This ensured continuous operation and reliability, even when encountering unexpected or malformed data.

4.6 Summary of Key Findings

1. **Accuracy and Reliability:** The system's rule matching was precise, consistently identifying suspicious packets based on predefined criteria.
2. **Real-Time Analysis:** The real-time processing of packets and immediate feedback on network activity enhanced its effectiveness for threat detection.
3. **User Interface Effectiveness:** The user-friendly GUI provided clear and actionable insights into network traffic, with comprehensive displays of packets and streams.
4. **In-Depth Analysis Capabilities:** The system's ability to decode and analyze TCP and HTTP2 streams significantly enriched its utility for detailed network threat analysis.

Overall, the system proved to be an effective tool for capturing, analyzing, and displaying network traffic, offering valuable functionality for network monitoring and threat detection.

4. Risk Management

3.1 Technical Challenges

Risk: Things might get tricky when we're building the system, especially with all the technical stuff involved in capturing and analyzing network data using Python.

Mitigation: We'll be testing everything thoroughly as we go along, making sure we fix any problems early on. Plus, we'll be tapping into online resources and docs to help us out.

3.2 Integration Issues

Risk: Bringing together different parts of the system, like capturing data, analyzing it, and the user interface, could cause some headaches. There might be compatibility issues or clashes in how things work.

Mitigation: We're breaking things down into smaller parts, testing each one to make sure they play nice together. We'll also be keeping a close eye on how everything fits together and sorting out any issues as soon as they crop up.

3.3 Security Concerns

Risk: Because we're dealing with network data, there's a chance of security problems cropping up, which could mean sensitive info gets exposed or we're open to cyber-attacks.

Mitigation: We're putting in strong security measures like encrypting data, controlling who can access what, and making sure our code is written securely. We'll also be regularly checking for any weak spots and fixing them up fast.

3.4 Time Constraints

Risk: Making sure we hit all our deadlines could be tough, especially with everything we've got to build and test.

Mitigation: We're planning things out carefully, setting realistic deadlines, and keeping a close eye on how we're doing. If anything does come up that slows us down, we've got backup plans in place.

3.5 Resource Limitations

Risk: We might not have everything we need to get the job done, whether it's people, hardware, or software.

Mitigation: We're making sure we're using our resources wisely, and looking at other options if we need to. If things do get tight, we'll figure out a way to make it work.

3.6 Risk Table

Category	Risk	Probability	Impact
Technical Challenges	Difficulties in implementing network data capture and analysis	Medium	High
Integration Issues	Compatibility problems between system components	Low	Medium
Security Concerns	Data breaches or cyber attacks	Medium	High
Time Constraints	Delays in project milestones	High	High
Resource Limitations	Insufficient resources for project completion	Low	Medium

5.TIMELINE TABLES

4.1 Planning table

Task	Start Date	End Date	Duration
Project Kickoff	**21.12.2023**	**25.12.2023**	**5 days**
Requirements gathering	**26.12.2023**	**15.01.2024**	**21 days**
Define project scope	**16.01.2024**	**20.01.2024**	**5 days**

Figure 6:Planning table

4.2 Analysis table

Task	Start Date	End Date	Duration
Research of similar projects	**21.12.2023**	**29.12.2023**	**9 days**
Potential use analysis	**30.12.2023**	**15.01.2024**	**17 days**
Collecting information	**16.01.2024**	**20.01.2024**	**5 days**
Process model determination	**21.01.2024**	**04.02.2024**	**15 days**
Risk management	**05.02.2024**	**19.02.2024**	**15 days**
Requirement analysis report	**20.02.2024**	**05.03.2024**	**11 days**

Figure 7:Analysis table

4.3 Design Table

Task	Start Date	End Date	Duration
Design Planning	06.03.2024	10.03.2024	5 days
Use case diagram	11.03.2024	13.03.2024	3 days
Data flow diagram	14.03.2024	16.03.2024	3 days
Class diagram	17.03.2024	19.03.2024	3 days
Sequence diagram	20.03.2024	22.03.2024	3 days
State diagram	23.03.2024	24.03.2024	2 days
User interface Design	25.03.2024	27.03.2024	3 days
Detailed design report	28.03.2024	30.03.2024	3 days

Figure 8:Design table

4.4 Implementation Table

Task	Start Date	End Date	Duration
Implementation planning	31.03.2024	04.04.2024	5 days
Implement the user interface	05.04.2024	19.04.2024	15 days
Integrate the modules	20.04.2024	24.04.2024	5 days

Figure 9:Implementation table

4.5 Testing Table

Task	Start Date	End Date	Duration
Preparing tests	25.04.2024	29.04.2024	5 days
Test the modules seperates	30.04.2024	12.05.2024	13 days
Prepare the test document	13.05.2024	17.05.2024	5 days

Figure 10:Testing table

6.UNIFIED MODELING LANGUAGE(UML)

