

The Advanced Packet Capture and Analysis Tool

COMP 8047 – Major Project

HyungJoon LEE – A01204540
3-26-2024

Table of Contents

1.	Introduction	2
1.1.	Student Background	2
1.2.	Acknowledgements	2
1.3.	Project Description	2
1.3.1.	Essential Problems	3
1.3.2.	Goals and Objectives.....	3
2.	Body	4
2.1.	Background.....	4
2.2.	Project Statement	4
2.3.	Possible Alternative Solutions	5
2.4.	Chosen Solution.....	6
2.5.	Details of Design and Development	7
2.6.	Testing Details and Results.....	18
2.7.	Implications of Implementation.....	29
2.8.	Innovation	30
2.9.	Complexity.....	31
2.10.	Research in New Technologies.....	32
2.11.	Future Enhancements	33
2.12.	Timeline and Milestones	35
3.	Conclusion.....	41
3.1.	Lessons Learned	42
3.2.	Closing Remarks	43
4.	Appendix	44
4.1.	Approved Proposal	44
4.2.	Project Supervisor Approvals	44
5.	References	45
6.	Change Log.....	45

1. Introduction

1.1. Student Background

I am a dedicated student with a strong focus on Network Security Application Development. Throughout my academic journey, I have honed my skills in various aspects of network security, with a particular emphasis on socket programming, raw socket handling, and packet analyzing. My extensive experience in C programming has allowed me to tackle complex challenges in this field with confidence. I have a proven track record of developing secure and efficient network applications during course work. My academic pursuits align seamlessly with my passion for network security, and I continually seek out innovative solutions to stay at the forefront of this ever-evolving cybersecurity landscape.

1.2. Acknowledgements

Reflecting on the journey that has led me to the culmination of my studies in Network Security Application Development, I am filled with immense gratitude for the guidance and support by D'Arcy Smith. Over the course of three transformative years, starting from my foundational studies in Data Communication & Internetworking option in diploma, progressing through to the complexities of network security, D'Arcy Smith has been an unwavering pillar of knowledge, encouragement, and inspiration.

I am deeply thankful for the opportunity to have been taught by D'Arcy Smith. His guidance has been a guiding light throughout my studies, illuminating the path to achieving my academic and professional goals. As I move forward in my career, I carry with me the valuable lessons and insights gained from his mentorship, confident in the knowledge that I am well-prepared to make meaningful contributions to the field of network security.

Thank you, D'Arcy Smith for your remarkable dedication, patience, and for believing in me even when I doubted myself. Your influence extends beyond the classroom and into the lives you touch, shaping the future of your students for the better. I am honored to have been one of those students.

I extend my sincere gratitude to Borna Nouredin for his invaluable guidance and mentorship throughout the development of my project. His dedication to providing insightful advice and support, from the initial proposal writing to the final stages, has been instrumental in my success. I am truly appreciative of the time and expertise he has shared with me.

1.3. Project Description

The project aims to make network packet analysis easier and more user-friendly, addressing the complex and technical aspects of Wireshark. The project aims to enhance the visualization of packet data, akin to Wireshark, by reading .pcap files and improving the presentation of data. It incorporates a color-coded system within data tables to highlight warnings and risks, informed by Snort 2 or 3 rules, which simplifies the identification of potential security issues. Moreover, it

streamlines packet classification through a user-friendly drag-and-drop interface, bypassing the need for complex commands required by Wireshark. Users can effortlessly categorize packets by selecting options from a dropdown menu, such as TCP packets, specific IPs, or ports, enhancing ease of use. An optional feature includes the capability for real-time packet analysis, permitting users to examine packets as they are captured, similar to Wireshark. This tool is designed to make the process of analyzing network traffic more accessible and efficient for all users, not just those with expertise in network security, thereby addressing some of Wireshark's usability challenges and broadening its applicability to a wider users.

1.3.1. Essential Problems

In today's digital landscape, where data moves through networks at an unprecedented scale, the ability to analyze network packet traffic is key to detecting potential cybersecurity threats. Wireshark, a traditional standard in this realm, offers robust analysis capabilities but comes with significant limitations. One such limitation is the tool's insufficient visual threat detection; it does not inherently provide visual cues to identify threats in packet data. This deficiency forces network security professionals to rely on the integration of additional systems for threat visualization, a process that is often manual and cumbersome. Additionally, Wireshark's dependence on complex command-line syntax can be daunting for those without deep technical knowledge, creating a steep learning curve. This complexity can act as a barrier, preventing effective use of the tool for network analysis and potentially hindering timely threat detection and response.

1.3.2. Goals and Objectives

The goals/objectives of this project focus on making network analysis more intuitive and accessible. The primary goal is to enhance threat visualization by developing a system that provides immediate visual insights into security risks within packet capture (.pcap) files. This system will place a special emphasis on identifying threats that adhere to Snort 3 rules, thereby streamlining the threat detection process. In terms of user experience, the initiative seeks to revolutionize packet classification by introducing a simplified, drag-and-drop interface for the packet analyzer. This will help reduce reliance on complex command-line-like syntax, making the tool more inclusive and easier to use for people with varying levels of technical expertise. Additionally, the project aims to create dynamic and informative visual representations of packet data. These visualizations will bring to light traffic patterns and anomalies, significantly improving the user's capability to monitor and safeguard network traffic effectively.

2. Body

2.1. Background

The project introduces a groundbreaking approach to make network analysis and intrusion detection systems (IDS) more user-friendly, particularly for those without deep expertise in network or security domains. By integrating Snort3, an open-source network intrusion detection system, with Wireshark, a popular network protocol analyzer, the initiative seeks to overcome several challenges faced by users of these tools when used separately. Wireshark, despite being a robust tool for packet analysis and network troubleshooting, often intimidates users with its complex interface and the need to remember intricate filtering syntax. Furthermore, the traditional separation of network analysis and intrusion detection activities requires analysts to toggle between different systems and manually correlate data, complicating the detection and understanding of network threats.

This project aims to make these powerful tools more accessible to a broader audience, enhancing the efficiency of security analysts by streamlining their workflows. This reduces the time and effort needed to analyze network traffic and identify security threats. Additionally, the integration serves as an educational platform for students and professionals new to network analysis and security, offering a more approachable way to develop their skills. This initiative not only promotes accessibility but also ensures that the depth and quality of analysis are not compromised, meeting the increasing demand for tools that simplify complex technical processes.

2.2. Project Statement

The project aims to develop an integrated network analysis and intrusion detection platform by combining the capabilities of Snort3, an open-source network intrusion detection system (IDS), with Wireshark, a comprehensive network protocol analyzer. This integration seeks to address the complexity and accessibility challenges faced by users, particularly those without extensive expertise in networking or security. The project introduces a novel user interface enhancement through the implementation of drop-down filters, significantly simplifying the packet filtering process compared to the complex syntax required by standalone Wireshark. By doing so, the project not only makes network analysis more approachable but also enhances the effectiveness of real-time security monitoring and threat detection.

The objective of the project is to develop an integrated platform that merges the packet analysis capabilities of Wireshark with the intrusion detection features of Snort3. This new platform will be enhanced with a user-friendly interface, incorporating drop-down filters for packet filtering to make network analysis and intrusion detection accessible to users at all levels of expertise, thereby enhancing security monitoring and response efficiency.

The challenges this project addresses include the complexity of packet analysis, which it simplifies by making the filtering process more accessible to non-experts. It also aims to bridge the gap between network analysis and intrusion detection, which are typically conducted using separate tools, by integrating Snort3 and Wireshark into a single, cohesive platform. Additionally, the project enhances the detection of security threats by incorporating visual alerts from Snort3 within the Wireshark interface, enabling immediate identification and response to potential intrusions.

Key features of this integrated platform include a unified interface that combines the deep packet inspection capabilities of Wireshark with the intrusion detection prowess of Snort3. It introduces a user-friendly packet filtering system that utilizes a drop-down method, eliminating the need to remember and type complex filter syntax. The platform also features real-time visual intrusion alerts within the analysis interface to highlight potential security threats, facilitating quicker response actions.

The impact and significance of this project are profound, as it stands to significantly enhance the field of network security by making advanced tools more accessible and efficient for a wider audience. It will not only serve as a practical tool for seasoned professionals but also as an invaluable learning resource for students and newcomers to the field of network security. The integration of Snort3 and Wireshark, coupled with the innovative drop-down filter feature, represents a significant advancement in the democratization of network analysis and security practices.

2.3. Possible Alternative Solutions

Exploring alternative solutions to the challenge of simplifying network analysis and intrusion detection reveals various approaches that could complement or compete with the project's integration of Snort3 and Wireshark. These alternatives offer different strategies for making network security tools more accessible and user-friendly, each with its own set of advantages and potential limitations. The project explores several innovative approaches to simplify network analysis and intrusion detection, each with distinct advantages and limitations.

One approach involves developing customized graphical user interfaces (GUIs) that act as wrappers around existing tools like Wireshark and Snort. These interfaces aim to provide simplified controls, preset configurations, and guided workflows, helping users manage complex functionalities without extensive technical knowledge. The advantages of this method include its simplicity, which significantly lowers the entry barrier for new users, and its flexibility, allowing users to toggle between simplified and advanced views. However, there are limitations such as development overhead, as creating and maintaining custom wrappers requires additional effort, and performance overhead, which might impact the efficiency of the underlying tools.

Another approach is the use of cloud-based network analysis services. These services enable users to upload packet captures (.pcap) or direct network traffic to a simplified web interface for analysis, leveraging advanced backend algorithms without the need for local installations. The key advantages here include accessibility from anywhere and scalability, as cloud services can adjust resources based on demand. Nonetheless, potential drawbacks include concerns over data privacy and security when sending sensitive network data to a third party, as well as a dependence on reliable internet connectivity, which may not always be feasible, especially in secure or sensitive environments.

Lastly, the project considers implementing advanced AI-assisted analysis tools that use artificial intelligence (AI) and machine learning (ML) to automate the analysis of network traffic and the

detection of security threats. AI helps identify patterns, anomalies, and potential threats with minimal human intervention, offering a summarized and prioritized view of potential issues. This method enhances efficiency, as AI can process data much faster than human operators, and adaptability, with machine learning algorithms improving over time to better detect new threats. However, it also presents challenges such as the complexity involved in developing and training effective AI algorithms and the potential for false positives or negatives, where AI might incorrectly categorize benign or malicious activities.