6.1 Use-Case Diagram

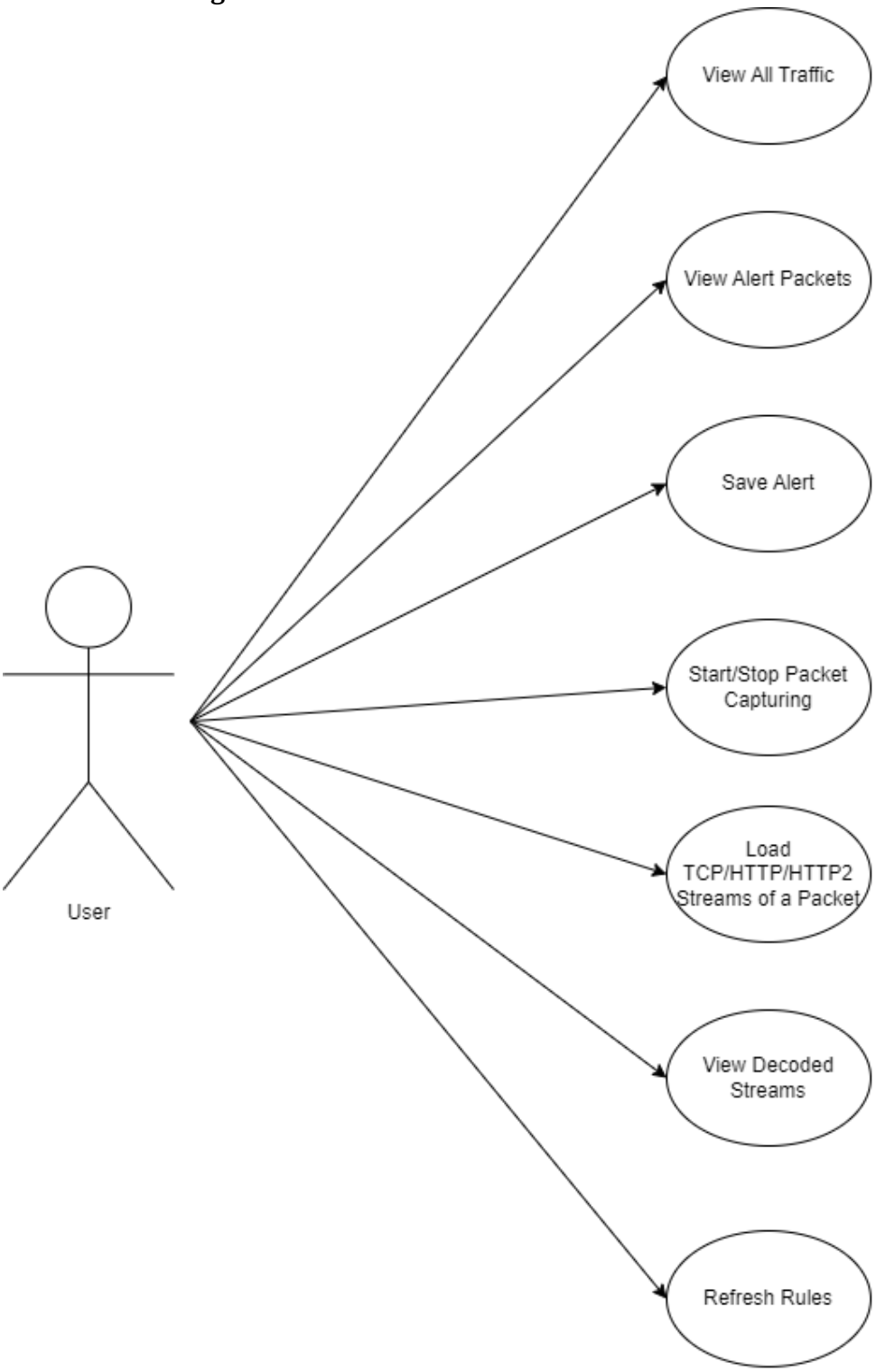

Figure 11:Use-Case Diagram

6.2 Activity Diagram

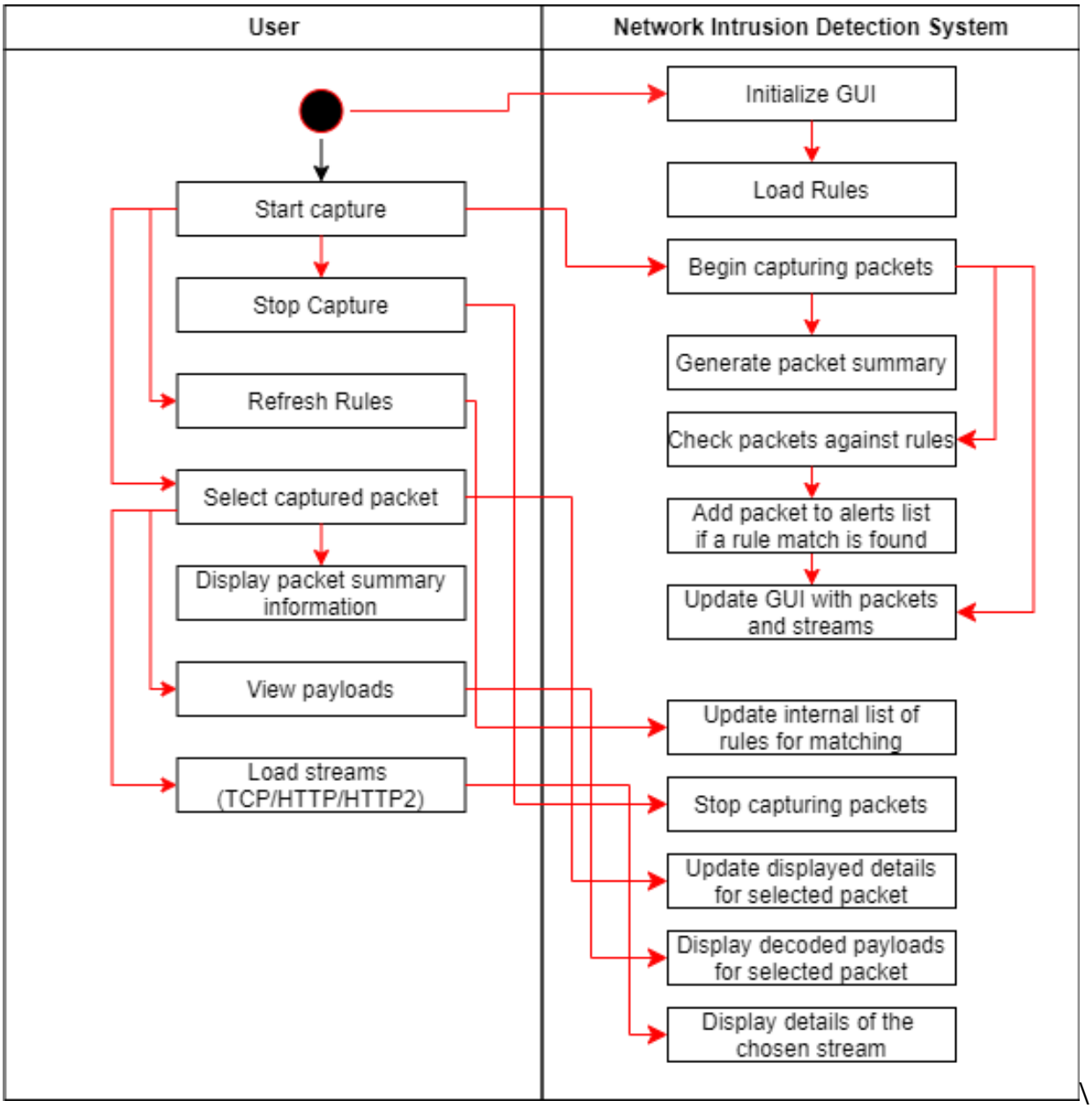

Figure 12:Activity Diagram

6.3 Sequence Diagram

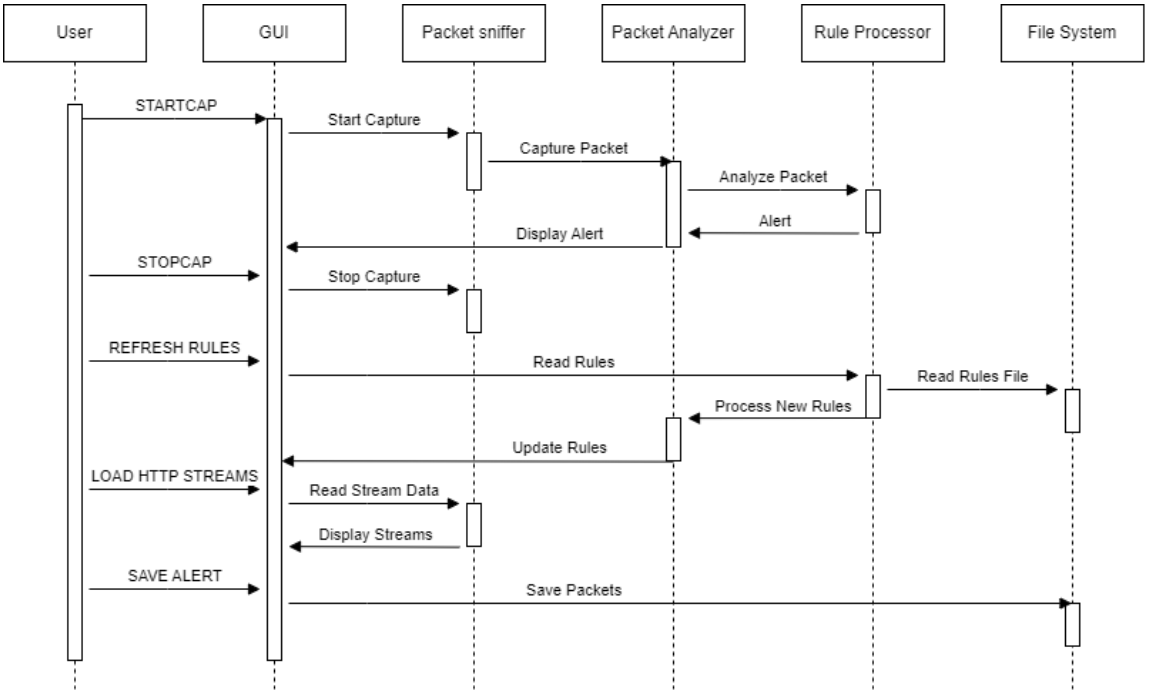

Figure 13:Sequence Diagram

6.4 Class Diagram

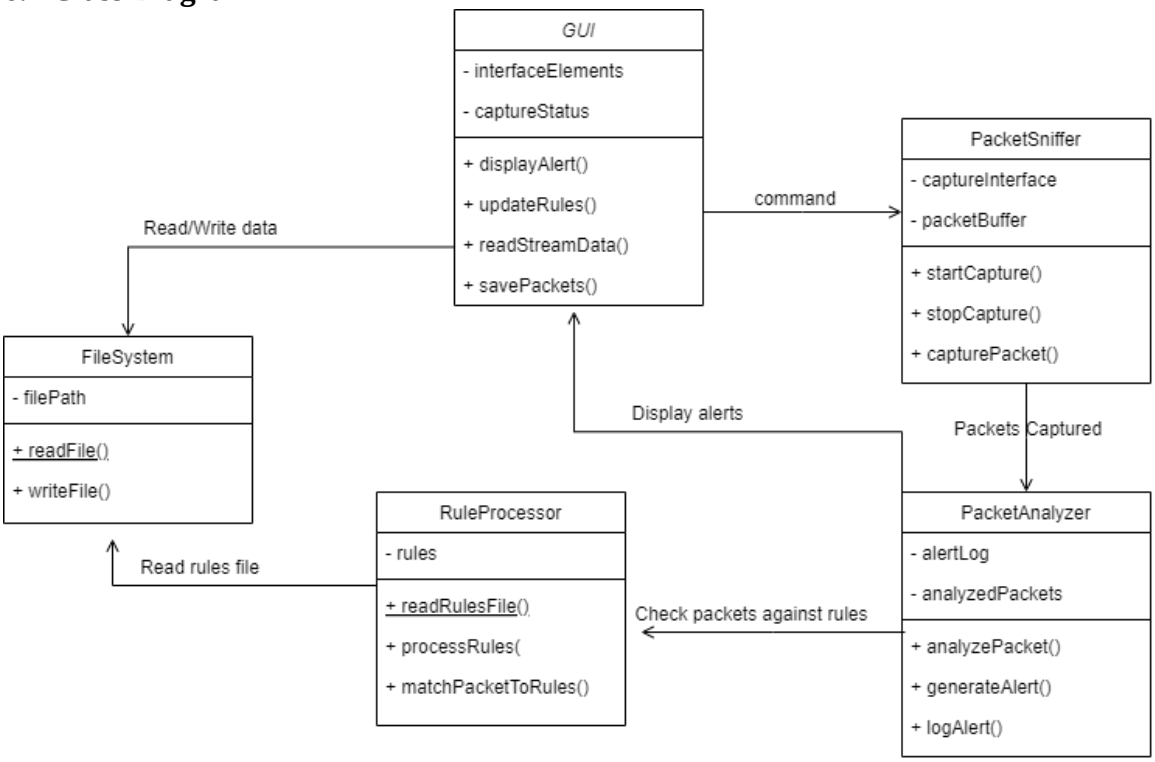

Figure 14:Class Diagram

7.USER INTERFACE DESIGN

7.1 Dashboard

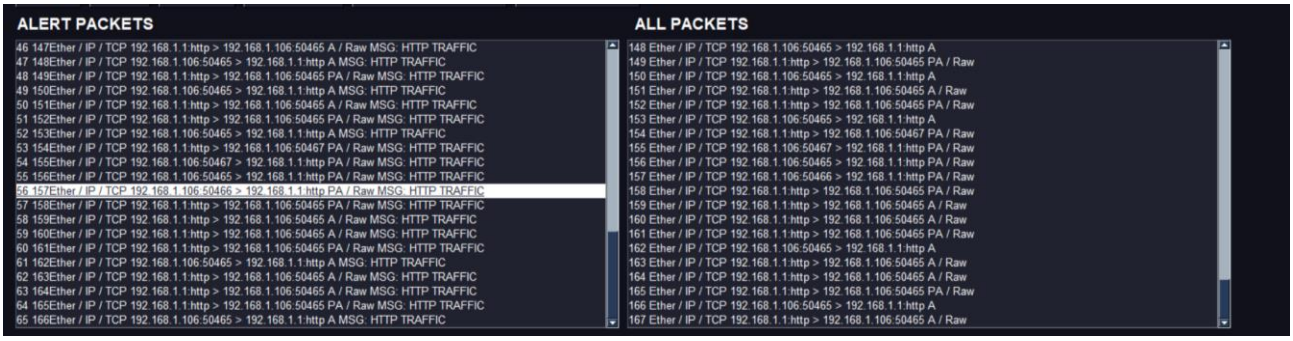

Figure 15:In-Sight Dashboard

The dashboard is the main interface where users can get an overview of the system status and recent activities. Key elements include:

- **Real-time Traffic Overview**: A graphical representation of the current network traffic, typically displayed as a line graph or bar chart.
- **Alerts Summary**: A section displaying recent alerts with details.

7.2 Control Panel

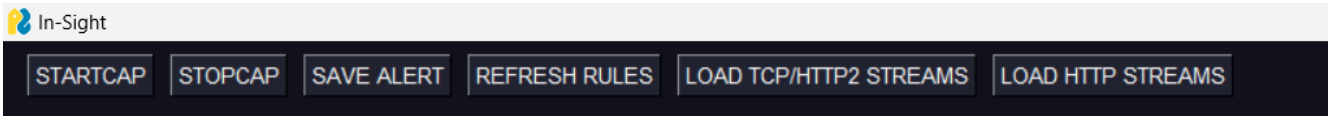

Figure 16:In-Sight Control Panel

The control panel allows users to manage the packet capturing process and other system settings. Key controls include:

- **Start Capture (StartCap)**: Initiates the packet capturing process. When capturing is active, the system continuously monitors network traffic.
- **Stop Capture (StopCap)**: A button to terminate the packet capturing process.
- **Save Alert**: Saves the current alerts, possibly to a file or a database for future reference.
- **Refresh Rules**: A button to reload the rules from the configuration file, ensuring the system uses the latest rules.
- **Load TCP/HTTP2 Streams**: A button to load and analyze streams that use the TCP or HTTP/2 protocols , providing detailed views of these streams.

- **Load HTTP Streams**: A button to load and analyze streams that use the HTTP protocol, providing detailed views of these streams.

7.3 Alert Decoded Window

Figure 17:Alert Decoded Window

The "Alert Decoded" window in the user interface of the Network Monitoring System provides users with a detailed view of the decoded payload of a packet that triggered an alert. This window plays a crucial role in aiding users' understanding of the content and context of the captured packet, helping them assess the severity and nature of the alert and take appropriate actions.

Packet Details Display:

- The window presents a comprehensive breakdown of the packet's content, including headers, payload, and any relevant metadata.
- Users can see decoded information such as source and destination IP addresses, port numbers, protocol type, and timestamp.

Hexadecimal and ASCII Representation:

- The payload of the packet is displayed in both hexadecimal and ASCII formats, allowing users to analyze the raw data and its corresponding text representation simultaneously.
- This dual representation facilitates deeper inspection of the packet content, especially for detecting anomalies or suspicious patterns.

Decoded Fields:

- The decoded payload may include various fields depending on the packet's protocol and content. For example, in an HTTP packet, users can see the request method, URL, headers, and body content.
- Each field is labeled and organized in a structured format, enhancing readability and comprehension.

Syntax Highlighting and Formatting:

- To improve readability and highlight important information, syntax highlighting and formatting techniques are applied to the decoded payload.
- Keywords, headers, and other significant elements are color-coded or styled differently, making them stand out against the rest of the content.

Navigation and Interaction:

- Users can navigate through the decoded payload easily, scrolling vertically to explore the entire content of the packet.
- Interactive features such as zooming, collapsing sections, and expanding details allow users to focus on specific areas of interest and analyze them in depth.

Contextual Information:

- Relevant contextual information, such as the alert description, matched rule, and severity level, is displayed alongside the decoded payload.

- This contextual information helps users correlate the packet content with the triggering event and understand the implications of the alert.

7.4 HTTP/2 Streams:

Figure 18:HTTP2 Streams window

Functionality: This window is dedicated to analyzing HTTP/2 streams, which are multiplexed streams of data exchanged between clients and servers over the HTTP/2 protocol.

Loading and Displaying Streams:

- When the user clicks the "Load HTTP/2 Streams" button in the UI, the system triggers a function that uses Pyshark to filter and extract HTTP/2 streams from the captured packets.
- Pyshark parses the network packets to identify HTTP/2 streams based on the protocol headers and stream identifiers.
- The extracted streams are then displayed in the HTTP/2 Streams window, typically in a tabular format. Each row represents a unique HTTP/2 stream and includes details such as stream ID, source/destination IP addresses, and payload size.

7.5 TCP Streams

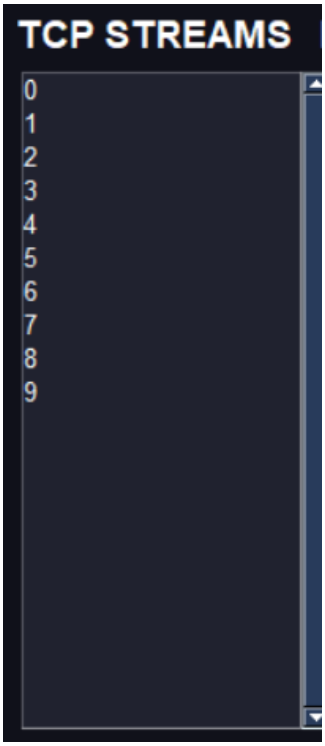

Figure 19:TCP Streams Window

Functionality: This window focuses on analyzing TCP streams, which are connections established between devices for transmitting data using the TCP/IP protocol suite.