Each of these solutions addresses the need to simplify network analysis and intrusion detection from different angles, providing a range of options for organizations and individuals aiming to enhance their capabilities in these critical areas. The selection among these alternatives would depend on specific needs, resources, and constraints, emphasizing the importance of a well-considered strategy in choosing the appropriate solution.

2.4. Chosen Solution

As a student undertaking a graduation project focused on simplifying network analysis and intrusion detection, the chosen solution involves developing customized graphical user interfaces (GUIs) that act as wrappers around existing tools like Wireshark and Snort. This approach seeks to blend the powerful capabilities of these established tools with a more accessible and intuitive user experience. Below is a detailed description of the chosen solution and the rationale behind this decision.

The chosen solution for the project involves developing customized graphical user interfaces (GUIs) using Qt6 that serve as wrappers for Wireshark and Snort3. These GUIs are crafted to offer simplified controls, preset configurations, and guided workflows, helping users effectively navigate and utilize the complex functionalities of these tools without requiring deep technical knowledge. The interfaces include a dual-view system that allows users to switch between a simplified view for beginners and an advanced view for experienced users. Additionally, the integration of visual alerts for intrusion detection leverages Snort3's capabilities within the Wireshark analysis environment, enhancing the detection and response to network threats.

The key features of these GUI wrappers encompass simplified controls and preset configurations to make the tools more accessible to new users, along with guided workflows to assist in performing complex analyses. Users can customize their experience based on their skill level by switching between simplified and advanced views, and the inclusion of visual alerts for effective intrusion detection enriches the overall functionality.

The rationale behind choosing this solution centers on several key aspects. First, the project aims to significantly lower the entry barrier to network analysis and intrusion detection, making these powerful tools accessible to a broader audience, including those without specialized knowledge in network or security fields. It also offers flexibility and customization, catering to both novices and experts by providing different interface views and guided workflows. This approach is resource and time-efficient, focusing on enhancing the usability of existing tools rather than building new capabilities from scratch, which is particularly practical given the constraints of a graduation project. Moreover, the project serves an educational purpose, acting as a practical learning tool for students

and professionals new to network security, enabling them to engage with complex tools in a more user-friendly environment.

The development of these GUI wrappers is feasible within the scope of a graduation project and promises to deliver significant value by improving the usability of widely-used network security tools, thereby enhancing network security practices among a wider audience. By prioritizing the simplification of the user experience without sacrificing the advanced capabilities of Wireshark and Snort, this solution effectively bridges a critical gap in the domain of network analysis and security. It represents a pragmatic and impactful approach that aligns well with the project's objectives and the student's capabilities, aiming to make a tangible difference in how network security tools are accessed and utilized.

2.5. Details of Design and Development

1) Installation

The very first thing to do is make sure all necessary dependencies are installed. The following is a list of required packages:

- cmake to build from source
- The Snort 3 libdaq for packet IO
<https://github.com/snort3/libdaq>
- dnet for network utility functions
<https://github.com/dugsong/libdnet.git>
- flex >= 2.6.0 for JavaScript syntax parsing
- g++ >= 5 or other C++14 compiler
- hwloc for CPU affinity management
<https://www.open-mpi.org/projects/hwloc/>
- LuaJIT for configuration and scripting
<http://luajit.org>
- OpenSSL for SHA and MD5 file signatures, the protected_content rule option, and SSL service detection
<https://www.openssl.org/source/>
- pcap for tcpdump style logging
<http://www.tcpdump.org>

- pcre for regular expression pattern matching
<http://www.pcre.org>
- pkgconfig to locate build dependencies
<https://www.freedesktop.org/wiki/Software/pkg-config/>
- zlib for decompression
<http://www.zlib.net>

Installing LibDAQ

We now need to install the Snort 3 LibDAQ, which provides an abstraction layer for communicating with a data source (such as a network interface). If you have LibDAQ already installed for Snort 2 and want to install a DAQ just for Snort 3, or if you want to install LibDAQ in a custom location, you can change the DAQ install location with the `--prefix` option when configuring: `./configure --prefix=/usr/local/lib/daq_s3`.

To show this in action, first clone the LibDAQ repository from GitHub:

```
$ git clone https://github.com/snort3/libdaq.git
```

Then, run the following commands to generate the configure script, configure with a specified prefix, build, and lastly install:

```
$ cd libdaq
```

```
$ ./bootstrap
```

```
$ ./configure --prefix=/usr/local/lib/daq_s3
```

```
$ make install
```

After installing libdaq, you must then run `ldconfig` to configure your system's dynamic linker run-time bindings. However, if you have installed the DAQ in a nonstandard location, you'll first need to tell your system where to find the new shared libraries. One common solution is to create a file in the `/etc/ld.so.conf.d/` directory that points to where those libraries are located:

```
$ cat /etc/ld.so.conf.d/libdaq3.conf /usr/local/lib/daq_s3/lib/
```

Once ready you may proceed with the `ldconfig` command to configure the run-time bindings:

```
$ sudo ldconfig
```

Building Snort

After all dependencies have been installed, it is time to build Snort. To do this, first clone the Snort 3 repository:

```
$ git clone https://github.com/snort3/snort3.git
```

You can choose to install Snort in the system-default directories, or you can specify to install it in some other directory with the `--prefix=<path>` command line argument.

```
$ export my_path=/path/to/snorty
```

```
$ mkdir -p $my_path
```

```
$ cd snort3
```

```
$ ./configure_cmake.sh --prefix=$my_path
```

Additionally, if the LibDAQ has been installed in a non-standard or custom location, then you must include the `--with-daq-libraries` and `--with-daq-includes` arguments and set them accordingly.

```
$ ./configure_cmake.sh --prefix=$my_path \  
    --with-daq-includes=/usr/local/lib/daq_s3/include/ \  
    --with-daq-libraries=/usr/local/lib/daq_s3/lib/
```

There are many more CMake configuration options to choose from (like enabling debug mode, for example), and the full list of options can be seen by running the following command:

```
$ ./configure_cmake.sh --help
```

Once you've configured CMake to your liking and the build files are ready to go, it's time to compile and install Snort. To do this, `cd` to the newly-created build directory, and then compile and install:

```
$ cd build
```

```
$ make -j $(nproc)
```

```
$ make install
```

Qt6

Download the Qt Online Installer:

Visit the Qt download page and download the Qt Online Installer for Linux.

Make the Installer Executable:

Open a terminal and navigate to the directory where the installer has been downloaded.

Make the installer executable by running:

```
$ chmod +x qt-unified-linux-x64-*.run
```

Run the Installer:

Execute the installer: `./qt-unified-linux-x64-*.run`

Follow the on-screen instructions.

Qt Account:

The installer will prompt you to log in with your Qt account or create a new one.

Select Components:

Select Qt 6 from the available options, along with any specific modules you require.

Installation:

Choose your installation directory and proceed with the installation.

Project File

```
$ git clone https://github.com/HyungJoonLEE/Lazyshark.git
```

```
$ cd Lazyshark
```

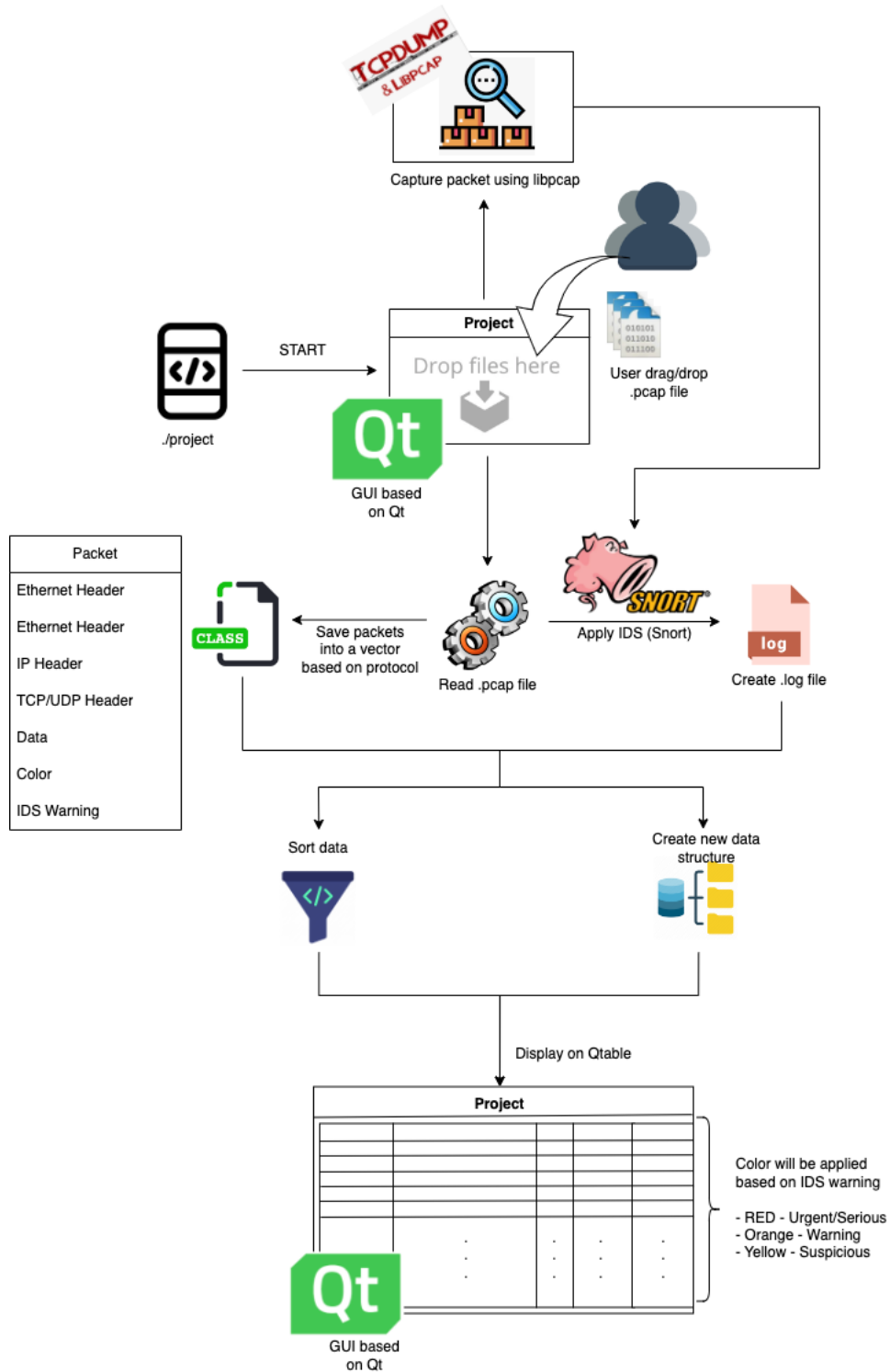
```
$ cmake -S . -B cmake-build-debug
```

```
$ cmake --build cmake-build-debug
```

```
$ cd cmake-build-debug
```

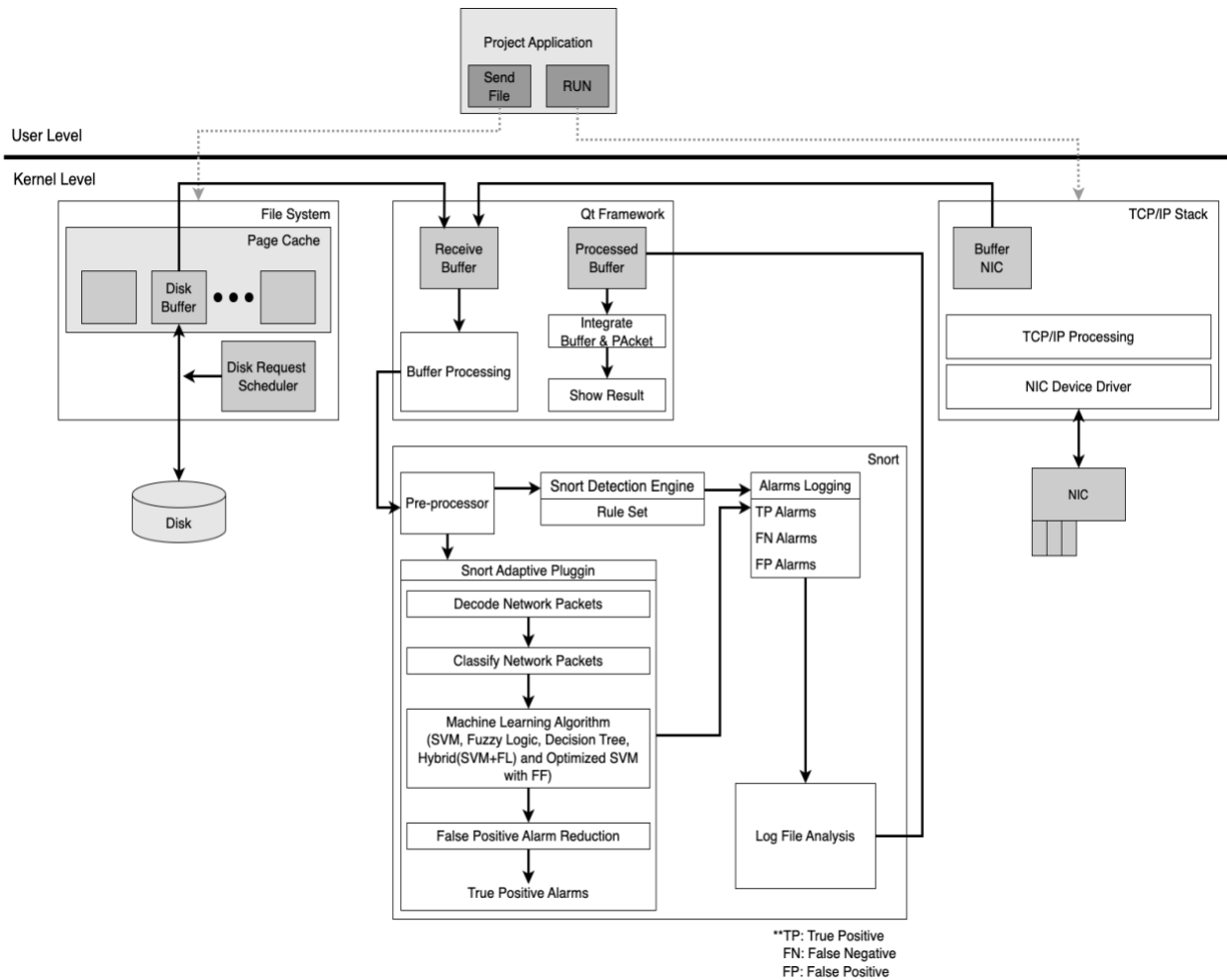
```
$ sudo ./Lazyshark
```

2) Design



The diagram illustrates the workflow of a Qt-based GUI application for network packet analysis. Initially, a user starts the application by executing a `./Lazyshark` command. Within the application's GUI, users can drag and drop .pcap files, which contain network packet data. The application leverages TCPDUMP and Libpcap for capturing packet data and employs Snort, an open-source intrusion detection system (IDS), to analyze packets for security threats. Upon processing, packets are sorted and saved into vectors according to their protocol, with distinct classes for different packet headers and data. Snort generates a .log file of its analysis. Subsequently, the application sorts the packet data and organizes it into a new table structure, which is then displayed in a QTable widget. The GUI enhances data readability by applying color codes—red for urgent/serious issues, orange for warnings, and yellow for suspicious activities—to the IDS warnings, enabling users to quickly identify and respond to potential security incidents. This flowchart maps out an efficient process for network traffic analysis and threat detection, integrating packet capturing, intrusion detection, and a user-friendly interface for data representation.

3) System/Software Architecture Diagram



This diagram presents an intricate architecture for a network packet analysis system that operates at both the kernel and user levels. At the user level, the process starts with the 'Project Application' where a user can initiate a 'Send File' to upload .pcap file to analysis or 'RUN' to capture packets in a live mode. This interacts with the 'Qt Framework' at the kernel level, where a 'Receive Buffer' collects the data and then processes it in the 'Processed Buffer'. The integration of the buffer and packet data is carried out, and the results are displayed to the user.

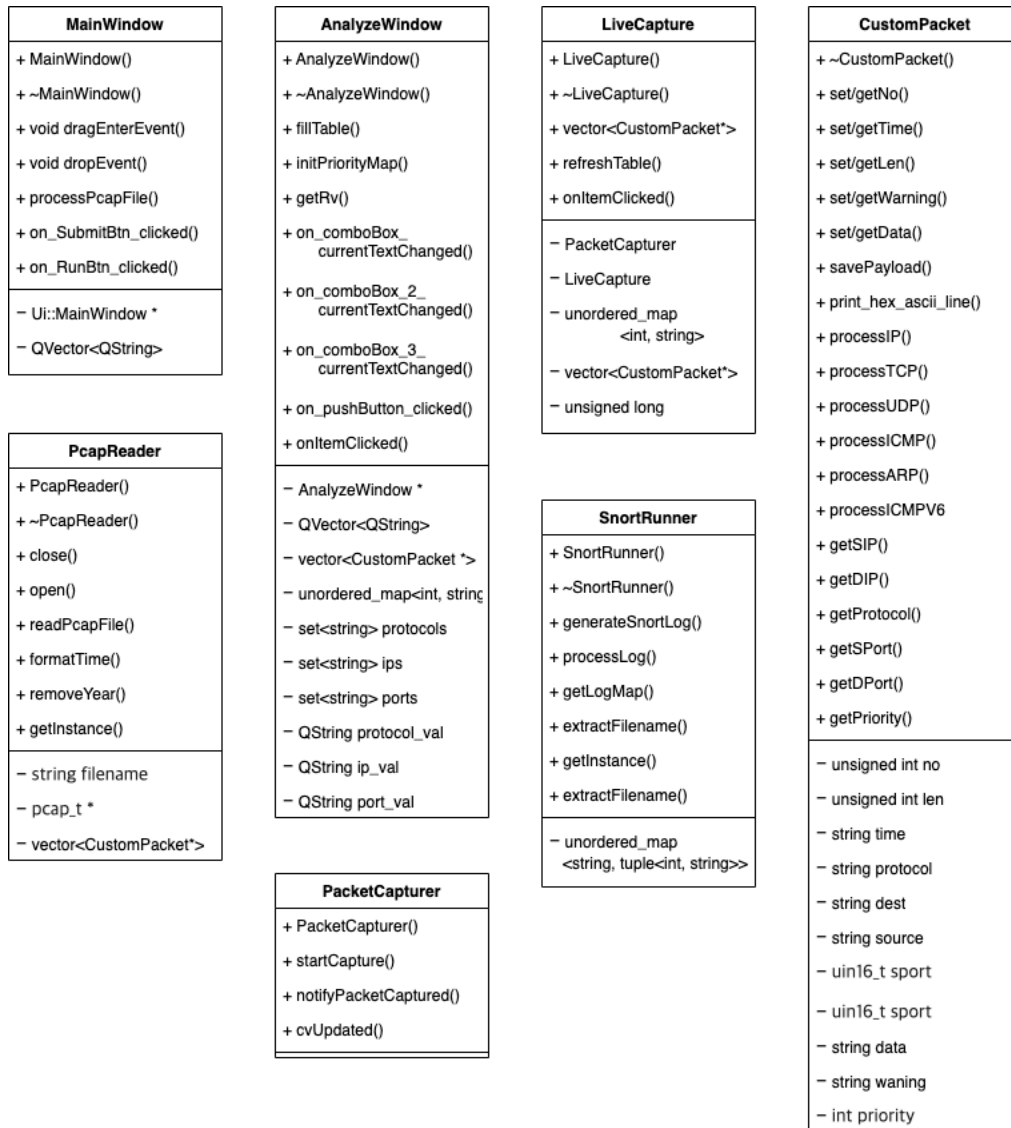
Simultaneously, the kernel-level workings involve the 'File System' with a 'Page Cache', a 'Disk Buffer', and a 'Disk Request Scheduler' that manages the interaction with the disk storage. On another front, the 'TCP/IP Stack' includes a 'Buffer NIC' for network interface controller (NIC) management, 'TCP/IP Processing', and the 'NIC Device Driver', which ensures proper communication with the NIC hardware.