Loading and Displaying Streams:

- Similar to the HTTP/2 Streams window, the user triggers the loading of TCP streams by clicking the "Load TCP Streams" button.
- Pyshark is utilized to filter and extract TCP streams from the captured packets based on TCP headers and connection identifiers.
- The extracted TCP streams are presented in the TCP Streams window, typically organized in a tabular format. Each row represents a TCP stream.

7.6 HTTP Objects

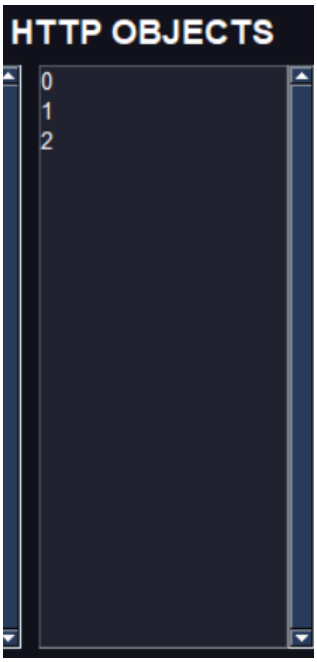

Figure 20:HTTP Objects Window

Functionality: This window focuses on analyzing individual HTTP objects, such as web pages, images, scripts, and other resources exchanged over HTTP connections.

Loading and Displaying Objects:

- When HTTP traffic is captured, Pyshark filters and extracts HTTP objects from the packets based on HTTP headers and content types.
- The HTTP objects, such as HTML pages, images, CSS files, and JavaScript files, are listed in the HTTP Objects window along with their associated metadata, including URLs, content types, and sizes.

7.7 TCP Stream

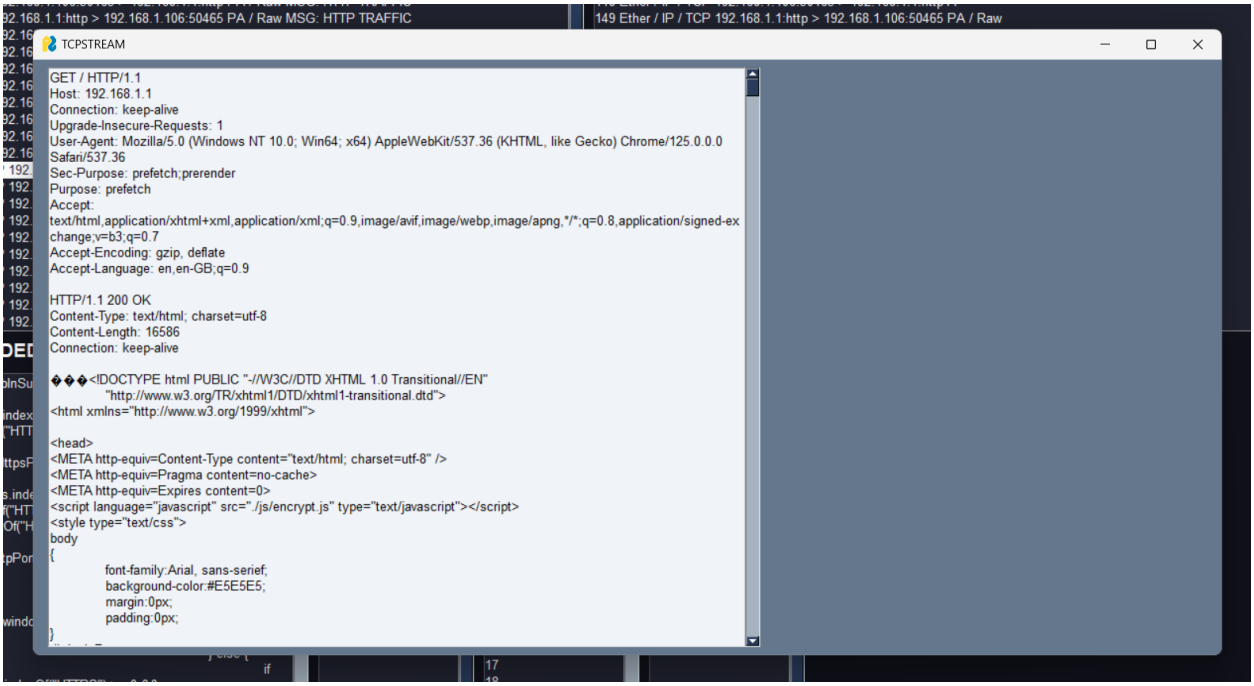

Figure 21: TCP Stream Window

Purpose: This window enables users to inspect the details of a selected TCP stream, offering insights into the underlying network communication and data transmission.

Functionality:

- Presents the complete data flow of the selected TCP stream, including transmitted packets, sequence numbers, and payload content.
- Decodes and interprets the TCP stream data, providing visibility into the exchanged information and any anomalies or irregularities.
- Offers features such as packet reassembly, packet sequence analysis, and payload inspection for comprehensive TCP stream analysis.

Usage:

- Users can inspect the TCP stream to identify any network-level issues, such as packet loss, retransmissions, or out-of-order delivery.
- Analyzing TCP streams aids in diagnosing network performance issues, troubleshooting connectivity problems, and detecting potential security threats.

7.8 HTTP2 Stream

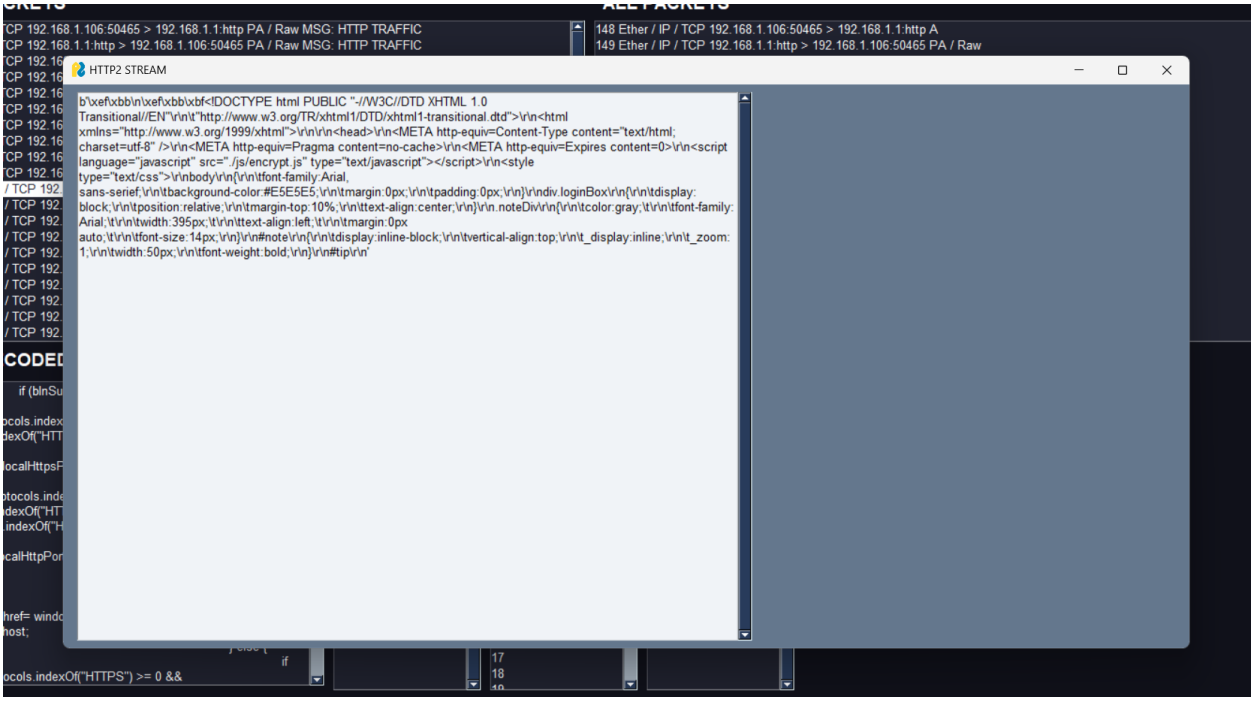

Figure 22: HTTP2 Stream Window

Purpose: The HTTP/2 Stream window offers detailed information about a selected HTTP/2 stream, allowing users to analyze the exchanged data packets and associated metadata.