The Qt Framework feeds into the 'Snort Detection Engine', which utilizes a 'Rule Set' and a 'Snort Adaptive Plugin' to preprocess the data. Snort decodes and classifies network packets, funneling them through a complex 'Machine Learning Algorithm' that may include techniques like SVM (Support Vector

Machine), Fuzzy Logic, Decision Trees, and optimized hybrid algorithms. This machine learning component is responsible for the 'False Positive Alarm Reduction', ensuring that the final output consists of 'True Positive Alarms' which indicate legitimate threats or anomalies.

Alarms are logged separately into categories of 'True Positives (TP)', 'False Negatives (FN)', and 'False Positives (FP)', with a focus on log file analysis for detailed examination. The intricate process is aimed at enhancing network security by meticulously filtering and analyzing network traffic to distinguish between benign and malicious packets with high accuracy, thus reducing the incidence of false alerts while efficiently detecting true threats.

4) Class Diagram



The provided image is a UML (Unified Modeling Language) class diagram for a network packet analysis application with several components each responsible for specific tasks within the system:

MainWindow: This class is responsible for the main operations of the GUI such as file drag-and-drop, processing of pcap files, and button event handling. It also maintains a list of strings, possibly for display or logging.

AnalyzeWindow: An extension of the MainWindow, this class seems to focus on analysis features, handling various UI elements like combo boxes for selections and a push button. It can also manipulate custom packet data and track protocols, IPs, and ports.

PcapReader: This class is tailored to handle reading pcap files which contain packet data captured over the network. It can open, read, and close pcap files, and includes formatting methods for time and removal of specific year data.

PacketCapturer: This class likely controls the live capturing of packets, starting the capture process, and notifying when new packets are captured. It also includes a method to update the UI based on new packet data.

LiveCapture: This class is to manage live packet capturing sessions, refreshing the display table with new packet data, and responding to user interactions within the live capture interface.

CustomPacket: Represents individual packets with methods to set and get various attributes such as number, length, data, and associated warnings. It also includes methods to save the payload and process different types of protocols (TCP, UDP, ICMP, etc.).

SnortRunner: A backend class possibly interfacing with the Snort application for intrusion detection. It can generate and process Snort logs, manage log maps, and extract file names. It's used to run Snort processes and handle the associated data.

CustomPacket: This class holds packet as objects, suggesting that the system can handle multiple packets at once, storing them for analysis and processing.

5) Design Pattern

In the development of a network analysis and intrusion detection application, the **Singleton design pattern** has been strategically applied to the **PcapReader** and **SnortRunner** classes. These implementations ensure that both crucial components, responsible for reading packet capture (pcap) files and running Snort for intrusion detection, respectively, are instantiated only once throughout the application's lifecycle. This report outlines the rationale, benefits, and key aspects of using the Singleton pattern for these classes.

Implementation Overview

The Singleton pattern was chosen for the PcapReader and SnortRunner classes to enforce a single point of control and access to these critical resources. Here's a brief overview of their implementations:

PcapReader Singleton

Handles the reading of pcap files, ensuring that packet data is read and processed in a consistent and controlled manner. The Singleton implementation guarantees that only one instance of the PcapReader manages pcap file access, preventing concurrent file access issues and ensuring efficient use of system resources.

```
PcapReader();
PcapReader(const PcapReader& ref);
PcapReader& operator=(const PcapReader& ref) {}
~PcapReader();

static PcapReader& getInstance() {
    static PcapReader pcapReader;
    return pcapReader;
}
```

SnortRunner Singleton

Manages the execution of the Snort intrusion detection system. Given the resource-intensive nature of running Snort, the Singleton ensures that only one instance of SnortRunner is active at any given time, providing a centralized control mechanism for initiating and managing intrusion detection processes.

```
SnortRunner();
SnortRunner(const SnortRunner& ref);
~SnortRunner();
SnortRunner& operator= (const SnortRunner& ref) {}

static SnortRunner& getInstance() {
    static SnortRunner snortRunner;
    return snortRunner;
}
```

The Singleton pattern for both PcapReader and SnortRunner is implemented by: Making the constructor private to prevent external instantiation. Deleting the copy constructor and copy assignment operator to prevent copying. Providing a static getInstance() method that returns a reference to the static instance of the class, ensuring that only one instance is created and accessible throughout the application.

Rationale and Benefits

In the context of resource management, utilizing the Singleton pattern with both PcapReader and SnortRunner is crucial due to their involvement in resource-intensive operations. This design approach ensures efficient management of resources by preventing the creation of multiple instances that could compete for the same resources. Additionally, employing a Singleton ensures consistency across the application, as all parts interact with a single instance of each class. This is particularly important for operations like reading pcap files and running intrusion detection analyses, where maintaining a consistent application state and behavior is essential. Furthermore, by providing a single access point to these services, the application can exert better control over operations, such as limiting the execution of one intrusion detection analysis at a time, thereby avoiding potential conflicts or resource contention. This controlled access helps in maintaining robustness and stability within the application's functionality.

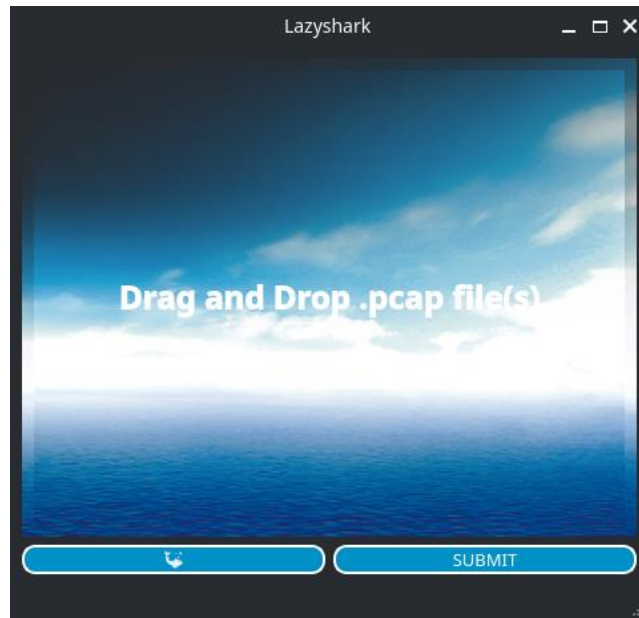
Considerations

While the Singleton pattern provides substantial benefits for managing resources in PcapReader and SnortRunner, it's crucial to also consider the potential drawbacks it brings to the table. Firstly, employing Singletons may limit the flexibility of the application's design. This could make it more challenging to extend or modify the behavior of these components, as the pattern restricts the instantiation of these classes to a single object. Secondly, Singletons can pose significant challenges in testing environments. Since they maintain state across different tests, they can lead to dependencies between tests, which can complicate test setups and potentially lead to less reliable testing outcomes. These considerations are important to weigh against the benefits to ensure that the Singleton pattern is the best choice for these components.

2.6. Testing Details and Results

1) User Interface Functionality Verification

The objective of this test case is to verify the functionality of two specific buttons in the application's user interface: the "Live Packet Capture" button, which is represented with a shark icon, and the "Submit" button, which is designed for reading .pcap files. To conduct this test, the first step is to launch the application and navigate to the main interface where these buttons are located. Once there, the tester needs to locate and identify the two targeted buttons - one marked with a shark icon for initiating "Live Packet Capture" mode, and the other labeled as "Submit," which is used for reading .pcap files. This process ensures that each button performs its designated function correctly and efficiently within the application.



Dropped file: `"/home/hj/Desktop/Lazyshark/pcaps/oct.pcap"`

This test case is to verify the functionality of two specific actions within the application's user interface: activating "Live Packet Capture" mode and reading .pcap files. The test begins with clicking the button marked with a shark icon, expected to initiate "Live Packet Capture" mode. Upon activation, the application should allow for real-time packet capturing, with changes in the button's appearance or a status message confirming the mode's activation. The second part of the test involves the .pcap file reading functionality. Here, the tester clicks the "Submit" button after selecting a .pcap file through the application's file selection interface. The expected result is that the application reads the selected .pcap file, either displaying its contents or analyzing the packets, depending on the application's functionality. Successful reading should be confirmed through a visual indicator or a message.

The testing environment needs to be controlled, ensuring the application has the necessary resources for live packet capturing and a valid .pcap file is available for the file reading test. The outcome criteria for success are based on the application's ability to enter "Live Packet Capture" mode and to read .pcap files as expected, with visual confirmations or successful execution of the intended functionalities serving as verification of proper operation.

2) .pcap File Analysis and Packet Detail Display

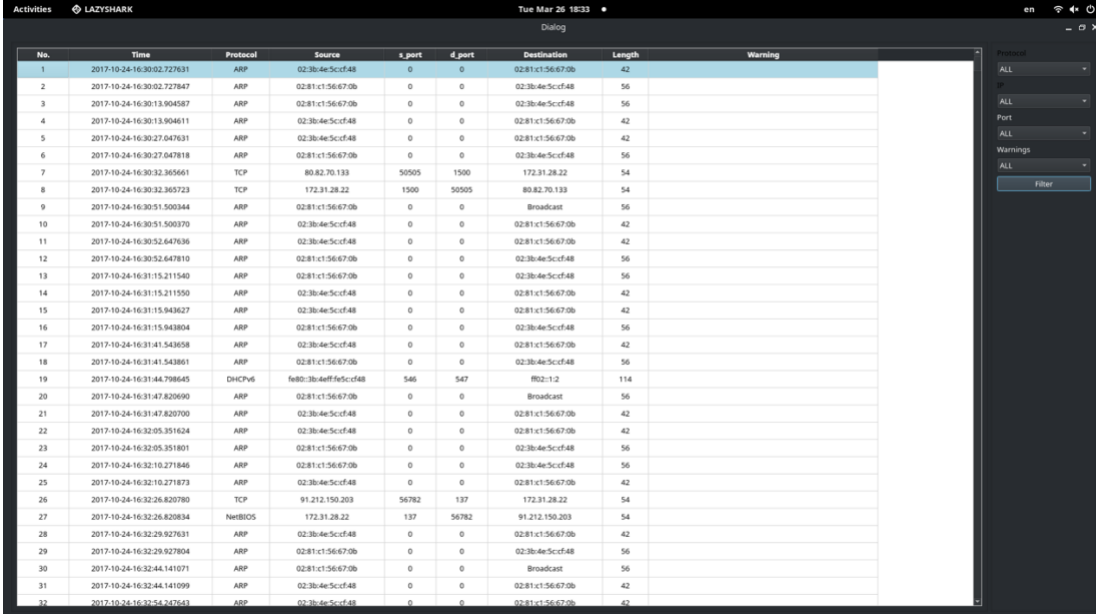
The objective of this test case is to validate the application's ability to read and display packet data from a .pcap file and ensure that it supports functionality allowing a user to view detailed information about individual packets. The test involves several key steps:

First, the test will start the application and load a .pcap file through the user interface. The expected outcome of this action is for the application to successfully read the .pcap file and display a summary of the packets in a tabular format. This table should include columns for packet number (No.), timestamp (Time), protocol (Protocol), source IP address (Source), source port (s_port), destination port (d_port), destination IP address (Destination), packet length (Length), and any warnings (Warning) related to each packet. This layout ensures that users can quickly grasp the essential packet data at a glance, facilitating easier analysis.

Additionally, the tester will verify that clicking on any packet in the summary table opens a new window or pane displaying detailed information about the selected packet. This functionality is crucial for in-depth packet analysis, allowing users to explore each packet's detailed attributes and data payloads, thereby enhancing the application's utility for network diagnostics and troubleshooting.

Overall, this test case assesses both the basic functionality of the .pcap file reading and the interactive capabilities of the application concerning detailed packet analysis. The success of the test is determined by the application's ability to accurately and efficiently display both summary and detailed packet data in an accessible and user-friendly manner.

Verify Summary of Packets



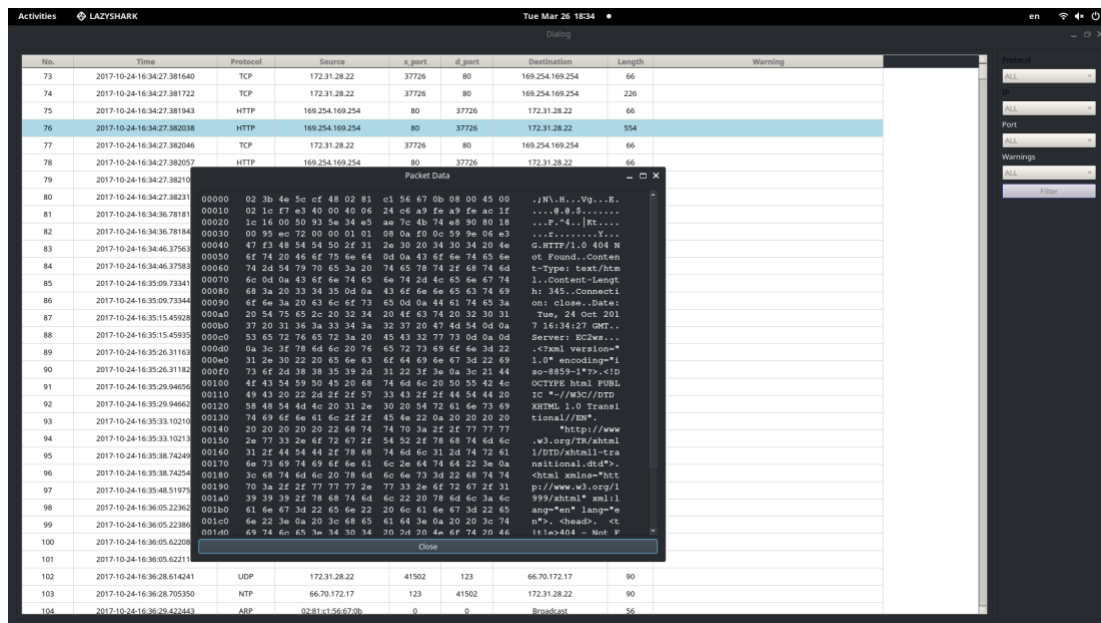
No.	Time	Protocol	Source	s_port	d_port	Destination	Length	Warning
1	2017-10-24-16:30:02.727631	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
2	2017-10-24-16:30:02.727847	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
3	2017-10-24-16:30:13.904587	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
4	2017-10-24-16:30:13.904611	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
5	2017-10-24-16:30:27.047631	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
6	2017-10-24-16:30:27.047818	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
7	2017-10-24-16:30:32.365661	TCP	80.82.70.133	50905	1500	172.31.28.22	54	
8	2017-10-24-16:30:32.365723	TCP	172.31.28.22	1500	50905	80.82.70.133	54	
9	2017-10-24-16:30:51.500344	ARP	02:81:c1:56:67:0b	0	0	Broadcast	56	
10	2017-10-24-16:30:51.500370	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
11	2017-10-24-16:30:52.647636	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
12	2017-10-24-16:30:52.647810	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
13	2017-10-24-16:31:15.211540	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
14	2017-10-24-16:31:15.211550	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
15	2017-10-24-16:31:15.943627	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
16	2017-10-24-16:31:15.943804	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
17	2017-10-24-16:31:41.543658	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
18	2017-10-24-16:31:41.543861	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
19	2017-10-24-16:31:44.798645	DHCPv6	fe80:3b:aef:fec:f4:8	546	547	ff02::1:2	114	
20	2017-10-24-16:31:47.820690	ARP	02:81:c1:56:67:0b	0	0	Broadcast	56	
21	2017-10-24-16:31:47.820790	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
22	2017-10-24-16:32:05.351624	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
23	2017-10-24-16:32:05.351801	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
24	2017-10-24-16:32:10.271846	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
25	2017-10-24-16:32:10.271873	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
26	2017-10-24-16:32:26.820780	TCP	91.212.150.203	56782	137	172.31.28.22	54	
27	2017-10-24-16:32:26.820834	NBIOOS	172.31.28.22	137	56782	91.212.150.203	54	
28	2017-10-24-16:32:29.927631	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
29	2017-10-24-16:32:29.927804	ARP	02:81:c1:56:67:0b	0	0	02:3b:4e:5c:f4:8	56	
30	2017-10-24-16:32:44.141071	ARP	02:81:c1:56:67:0b	0	0	Broadcast	56	
31	2017-10-24-16:32:44.141099	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	
32	2017-10-24-16:32:54.247643	ARP	02:3b:4e:5c:f4:8	0	0	02:81:c1:56:67:0b	42	

The test case continues with two further actions aimed at validating the functionality of the application's packet summary display and the interactive packet selection feature:

Firstly, the action involves observing the table populated with packet summaries after a .pcap file is loaded. The expected result is that each packet listed in the table should accurately reflect data corresponding to its network traffic attributes. This includes details such as protocol types (e.g., ARP,

TCP, HTTP), source and destination IP addresses, source and destination ports, timestamps, the length of the packet, and any warnings that might indicate issues like potential threats or errors in the packet data. This step verifies the application's ability to present an organized and precise summary of network traffic, which is critical for initial analyses and quick assessments by the user.

Interactive Packet Selection



Secondly, the action to test involves clicking on an individual packet entry in the summary table. The expected result of this interaction is that the application should open a new pane or window containing detailed information about the selected packet. This detail should include, but not be limited to, the hexadecimal representation of the packet data, and any human-readable content that might be extracted, such as HTTP responses. This functionality is crucial for users who need to perform deep dives into specific packets to investigate anomalies, study network behavior, or confirm suspected security incidents.

These test steps are designed to ensure that the application not only provides a high-level overview of the network traffic but also supports detailed investigations when users need more comprehensive data about particular packets. The success of this test will be determined by the application's responsiveness and its ability to accurately display detailed packet data, enhancing the user's ability to perform effective network analysis and troubleshooting.

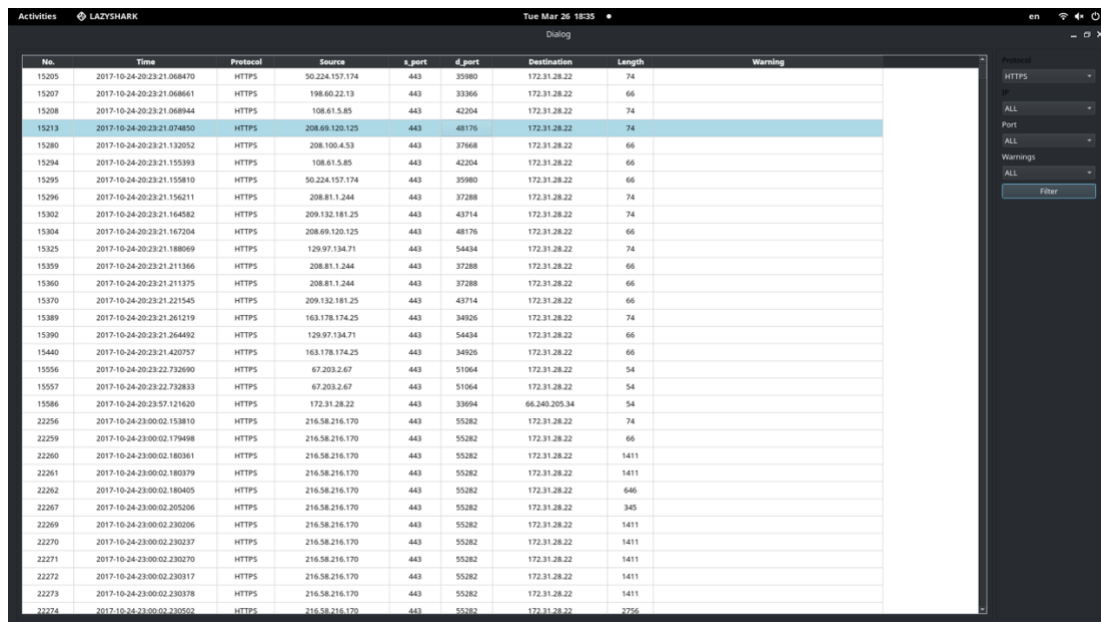
3) .pcap File Filtering and Analysis

The objective of this test case is to confirm the application's capability to read .pcap files and efficiently display packet data in a table that can be filtered using dropdown menus. This

functionality allows users to streamline their analysis by focusing on specific types of data such as protocol, IP, port, and warnings.

The first action in this test involves reviewing the table that displays packet data to ensure the presence and accuracy of various columns. The expected result is that the table correctly populates and displays all relevant columns, which typically include packet number, timestamp, protocol type, source and destination IP addresses, ports, packet length, and any associated warnings. This step is crucial for verifying that the application provides a comprehensive overview of packet data, enabling effective initial analyses.