Functionality:

- Displays the content of the selected HTTP/2 stream, including headers, payloads, and any associated metadata.
- Parses and decodes the HTTP/2 stream data to present it in a human-readable format, facilitating easy comprehension and analysis.
- Provides details such as stream identifiers, header fields, payload content, and other relevant information specific to the selected HTTP/2 stream.

Usage:

- Users can examine the contents of the HTTP/2 stream to understand the nature of the communication between the client and server.
- Analyzing the HTTP/2 stream data helps in troubleshooting issues related to web application performance, security, and protocol compliance.

7.9 HTTP Object

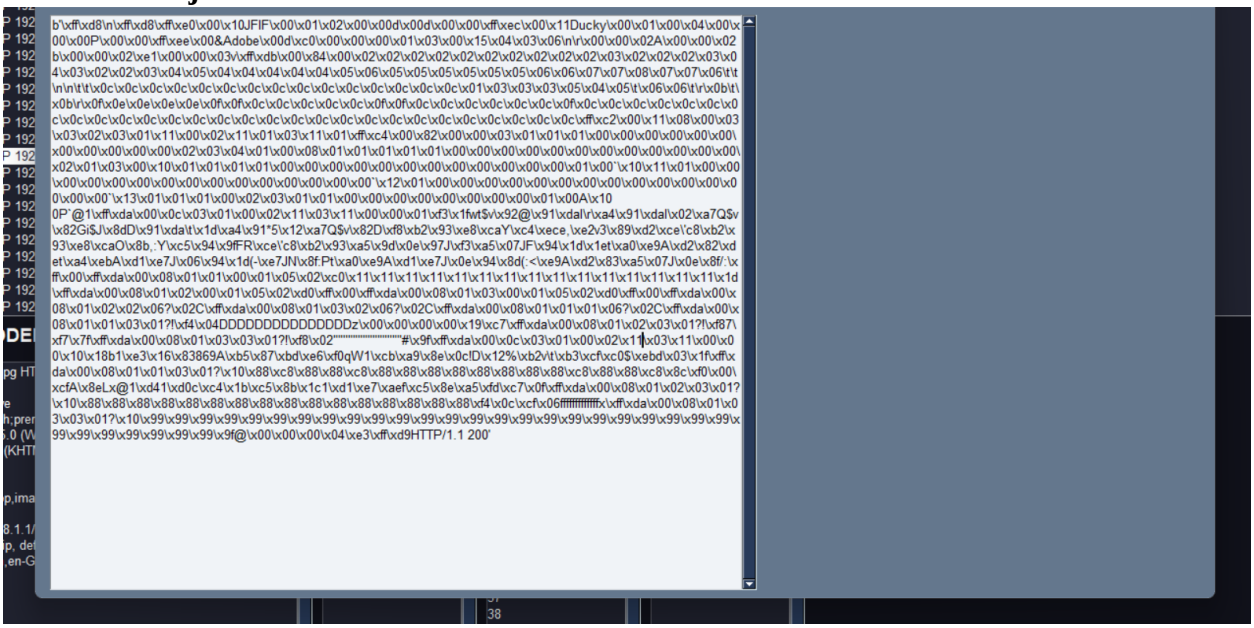

Figure 23: HTTP Object Window

Purpose: This window focuses on specific HTTP objects of a selected packet, such as web pages, images, or other resources.

Functionality:

- **Object Details**: Presents metadata about the HTTP object, including URL, content type, size, and headers.
- **Content Display**: Renders the content of the HTTP object in a human-readable format (e.g., HTML, JSON, images).
- **Analysis Tools**: Provides tools for further analysis, such as header inspection and content decoding.

Example: The provided content represents a binary JPEG image. When this object is selected:

- **Hexadecimal View**: Displays the raw binary data of the image.
- **Decoded View**: Shows the image in its visual form if it's an image file.
- **Metadata**: Displays metadata such as Content-Type: image/jpeg, size, and other HTTP headers.

7.10 Integration with the Network Monitoring System

- These dedicated windows seamlessly integrate with the Network Monitoring System's user interface, providing users with a focused view of selected streams and objects for in-depth analysis.
- By opening these windows upon user selection, the system enhances the usability and effectiveness of network traffic analysis, enabling users to gain actionable insights and make informed decisions regarding network performance, security, and optimization.

Through these specialized windows, users can efficiently investigate HTTP/2 streams, TCP streams, and HTTP objects, gaining a deeper understanding of network behavior and facilitating effective network management and troubleshooting.

8. SUMMARY AND CONCLUSIONS

This Network Monitoring System project successfully provides a comprehensive tool for capturing, analyzing, and displaying network traffic data. By leveraging Scapy, and Pyshark for network traffic capture, the system offers a user-friendly interface that enables detailed examination of network interactions.

The Control Panel facilitates essential functions such as starting and stopping captures, saving alerts, refreshing rules, and loading specific streams. This makes it easy for users to manage network traffic effectively. The system's ability to display and analyze active HTTP Objects, as well as TCP and HTTP/2 streams of captured packets provides valuable insights into network activities.

One of the key features is the detailed presentation of HTTP objects, which includes both raw data and decoded content. This allows users to gain a deeper understanding of the data being transmitted over the network. Additionally, the system automates alerting and filtering through rules file, enhancing its monitoring capabilities, and providing flexibility, and the option to customize the system based on user needs and requirements.

Overall, the project meets its objectives by delivering robust capabilities for network traffic analysis. It empowers network administrators and security professionals to monitor and troubleshoot network issues effectively.

Looking ahead, there are several opportunities for enhancing the system. Future improvements could include scalability to handle larger volumes of traffic, advanced analytics with machine learning for predictive insights, and continuous user interface enhancements to improve the user experience.

In conclusion, this Network Monitoring System is a valuable tool for network analysis and monitoring. It demonstrates significant potential for further development and integration in network security and management, making it a critical asset for professionals in the field.

9.REFERENCES

https://realpython.com/python-network-monitoring/

https://kiminewt.github.io/pyshark/

https://www.wireshark.org/docs/wsug_html_chunked/

https://ieeexplore.ieee.org/document/6522520

https://dl.acm.org/doi/10.1145/3298666

https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-557822.html

https://www.oreilly.com/library/view/building-network-tools/9781492024421/

https://www.packtpub.com/product/python-network-programming-cookbook-second-edition/9781786463999

https://realpython.com/python-sockets/

https://www.udemy.com/course/python-network-programming/

https://www.oreilly.com/library/view/effective-python-pen/9781783989300/

https://en.wikipedia.org/wiki/Network_monitoring

https://scapy.readthedocs.io/en/latest/

https://pysimplegui.readthedocs.io/en/latest/

10. Appendices

The appendices section contains the complete source code and supplementary materials used in the implementation of the project. This includes detailed scripts for rule reading and processing, packet handling, and the graphical user interface. The provided code snippets illustrate the integration of various Python libraries such as Scapy, PySimpleGUI, PyShark, and the socket library to achieve comprehensive network traffic analysis and alerting functionalities. Each code segment is designed to be modular and extensible, allowing for easy modification and enhancement to suit specific requirements or to integrate additional features. Below are the full scripts used in the project for reference.