Filtering Functionality Test – Protocol(HTTPS)



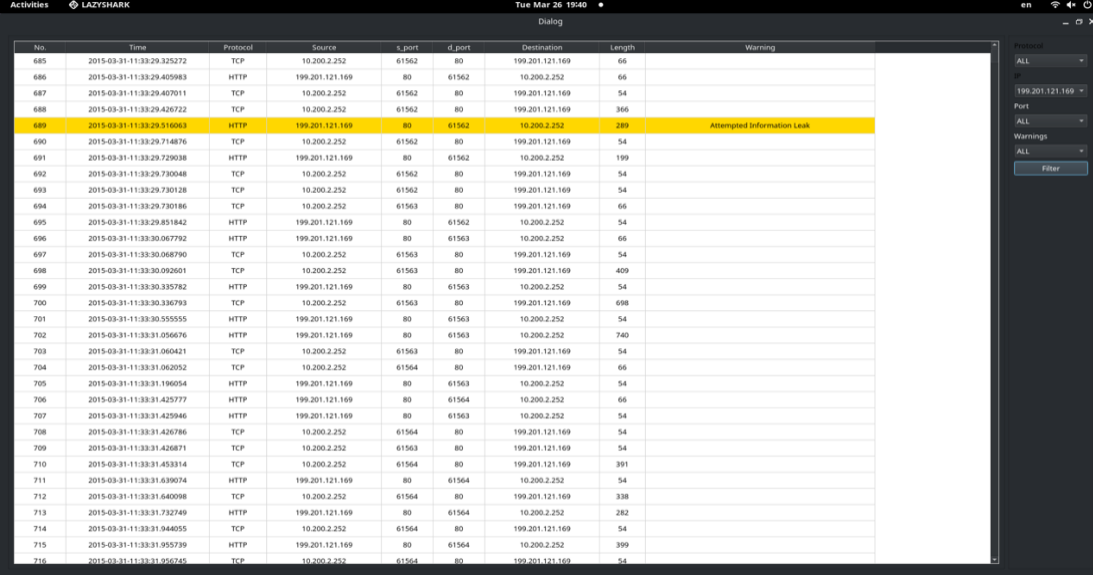
No.	Time	Protocol	Source	s.port	d.port	Destination	Length	Warning
15205	2017-10-24-20:23:21.068470	HTTPS	50.224.157.174	443	35980	172.31.28.22	74	
15207	2017-10-24-20:23:21.068661	HTTPS	198.60.22.13	443	33366	172.31.28.22	66	
15208	2017-10-24-20:23:21.068944	HTTPS	108.61.5.85	443	42204	172.31.28.22	74	
15213	2017-10-24-20:23:21.074850	HTTPS	208.69.120.125	443	48176	172.31.28.22	74	
15280	2017-10-24-20:23:21.132052	HTTPS	208.100.4.53	443	37568	172.31.28.22	66	
15294	2017-10-24-20:23:21.155393	HTTPS	108.61.5.85	443	42204	172.31.28.22	66	
15295	2017-10-24-20:23:21.155810	HTTPS	50.224.157.174	443	35980	172.31.28.22	66	
15296	2017-10-24-20:23:21.156211	HTTPS	208.81.1.244	443	37288	172.31.28.22	74	
15302	2017-10-24-20:23:21.164582	HTTPS	209.132.181.25	443	43714	172.31.28.22	74	
15304	2017-10-24-20:23:21.167204	HTTPS	208.69.120.125	443	48176	172.31.28.22	66	
15325	2017-10-24-20:23:21.188069	HTTPS	129.97.134.71	443	54434	172.31.28.22	74	
15359	2017-10-24-20:23:21.211366	HTTPS	208.81.1.244	443	37288	172.31.28.22	66	
15360	2017-10-24-20:23:21.211375	HTTPS	208.81.1.244	443	37288	172.31.28.22	66	
15370	2017-10-24-20:23:21.221545	HTTPS	209.132.181.25	443	43714	172.31.28.22	66	
15389	2017-10-24-20:23:21.261219	HTTPS	163.178.174.25	443	34926	172.31.28.22	74	
15390	2017-10-24-20:23:21.264402	HTTPS	129.97.134.71	443	54434	172.31.28.22	66	
15440	2017-10-24-20:23:21.420757	HTTPS	163.178.174.25	443	34926	172.31.28.22	66	
15556	2017-10-24-20:23:22.732690	HTTPS	67.203.2.67	443	51064	172.31.28.22	54	
15557	2017-10-24-20:23:22.732833	HTTPS	67.203.2.67	443	51064	172.31.28.22	54	
15586	2017-10-24-20:23:57.121620	HTTPS	172.31.28.22	443	33694	66.240.205.34	54	
22256	2017-10-24-23:00:02.153810	HTTPS	216.58.216.170	443	55282	172.31.28.22	74	
22259	2017-10-24-23:00:02.179498	HTTPS	216.58.216.170	443	55282	172.31.28.22	66	
22260	2017-10-24-23:00:02.180361	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22261	2017-10-24-23:00:02.180379	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22262	2017-10-24-23:00:02.180405	HTTPS	216.58.216.170	443	55282	172.31.28.22	646	
22267	2017-10-24-23:00:02.205206	HTTPS	216.58.216.170	443	55282	172.31.28.22	345	
22269	2017-10-24-23:00:02.230206	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22270	2017-10-24-23:00:02.230237	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22271	2017-10-24-23:00:02.230270	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22272	2017-10-24-23:00:02.230317	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22273	2017-10-24-23:00:02.230378	HTTPS	216.58.216.170	443	55282	172.31.28.22	1411	
22274	2017-10-24-23:00:02.230502	HTTPS	216.58.216.170	443	55282	172.31.28.22	7256	

The second part of the test focuses on the filtering functionality, specifically for protocols. The action required here is to use a dropdown menu to select a specific protocol, such as HTTPS, and apply the filter. The expected outcome is that the table should update to only display packets that match the selected protocol. This filtering capability is vital for users who need to isolate specific types of network traffic for more detailed analysis or troubleshooting purposes.

Overall, this test case assesses the application's ability to not only present detailed and accurate packet data but also to enhance the user experience by providing robust filtering tools. Successful completion of this test would confirm the application's functionality in handling and analyzing .pcap files efficiently, making it a valuable tool for network analysts and security professionals.

Continuing with the test case for verifying the filtering functionality of the application, we assess the ability to isolate packet data based on specific criteria such as IP addresses, ports, and warnings.

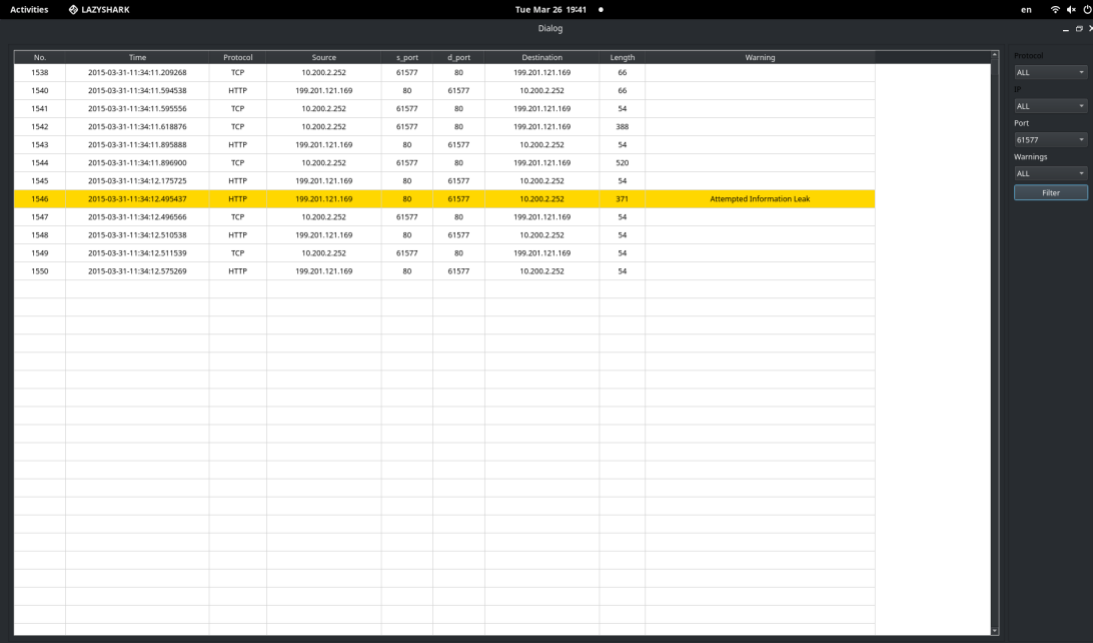
Filtering Functionality Test – IP:



No.	Time	Protocol	Source	s.port	d.port	Destination	Length	Warning
685	2015-03-31-11:33:29.325272	TCP	10.200.2.252	61562	80	199.201.121.169	66	
686	2015-03-31-11:33:29.405983	HTTP	199.201.121.169	80	61562	10.200.2.252	66	
687	2015-03-31-11:33:29.407011	TCP	10.200.2.252	61562	80	199.201.121.169	54	
688	2015-03-31-11:33:29.426722	TCP	10.200.2.252	61562	80	199.201.121.169	366	
689	2015-03-31-11:33:29.516063	HTTP	199.201.121.169	80	61562	10.200.2.252	289	Attempted Information Leak
690	2015-03-31-11:33:29.714876	TCP	10.200.2.252	61562	80	199.201.121.169	54	
691	2015-03-31-11:33:29.729038	HTTP	199.201.121.169	80	61562	10.200.2.252	199	
692	2015-03-31-11:33:29.730048	TCP	10.200.2.252	61562	80	199.201.121.169	54	
693	2015-03-31-11:33:29.730128	TCP	10.200.2.252	61562	80	199.201.121.169	54	
694	2015-03-31-11:33:29.730186	TCP	10.200.2.252	61563	80	199.201.121.169	66	
695	2015-03-31-11:33:29.851842	HTTP	199.201.121.169	80	61562	10.200.2.252	54	
696	2015-03-31-11:33:30.067792	HTTP	199.201.121.169	80	61563	10.200.2.252	66	
697	2015-03-31-11:33:30.068790	TCP	10.200.2.252	61563	80	199.201.121.169	54	
698	2015-03-31-11:33:30.092601	TCP	10.200.2.252	61563	80	199.201.121.169	409	
699	2015-03-31-11:33:30.335782	HTTP	199.201.121.169	80	61563	10.200.2.252	54	
700	2015-03-31-11:33:30.336793	TCP	10.200.2.252	61563	80	199.201.121.169	698	
701	2015-03-31-11:33:30.555555	HTTP	199.201.121.169	80	61563	10.200.2.252	54	
702	2015-03-31-11:33:31.056676	HTTP	199.201.121.169	80	61563	10.200.2.252	740	
703	2015-03-31-11:33:31.060421	TCP	10.200.2.252	61563	80	199.201.121.169	54	
704	2015-03-31-11:33:31.062052	TCP	10.200.2.252	61564	80	199.201.121.169	66	
705	2015-03-31-11:33:31.196054	HTTP	199.201.121.169	80	61563	10.200.2.252	54	
706	2015-03-31-11:33:31.425777	HTTP	199.201.121.169	80	61564	10.200.2.252	66	
707	2015-03-31-11:33:31.425946	HTTP	199.201.121.169	80	61563	10.200.2.252	54	
708	2015-03-31-11:33:31.426786	TCP	10.200.2.252	61564	80	199.201.121.169	54	
709	2015-03-31-11:33:31.426871	TCP	10.200.2.252	61563	80	199.201.121.169	54	
710	2015-03-31-11:33:31.453314	TCP	10.200.2.252	61564	80	199.201.121.169	391	
711	2015-03-31-11:33:31.639074	HTTP	199.201.121.169	80	61564	10.200.2.252	54	
712	2015-03-31-11:33:31.640098	TCP	10.200.2.252	61564	80	199.201.121.169	338	
713	2015-03-31-11:33:31.732749	HTTP	199.201.121.169	80	61564	10.200.2.252	282	
714	2015-03-31-11:33:31.944055	TCP	10.200.2.252	61564	80	199.201.121.169	54	
715	2015-03-31-11:33:31.955739	HTTP	199.201.121.169	80	61564	10.200.2.252	399	
716	2015-03-31-11:33:31.956745	TCP	10.200.2.252	61564	80	199.201.121.169	54	

The action involves using a dropdown menu to select a specific IP address and applying this filter. The expected result is that the table should update to display only the packets that match the selected IP address. This step is crucial for users who need to focus on traffic from or to a particular IP address, which can be essential for identifying localized network issues or tracking specific devices.

Filtering Functionality Test – Port



No.	Time	Protocol	Source	s.port	d.port	Destination	Length	Warning
1538	2015-03-31-11:34:11.209268	TCP	10.200.2.252	61577	80	199.201.121.169	66	
1540	2015-03-31-11:34:11.594538	HTTP	199.201.121.169	80	61577	10.200.2.252	66	
1541	2015-03-31-11:34:11.595556	TCP	10.200.2.252	61577	80	199.201.121.169	54	
1542	2015-03-31-11:34:11.618876	TCP	10.200.2.252	61577	80	199.201.121.169	388	
1543	2015-03-31-11:34:11.895888	HTTP	199.201.121.169	80	61577	10.200.2.252	54	
1544	2015-03-31-11:34:11.896900	TCP	10.200.2.252	61577	80	199.201.121.169	520	
1545	2015-03-31-11:34:12.175725	HTTP	199.201.121.169	80	61577	10.200.2.252	54	
1546	2015-03-31-11:34:12.405437	HTTP	199.201.121.169	80	61577	10.200.2.252	371	Attempted Information Leak
1547	2015-03-31-11:34:12.406566	TCP	10.200.2.252	61577	80	199.201.121.169	54	
1548	2015-03-31-11:34:12.510538	HTTP	199.201.121.169	80	61577	10.200.2.252	54	
1549	2015-03-31-11:34:12.511539	TCP	10.200.2.252	61577	80	199.201.121.169	54	
1550	2015-03-31-11:34:12.575269	HTTP	199.201.121.169	80	61577	10.200.2.252	54	

In this part of the test, the user will select a specific port from the dropdown menu and apply the filter. The expected result is that the table should refresh to show only packets that use the selected port. This functionality is particularly useful for pinpointing traffic associated with certain services or applications, allowing for detailed analysis of traffic flow and potential security vulnerabilities related to specific ports.

Filtering Functionality Test – Warnings



The screenshot shows the LAZYSHARK application window. At the top, it displays 'Activities', 'LAZYSHARK', and the date 'Tue Mar 26 19:41'. Below this is a 'Dialog' box. The main area contains a table of network packets. The table has columns: No., Time, Protocol, Source, S. port, D. port, Destination, Length, and Warning. The 'Warning' column contains values like 'Attempted Information Leak' and 'Mix activity'. On the right side, there is a 'Filter' panel with dropdown menus for 'Protocol', 'Port', and 'Warnings', and a 'Filter' button.

No.	Time	Protocol	Source	S. port	D. port	Destination	Length	Warning
689	2015-03-31-11:33:29.516063	HTTP	199.201.121.169	80	61562	10.200.2.252	289	Attempted Information Leak
1546	2015-03-31-11:34:12.405437	HTTP	199.201.121.169	80	61577	10.200.2.252	371	Attempted Information Leak
1585	2015-03-31-11:34:26.158666	ICMP	46.166.166.51	32048	11866	10.200.2.252	79	Mix activity
1570	2015-03-31-11:34:35.828012	HTTP	45.55.154.235	80	61580	10.200.2.252	289	Attempted Information Leak
2348	2015-03-31-11:42:38.576626	HTTP	199.201.121.169	80	61588	10.200.2.252	289	Attempted Information Leak
2745	2015-03-31-11:42:45.976768	HTTP	199.201.121.169	80	61584	10.200.2.252	773	Attempted Information Leak
2761	2015-03-31-11:43:08.293515	HTTP	5.135.28.104	80	61587	10.200.2.252	289	Attempted Information Leak
3125	2015-03-31-11:43:15.854170	HTTP	5.135.28.104	80	61602	10.200.2.252	376	Attempted Information Leak
3142	2015-03-31-11:43:38.195399	HTTP	107.191.46.222	80	61605	10.200.2.252	289	Attempted Information Leak
3155	2015-03-31-11:43:39.409868	HTTP	107.191.46.222	80	61606	10.200.2.252	335	Attempted Information Leak
3163	2015-03-31-11:43:40.923484	HTTP	188.226.129.49	80	61607	10.200.2.252	289	Attempted Information Leak

The action here involves selecting a warning type (e.g., "Attempted Information Leak") from a dropdown menu and applying this filter. The expected result is that the table will update to show only the packets that carry the selected warning type. This functionality helps in quickly identifying and focusing on packets that may indicate security threats or network misconfigurations, thereby aiding in proactive threat management and resolution.

This test should be conducted on a stable release of the application within an environment equipped with the necessary hardware and software configurations required for comprehensive network analysis. Ensuring that the application runs in a controlled and capable setting is vital for accurately assessing its performance and functionality.

Success in this test case will be defined by the application's ability to accurately and efficiently filter and display packet data based on the user's selections from the dropdown menus. The ability to isolate and analyze specific subsets of packet data enhances the application's utility for detailed network analysis and troubleshooting, providing valuable insights into network behavior and security posture.

4) .pcap File Warning Prioritization and Packet Data Inspection

The objective of this test case is to validate the program's capability to effectively prioritize and filter packet data based on warnings and to ensure that users can inspect detailed packet data associated with specific warnings by clicking on them.

Filtering Test for Warnings



No.	Time	Protocol	Source	s.port	d.port	Destination	Length	Warning
7567	2017-10-24-20:15:07.855843	DNS	8.8.8.8	53	55059	172.31.28.22	95	Potentially Bad Traffic
8138	2017-10-24-20:23:07.450417	DNS	8.8.8.8	53	57400	172.31.28.22	145	Potentially Bad Traffic
13152	2017-10-24-20:23:20.982671	DNS	8.8.8.8	53	42844	172.31.28.22	107	Potentially Bad Traffic
13157	2017-10-24-20:23:30.983404	DNS	8.8.8.8	53	46257	172.31.28.22	97	Potentially Bad Traffic
15088	2017-10-24-20:23:31.145879	DNS	8.8.8.8	53	46335	172.31.28.22	93	Potentially Bad Traffic
15041	2017-10-24-20:25:50.204897	ICMP	172.31.28.22	11822	11822	145.239.140.92	484	Misc activity
15086	2017-10-24-20:34:17.555798	DNS	8.8.8.8	53	47526	172.31.28.22	85	Potentially Bad Traffic
16178	2017-10-24-20:42:59.010230	ICMP	172.31.28.22	11822	11878	155.94.89.66	475	Misc activity
16227	2017-10-24-20:43:48.715806	ICMP	172.31.28.22	11822	11822	8.8.4.4	115	Misc activity
16584	2017-10-24-20:53:29.040225	ICMP	172.31.28.22	11822	13145	51.15.87.28	480	Misc activity
17057	2017-10-24-21:06:36.341061	ICMP	172.31.28.22	11822	11810	46.166.142.60	477	Misc activity
17681	2017-10-24-21:29:21.020215	ICMP	172.31.28.22	31278	32086	125.212.217.215	93	Misc activity
18019	2017-10-24-21:37:45.839594	ICMP	172.31.28.22	11822	20526	80.82.70.222	474	Misc activity
18113	2017-10-24-21:39:23.252170	ICMP	172.31.28.22	11822	11822	46.101.108.19	99	Misc activity
18199	2017-10-24-21:41:16.062141	ICMP	172.31.28.22	11822	11877	155.94.89.66	476	Misc activity
18214	2017-10-24-21:44:07.120364	TCP	61.188.189.5	36664	32	172.31.28.22	296	Executable code was detected
18268	2017-10-24-21:44:10.660584	TCP	61.188.189.5	41057	32	172.31.28.22	296	Executable code was detected
18284	2017-10-24-21:44:14.801593	TCP	61.188.189.5	44043	32	172.31.28.22	296	Executable code was detected
18301	2017-10-24-21:44:18.780694	TCP	61.188.189.5	47174	32	172.31.28.22	296	Executable code was detected
18354	2017-10-24-21:44:23.090336	TCP	61.188.189.5	50575	32	172.31.28.22	296	Executable code was detected
18381	2017-10-24-21:44:27.037081	TCP	61.188.189.5	53957	32	172.31.28.22	296	Executable code was detected
18442	2017-10-24-21:44:30.919449	TCP	61.188.189.5	56724	32	172.31.28.22	296	Executable code was detected
18474	2017-10-24-21:44:34.381622	TCP	61.188.189.5	58640	32	172.31.28.22	296	Executable code was detected
18508	2017-10-24-21:44:38.457553	TCP	61.188.189.5	61886	32	172.31.28.22	296	Executable code was detected
18543	2017-10-24-21:44:42.182093	TCP	61.188.189.5	64678	32	172.31.28.22	296	Executable code was detected
18577	2017-10-24-21:44:46.050881	TCP	61.188.189.5	67414	32	172.31.28.22	296	Executable code was detected
18609	2017-10-24-21:44:50.251980	TCP	61.188.189.5	70171	32	172.31.28.22	296	Executable code was detected
18670	2017-10-24-21:47:12.296174	TCP	62.112.10.102	33814	32	172.31.28.22	218	Executable code was detected
18688	2017-10-24-21:47:16.236869	TCP	62.112.10.102	47015	32	172.31.28.22	218	Executable code was detected
18709	2017-10-24-21:47:19.756228	TCP	62.112.10.102	76089	32	172.31.28.22	218	Executable code was detected
18754	2017-10-24-21:47:22.674755	TCP	62.112.10.102	48944	32	172.31.28.22	218	Executable code was detected
18781	2017-10-24-21:47:26.006204	TCP	62.112.10.102	52544	32	172.31.28.22	218	Executable code was detected