10.1 nids.py

```
import scapy.all as scp
import codecs
import PySimpleGUI as sg
import os
import threading
import sys
import pyshark
import socket
import scapy.arch.windows as scpwinarch
import json
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
import re
import ipaddress

#Rules reading function
def readrules():
    rulefile = "rules.txt"
    ruleslist = []
    with open(rulefile, "r") as rf:
        ruleslist = rf.readlines()
    rules_list = []
    for line in ruleslist:
        if line.startswith("alert"):
            rules_list.append(line)
    print(rules_list)
    return rules_list
```

```
#Lists for each segment of the rule for easier parsing
alertprotocols = []
alertdestips = []
alertsrcips = []
alertsrcports = []
alertdestports = []
alertmsgs = []

# rule format --> "alert [srcip] [srcport] --> [dstip] [dstport] [msg]" [msg] may
include spaces and is not within quotes


#Rule processing function

def process_rules(rulelist):
    global alertprotocols
    global alertdestips
    global alertsrcips
    global alertsrcports
    global alertdestports
    global alertmsgs
    alertprotocols = []
    alertdestips = []
    alertsrcips = []
    alertsrcports = []
    alertdestports = []
    alertmsgs = []
    for rule in rulelist:
        rulewords = rule.split()
        if rulewords[1] != "any":
            protocol = rulewords[1]
            alertprotocols.append(protocol.lower())
        else:
            alertprotocols.append("any")
        if rulewords[2] != "any":
            srcip = rulewords[2]
            alertsrcips.append(srcip.lower())
        else:
            alertsrcips.append("any")
        if rulewords[3] != "any":
            srcport = int(rulewords[3])
            alertsrcports.append(srcport)
        else:
            alertsrcports.append("any")
```

```
        if rulewords[5] != "any":
            destip = rulewords[5]
            alertdestips.append(destip.lower())
        else:
            alertdestips.append("any")
        if rulewords[6] != "any":
            destport = rulewords[6]
            alertdestports.append(destport.lower())
        else:
            alertdestports.append("any")
        try:
            alertmsgs.append(" ".join([rulewords[x] for x in range(7,
len(rulewords))]))
        except:
            pass

    print(alertprotocols)
    print(alertdestips)
    print(alertsrcips)
    print(alertsrcports)
    print(alertdestports)
    print(alertmsgs)

process_rules(readrules())

deviceiplist = []
for route in scp.read_routes():
    if str(route[4]) not in deviceiplist:
        deviceiplist.append(str(route[4]))
        print(str(route[4]))



pktsummarylist = []
suspiciouspackets = []
suspacketactual = []
lastpacket = ""
sus_readablepayloads = []
all_readablepayloads = []
tcpstreams = []
SSLLOGFILEPATH = "C:\\Users\\yazee\\ssl1.log"
http2streams=[]
logdecodedtls = True
httpobjectindexes = []
httpobjectactuals = []
```

```
httpobjecttypes = []

#User Interface design
layout = [[sg.Button('STARTCAP', key="-startcap-", button_color='#20222F'),
        sg.Button('STOPCAP', key='-stopcap-', button_color='#20222F'),
sg.Button('SAVE ALERT', key='-savepcap-', button_color='#20222F'),
        sg.Button('REFRESH RULES', key='-refreshrules-', button_color='#20222F'),
        sg.Button('LOAD TCP/HTTP2 STREAMS', key='-showtcpstreamsbtn-',
button_color='#20222F'),
        sg.Button('LOAD HTTP STREAMS', key='-showhttpstreamsbtn-',
button_color='#20222F'),],
        [sg.Text("ALERT PACKETS", font=('Arial Bold', 14), size=(60, None),
justification="left",background_color='#10111A'),
         sg.Text("ALL PACKETS", font=('Arial Bold', 14), size=(60, None),
justification="left",background_color='#10111A')],
        [sg.Listbox(key='-pkts-', size=(100,20), values=suspiciouspackets,
enable_events=True, background_color='#20222F',text_color='white'),
        sg.Listbox(key='-pktsall-', size=(100,20), values=pktsummarylist,
enable_events=True, background_color='#20222F',text_color='white'),
        ],
        [sg.Text("ALERT DECODED", font=('Arial Bold', 14), size=(35,
None),justification="left",background_color='#10111A'),
         sg.Text("HTTP2 STREAMS", font=('Arial Bold',
14),justification="left",background_color='#10111A'),
         sg.Text("TCP STREAMS", font=('Arial Bold',
14),justification="left",background_color='#10111A'),
         sg.Text("HTTP OBJECTS", font=('Arial Bold',
14),justification="left",background_color='#10111A'),],
        [sg.Multiline(size=(60,20), key='-payloaddecoded-
',background_color='#20222F',text_color='white'),
        sg.Listbox(key='-http2streams-', size=(20, 20), values=http2streams,
enable_events=True,background_color='#20222F',text_color='white'),
        sg.Listbox(key='-tcpstreams-', size=(20,20), values=tcpstreams,
enable_events=True,background_color='#20222F',text_color='white'),
        sg.Listbox(key='-httpobjects-', size=(20, 20), values=httpobjectindexes,
enable_events=True,background_color='#20222F',text_color='white'),
        #sg.Multiline(size=(50,20), key='-payloaddecodedall-')
        ],
        [sg.Button('EXIT')]]

window = sg.Window('In-Sight', layout, size=(1600,800), resizable=True,
background_color='#10111A')

updatepktlist = False
pkt_list = []
```

```
def get_http_headers(http_payload):
    try:
        headers_raw = http_payload[:http_payload.index(b"\r\n\r\n") + 2]
        headers = dict(re.findall(b"(?P<name>.*?): (?P<value>.*?)\\r\\n",
headers_raw))

    except ValueError as err:
        logging.error('Could not find \\r\\n\\r\\n - %s' % err)
        return None
    except Exception as err:
        logging.error('Exception found trying to parse raw headers - %s' % err)
        logging.debug(str(http_payload))
        return None

    if b"Content-Type" not in headers:
        logging.debug('Content Type not present in headers')
        logging.debug(headers.keys())
        return None
    return headers

def extract_object(headers, http_payload):
    object_extracted = None
    object_type = None

    content_type_filters = [b'application/x-msdownload', b'application/octet-
stream']

    try:
        if b'Content-Type' in headers.keys():
            object_extracted = http_payload[http_payload.index(b"\r\n\r\n") +4:]
            object_type = object_extracted[:2]
            logging.info("Object Type: %s" % object_type)

        else:
            logging.info('No Content Type in Package')
            logging.debug(headers.keys())

        if b'Content-Length' in headers.keys():
            logging.info( "%s: %s" % (b'Content-Lenght', headers[b'Content-
Length']))
    except Exception as err:
```

```
        logging.error('Exception found trying to parse headers - %s' % err)
        return None, None
    return object_extracted, object_type

def read_http():
    objectlist = []
    objectsactual = []
    objectsactualtypes = []
    objectcount = 0
    global pkt_list
    try:
        os.remove(f".\\temp\\httpstreamread.pcap")
    except:
        pass
    httppcapfile = f".\\temp\\httpstreamread.pcap"
    scp.wrpcap(httppcapfile, pkt_list)
    pcap_flow = scp.rdpcap(httppcapfile)
    sessions_all = pcap_flow.sessions()

    for session in sessions_all:
        http_payload = bytes()
        for pkt in sessions_all[session]:
            if pkt.haslayer("TCP"):
                if pkt["TCP"].dport == 80 or pkt["TCP"].sport == 80 or
pkt["TCP"].dport == 8080 or pkt["TCP"].sport == 8080:
                    if pkt["TCP"].payload:
                        payload = pkt["TCP"].payload
                        http_payload += scp.raw(payload)
        if len(http_payload):
            http_headers = get_http_headers(http_payload)

            if http_headers is None:
                continue

            object_found, object_type = extract_object(http_headers,
http_payload)

            if object_found is not None and object_type is not None:
                objectcount += 1
                objectlist.append(objectcount-1)
                objectsactual.append(object_found)
                objectsactualtypes.append(object_type)

    return objectlist, objectsactual, objectsactualtypes
```

```
#Functions to check if a packet needs to be flagged

def proto_name_by_num(proto_num):
    for name,num in vars(socket).items():
        if name.startswith("IPPROTO") and proto_num == num:
            return name[8:]
    return "Protocol not found"

def check_rules_warning(pkt):
    global alertprotocols
    global alertdestips
    global alertsrcips
    global alertsrcports
    global alertdestports
    global alertmsgs
    global sus_readablepayloads
    global updatepktlist

    if 'IP' in pkt:
        try:
            src = pkt['IP'].src
            dest = pkt['IP'].dst
            proto = proto_name_by_num(pkt['IP'].proto).lower()
            #print(proto)
            sport = pkt['IP'].sport
            dport = pkt['IP'].dport

            for i in range(len(alertprotocols)):
                flagpacket = False
                if alertprotocols[i] != "any":
                    chkproto = alertprotocols[i]
                else:
                    chkproto = proto
                if alertdestips[i] != "any":
                    chkdestip = alertdestips[i]
                else:
                    chkdestip = dest
                if alertsrcips[i] != "any":
                    chksrcip = alertsrcips[i]
                else:
                    chksrcip = src
                if alertsrcports[i] != "any":
                    chksrcport = alertsrcports[i]
                else:
                    chksrcport = sport
```

```
                if alertdestports[i] != "any":
                    chkdestport = alertdestports[i]
                else:
                    chkdestport = dport


                if "/" not in str(chksrcip).strip() and "/" not in
str(chkdestip).strip():
                    if (str(src).strip() == str(chksrcip).strip() and
str(dest).strip() == str(chkdestip).strip() and str(proto).strip() ==
str(chkproto).strip() and str(dport).strip() == str(chkdestport).strip() and
str(sport).strip() == str(chksrcport).strip()):
                        flagpacket = True
                if "/" in str(chksrcip).strip() and "/" in
str(chkdestip).strip():
                    if (ipaddress.IPv4Address(str(src).strip()) in
ipaddress.IPv4Network(str(chksrcip).strip()) and
ipaddress.IPv4Address(str(dest).strip()) in
ipaddress.IPv4Network(str(chkdestip).strip()) and str(proto).strip() ==
str(chkproto).strip() and str(dport).strip() == str(chkdestport).strip() and
str(sport).strip() == str(chksrcport).strip()):
                        flagpacket = True
                if "/" in str(chksrcip).strip() and "/" not in
str(chkdestip).strip():
                    if (ipaddress.IPv4Address(str(src).strip()) in
ipaddress.IPv4Network(str(chksrcip).strip()) and str(dest).strip() ==
str(chkdestip).strip() and str(proto).strip() == str(chkproto).strip() and
str(dport).strip() == str(chkdestport).strip() and str(sport).strip() ==
str(chksrcport).strip()):
                        flagpacket = True
                if "/" not in str(chksrcip).strip() and "/" in
str(chkdestip).strip():
                    if (str(src).strip() == str(chksrcip).strip() and
ipaddress.IPv4Address(str(dest).strip()) in
ipaddress.IPv4Network(str(chkdestip).strip()) and str(proto).strip() ==
str(chkproto).strip() and str(dport).strip() == str(chkdestport).strip() and
str(sport).strip() == str(chksrcport).strip()):
                        flagpacket = True

                if flagpacket == True:
                        # print("Match")
                    if proto == "tcp":
                        try:
                            readable_payload =
bytes(pkt['TCP'].payload).decode('UTF8','replace')
```

```
                            sus_readablepayloads.append(readable_payload)
                        except Exception as ex:
                            sus_readablepayloads.append("Error getting tcp
payload!!")
                            print(ex)
                            pass
                    elif proto == "udp":
                        try:
                            readable_payload =
bytes(pkt['UDP'].payload).decode('UTF8','replace')
                            sus_readablepayloads.append(readable_payload)
                        except Exception as ex:
                            sus_readablepayloads.append("Error getting udp
payload!!")
                            print(ex)
                            pass
                    else:
                        sus_readablepayloads.append("NOT TCP PACKET!!")
                    if updatepktlist:
                        window['-payloaddecoded-
'].update(value=sus_readablepayloads[len(suspiciouspackets)])
                    return True, str(alertmsgs[i])
        except:
            pkt.show()

    return False, ""


def pkt_process(pkt):
    global deviceiplist
    global window
    global updatepktlist
    global suspiciouspackets
    global all_readablepayloads

    pkt_summary = pkt.summary()

    pktsummarylist.append(f"{len(pktsummarylist)} " + pkt_summary)
    pkt_list.append(pkt)
    sus_pkt, sus_msg = check_rules_warning(pkt)
    if sus_pkt == True:
        suspiciouspackets.append(f"{len(suspiciouspackets)} {len(pktsummarylist)
- 1}" + pkt_summary + f" MSG: {sus_msg}")
        suspacketactual.append(pkt)
```

```
    return

# Identify Wi-Fi interface name
def get_wifi_interface():
    interfaces = scpwinarch.get_windows_if_list()
    for interface in interfaces:
        if "Wi-Fi" in interface["name"]:
            return interface["name"]
    return None

# Use Wi-Fi interface for packet capturing
wifi_interface = get_wifi_interface()
if wifi_interface:
    sniffthread = threading.Thread(target=scp.sniff, kwargs={"prn":pkt_process,
"filter": "", "iface": wifi_interface}, daemon=True)
    sniffthread.start()
else:
    print("Wi-Fi interface not found.")

def show_tcp_stream_openwin(tcpstreamtext):
    layout = [[sg.Multiline(tcpstreamtext, size=(100,50), key="tcpnewwintext")]]
    window = sg.Window("TCPSTREAM", layout, modal=True, size=(1200, 600),
resizable=True)
    choice = None
    while True:
        event, values = window.read()
        if event == "Exit" or event == sg.WIN_CLOSED:
            break
    window.close()

def show_http2_stream_openwin(tcpstreamtext):
    layout = [[sg.Multiline(tcpstreamtext, size=(100,50), key="tcpnewwintext")]]
    window = sg.Window("HTTP2 STREAM", layout, modal=True, size=(1200, 600),
resizable=True)
    choice = None
    while True:
        event, values = window.read()
        if event == "Exit" or event == sg.WIN_CLOSED:
            break
    window.close()

def load_tcp_streams(window):
    global http2streams
    global logdecodedtls
```

```
    try:
        os.remove(f".\\temp\\tcpstreamread.pcap")
    except:
        pass
    scp.wrpcap(f".\\temp\\tcpstreamread.pcap", pkt_list)
    global tcpstreams
    tcpstreams = []
    tcpstreamfilename = ".\\temp\\tcpstreamread.pcap"
    cap1 = pyshark.FileCapture(
        tcpstreamfilename,
        display_filter="tcp.seq==1 && tcp.ack==1 && tcp.len==0",
        keep_packets=True)
    number_of_streams = 0
    for pkt in cap1:
        if pkt.highest_layer.lower() == "tcp" or pkt.highest_layer.lower() ==
"tls":
            print(pkt.tcp.stream)
            if int(pkt.tcp.stream) > number_of_streams:
                number_of_streams = int(pkt.tcp.stream) + 1
    for i in range(0, number_of_streams):
        tcpstreams.append(i)
    window["-tcpstreams-"].update(values=[])
    window["-tcpstreams-"].update(values=tcpstreams)

    if logdecodedtls == True:
        http2streams = []
        cap2 = pyshark.FileCapture(
        tcpstreamfilename,
        display_filter="http2.streamid",
        keep_packets=True)
        #numberofhttp2streams = 0
        for pkt in cap2:
            field_names = pkt.http2._all_fields
            for field_name in field_names:
                http2_stream_id = {val for key, val in field_names.items() if key
== 'http2.streamid'}
                http2_stream_id = "".join(http2_stream_id)
            if http2_stream_id not in http2streams:
                http2streams.append(http2_stream_id)
        window['-http2streams-'].update(values=http2streams)
        pass
```

```
def show_http2_stream(window, streamno):

    tcpstreamfilename = ".\\temp\\tcpstreamread.pcap"
    cap3 = pyshark.FileCapture(
            tcpstreamfilename,
            display_filter = f'http2.streamid eq {str(http2streamindex)}',
            override_prefs={'ssl.keylog_file': SSLLOGFILEPATH}
        )
    #print(cap3[0].http2.stream)
    dat = ""
    decode_hex = codecs.getdecoder("hex_codec")
    http_payload = bytes()
    for pkt in cap3:
        # for x in pkt[pkt.highest_layer]._get_all_field_lines():
        #     print(x)
        #try:
        try:
            payload = pkt["TCP"].payload
            http_payload += scp.raw(payload)
            #does literally nothing because we do not know the encoding format of
the payload so scp.raw returns type error
        except:
            pass

        print(pkt.http2.stream)
        if ("DATA" not in pkt.http2.stream):
            http2headerdat = ''
            rawvallengthpassed = False
            print(pkt.http2._all_fields.items())
            for field, val in pkt.http2._all_fields.items():
                if rawvallengthpassed == False:
                    if field == 'http2.header.name.length':
                        rawvallengthpassed = True
                else:
                    #if field.split(".")[-1] != "headers":
                    http2headerdat += str(field.split(".")[-1]) + " : " +
str(val) + " \n"
                    print(http2headerdat)
            dat += "\n" + http2headerdat
    if len(http_payload):
        http_headers = get_http_headers(http_payload)

        if http_headers is not None:
            object_found, object_type = extract_object(http_headers,
http_payload)
```

```
            dat += object_type + "\n" + object_found + "\n"


    print(dat)
    formatteddat = dat
    print(formatteddat)

    show_http2_stream_openwin(formatteddat)
    pass

def show_tcpstream(window, streamno):
    global SSLLOGFILEPATH
    tcpstreamfilename = ".\\temp\\tcpstreamread.pcap"
    streamnumber = streamno
    cap = pyshark.FileCapture(
        tcpstreamfilename,
        display_filter = 'tcp.stream eq %d' % streamnumber,
        override_prefs={'ssl.keylog_file': SSLLOGFILEPATH}
    )
    dat = b""
    decode_hex = codecs.getdecoder("hex_codec")
    for pkt in cap:
        try:
            payload = pkt.tcp.payload
            encryptedapplicationdata_hex =
"".join(payload.split(":")[0:len(payload.split(":"))])
            encryptedapplicationdata_hex_decoded =
decode_hex(encryptedapplicationdata_hex)[0]
            dat += encryptedapplicationdata_hex_decoded
            print(encryptedapplicationdata_hex_decoded)
        except Exception as ex:
            print(ex)

    formatteddat = str(dat, "ascii", "replace")

    if formatteddat.strip() == "" or len(str(formatteddat.strip)) < 1:
        sg.PopupAutoClose("No data")
    else:
        show_tcp_stream_openwin(formatteddat)

def show_saved_message():
    sg.popup("Alert saved")
```

```
def show_refreshed_message():
    sg.popup("Rules have been refreshed")

while True:

    print(suspiciouspackets)

    event, values = window.read()
    if event == '-refreshrules-':
        process_rules(readrules())
        show_refreshed_message()

    if event == "-startcap-":
        updatepktlist = True
        incomingpacketlist = []
        inc_pkt_list = []
        suspiciouspackets = []
        suspacketactual = []
        pktsummarylist = []
        sus_readablepayloads = []
        while True:
            event, values = window.read(timeout=10)
            if event == "-stopcap-":
                updatepktlist = False
                break
            if event == '-refreshrules-':
                process_rules(readrules())
            if event == sg.TIMEOUT_EVENT:
                #window['-pkts-'].update(pktsummarylist,
scroll_to_index=len(pktsummarylist))
                window['-pkts-'].update(suspiciouspackets,
scroll_to_index=len(suspiciouspackets))
                window['-pktsall-'].update(pktsummarylist,
scroll_to_index=len(pktsummarylist))
                #window['-payloaddecoded-
'].update(value=sus_readablepayloads[len(suspiciouspackets)])
            if event in (None, 'Exit'):
                sys.exit()
                break
            if event == '-pkts-' and len(values['-pkts-']):     # if a list item
is chosen
                sus_selected = values['-pkts-']
                #sus_selected_index = int(sus_selected.split()[0][0:2])
                sus_selected_index = values[event][0]
                try:
```

```
                    window["-tcpstreams-
"].update(scroll_to_index=int(suspacketactual[sus_selected_index].tcp.stream))
                except:
                    pass
                window['-payloaddecoded-
'].update(value=sus_readablepayloads[sus_selected_index ])
            if event == '-pktsall-' and len(values['-pktsall-']):     # if a list
item is chosen
                #pktselected = values['-pktsall-']
                pkt_selected_index = window["-pktsall-"].get_indexes()
                try:
                    window["-tcpstreams-
"].update(scroll_to_index=int(pkt_list[pkt_selected_index].tcp.stream))
                except:
                    pass

            if event == "-showtcpstreamsbtn-":
                load_tcp_streams(window)
            if event == "-tcpstreams-":
                streamindex = window["-tcpstreams-"].get_indexes()
                show_tcpstream(window, streamindex)
            if event == "-http2streams-":
                http2streamindex = values[event][0]
                show_http2_stream(window, int(http2streamindex))
            if event == "-showhttpstreamsbtn-":
                httpobjectindexes = []
                httpobjectactuals = []
                httpobjecttypes = []
                httpobjectindexes, httpobjectactuals, httpobjecttypes =
read_http()
                window["-httpobjects-"].update(values=httpobjectindexes)
            if event == "-httpobjects-":
                httpobjectindex = values[event][0]
                show_http2_stream_openwin(httpobjecttypes[httpobjectindex] +
b"\n" + httpobjectactuals[httpobjectindex][:900])


    if event == "-showhttpstreamsbtn-":
        httpobjectindexes = []
        httpobjectactuals = []
        httpobjecttypes = []
        httpobjectindexes, httpobjectactuals, httpobjecttypes = read_http()
        window["-httpobjects-"].update(values=httpobjectindexes)

    if event == "-httpobjects-":
```

```
        httpobjectindex = values[event][0]
        show_http2_stream_openwin(httpobjecttypes[httpobjectindex] + b"\n" +
httpobjectactuals[httpobjectindex][:900])

    if event == "-http2streams-":
        http2streamindex = values[event][0]
        print(http2streamindex)
        show_http2_stream(window, str(int(http2streamindex)))
    if event == '-pktsall-' and len(values['-pktsall-']):     # if a list item is
chosen
        #pktselected = values['-pktsall-']
        pkt_selected_index = window["-pktsall-"].get_indexes()[0]
        try:
            window["-tcpstreams-
"].update(scroll_to_index=int(pkt_list[pkt_selected_index].tcp.stream))
        except:
            pass
    if event == '-savepcap-':
        show_saved_message()
        pcapname = "nettrafic"
        scp.wrpcap(f'.\\savedpcap\\{pcapname}.pcap', inc_pkt_list)
    if event == '-pkts-' and len(values['-pkts-']):     # if a list item is
chosen
        sus_selected = values['-pkts-']
        #sus_selected_index = int(sus_selected.split()[0][0:2])
        sus_selected_index = window['-pkts-'].get_indexes()[0]
        try:
            window["-tcpstreams-
"].update(scroll_to_index=int(suspacketactual[sus_selected_index].tcp.stream))
        except:
            pass
        window['-payloaddecoded-
'].update(value=sus_readablepayloads[sus_selected_index])
    if event == "-showtcpstreamsbtn-":
        load_tcp_streams(window)
    if event == "-tcpstreams-":
        streamindex = window["-tcpstreams-"].get_indexes()
        show_tcpstream(window, streamindex)
    if event in (None, 'Exit'):
        break


window.close()
```

10.2 decrypthttp2.py

```
import pyshark
import os
from scapy.utils import RawPcapWriter

key_path = "C:\\Users\\yazee\\ssl1.log"
pcap_file = 'tcpstreamread.pcap'

cap = pyshark.FileCapture(pcap_file,
                          display_filter="http2.streamid eq 5",
                          override_prefs={'ssl.keylog_file': key_path})

# Collect packets into a list
packet_list = [packet for packet in cap]

# Path for saving pcap file
save_directory = './savedpcap'
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

pcapname = "nettrafic"
pcap_file_path = os.path.join(save_directory, f'{pcapname}.pcap')

# Save the pcap file
with RawPcapWriter(pcap_file_path) as writer:
    for packet in packet_list:
        writer.write(packet.raw_data)
```

10.3 Rules.txt

!alert udp any any -> any 53 DNS DNS DNS

!alert udp any 53 -> any any DNS DNS DNS

!alert tcp 192.168.0.0/24 any -> any any OUTGOING HOME NET RANGE READ

!alert udp any any -> any any UDP ALERT

alert tcp any any -> 192.168.0.0/24 any INCOMING HOME NET RANGE READ

alert tcp any any -> any 8080 HTTP TRAFFIC

alert tcp any 80 -> any any HTTP TRAFFIC

alert tcp any any -> any 80 HTTP TRAFFIC