The action required for this test involves using a dropdown menu to select a filter specifically for warnings and applying this filter to the packet data displayed. The expected result is that the table will update to display only the packets that contain warnings. Furthermore, these packets should be highlighted according to their level of urgency or priority, making it easier for users to identify and focus on the most critical issues first. This functionality is essential for efficient network monitoring and threat management, as it allows users to quickly and easily prioritize their responses based on the severity of the warnings.

This test is crucial for verifying that the application not only filters and displays warnings effectively but also categorizes and prioritizes them in a way that enhances user response and decision-making processes. The ability to click on a warning to view detailed packet data is an additional functionality that supports in-depth analysis, enabling users to understand the context and specifics of each warning, which is vital for troubleshooting and resolving network issues.

Success in this test will be judged by the application's precision in filtering and displaying only the packets with warnings and its effectiveness in organizing these warnings by priority, alongside the ease of accessing detailed data for each highlighted warning.

Inspect Packet Data with Warning

The screenshot displays a network analysis tool interface. The top section shows a list of packets with columns for No., Time, Protocol, Source, s_port, d_port, Destination, Length, and Warning. The list includes several entries with warnings such as "Indicator-Compromise 403 Forbidden" and "Protocol-ICMP Destination Unreachable Host Unreachable".

The bottom section shows a detailed view of a selected packet (No. 3163). The packet data is displayed in hexadecimal and ASCII format. The ASCII representation shows an HTTP 403 Forbidden response from a Microsoft Internet Information Services (IIS) server. The response includes headers like "Server: Microsoft-IIS/8.5", "Date: Tue, 31 Mar 2015 11:42:44 GMT", and "Content-Type: text/html". The body of the response contains an error message in Korean: "403 Forbidden. 세션이 만료되었습니다. 다시 로그인하십시오." (Session has expired. Please log in again).

Building on the earlier steps of testing, this phase focuses on verifying the interaction with a highlighted packet entry that contains a warning. The action involves clicking on this packet entry, with the expected outcome being that the program displays a detailed view of the packet's data. This view should include both hexadecimal and ASCII representations of the packet content, providing a comprehensive examination of the data. Additionally, it should highlight specific details that triggered the warning to allow for in-depth analysis, crucial for diagnosing issues and understanding the nature of warnings, vital for effective network management and security analysis.

The test should be conducted on the latest stable version of the application, ensuring that all necessary preconditions for handling .pcap files are met. This includes having a suitable computing environment with the necessary hardware specifications and software configurations to support the application's network analysis capabilities. The successful execution of this test hinges on several factors.

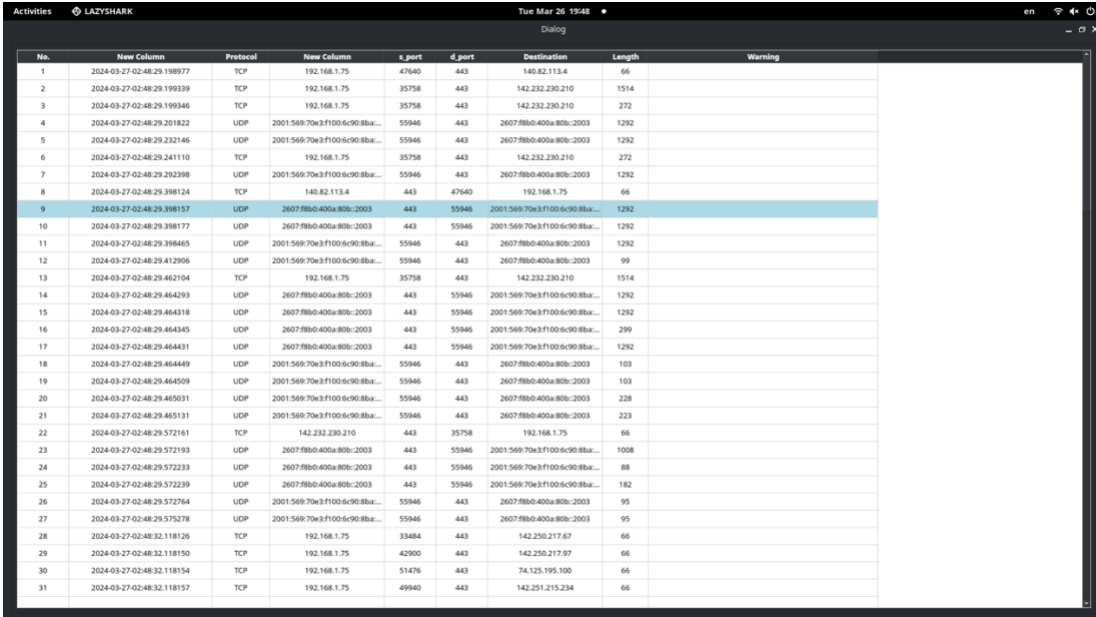
First, the application must accurately filter packet data based on warnings and correctly highlight these packets according to their priority levels. This initial filtering and prioritization are fundamental to the test's success. Secondly, the ability to click on a packet and view its detailed content without experiencing errors or malfunctions is essential. The detailed view must include all relevant data representations and highlight the reasons for the warning,

facilitating a thorough understanding of each packet's issues. This functionality is critical for enhancing the usability and effectiveness of the application in real-world network monitoring and security scenarios, and a successful test would confirm the application's reliability for network analysts and security professionals.

5) Live Packet Capture Functionality

The objective of this test phase is to evaluate the program's live packet capturing feature, specifically when the 'little shark button' is activated. The test aims to verify the functionality of dynamically capturing and displaying packet data, as well as the ability to view detailed packet information upon selecting a packet from the live capture table.

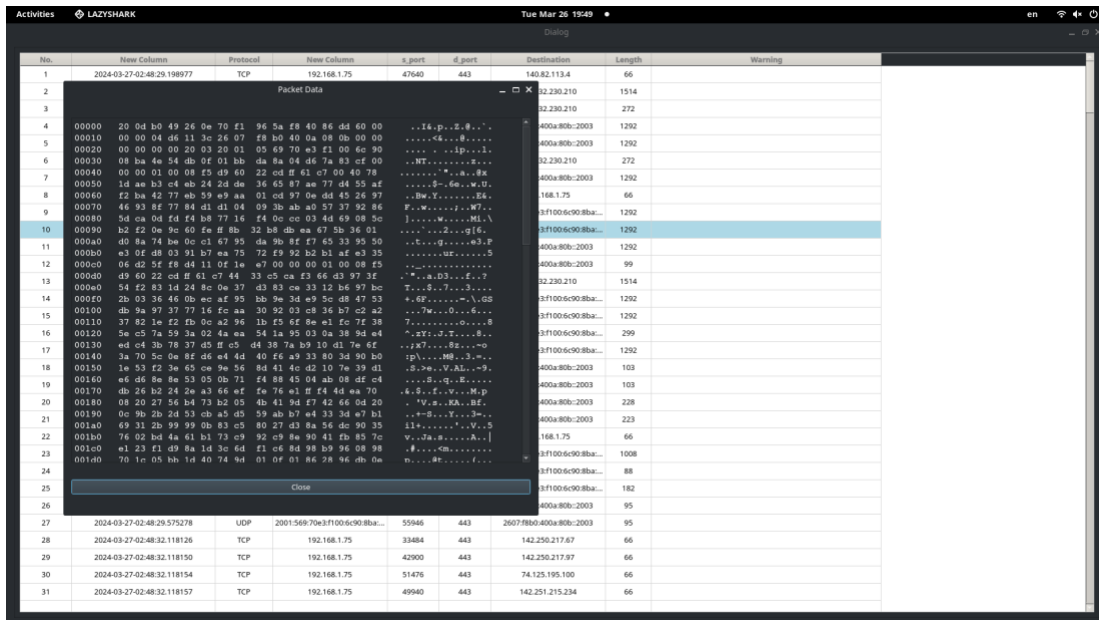
Verify Live Packet Stream



No.	New Column	Protocol	New Column	s.port	d.port	Destination	Length	Warning
1	2024-03-27 02:48:29.198977	TCP	192.168.1.75	47640	443	142.232.230.210	66	
2	2024-03-27 02:48:29.199339	TCP	192.168.1.75	35758	443	142.232.230.210	1514	
3	2024-03-27 02:48:29.199346	TCP	192.168.1.75	35758	443	142.232.230.210	272	
4	2024-03-27 02:48:29.201822	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	1292	
5	2024-03-27 02:48:29.202146	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	1292	
6	2024-03-27 02:48:29.241110	TCP	192.168.1.75	35758	443	142.232.230.210	272	
7	2024-03-27 02:48:29.292398	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	1292	
8	2024-03-27 02:48:29.398124	TCP	140.82.113.4	443	47640	192.168.1.75	66	
9	2024-03-27 02:48:29.398157	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1292	
10	2024-03-27 02:48:29.398177	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1292	
11	2024-03-27 02:48:29.398465	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	1292	
12	2024-03-27 02:48:29.412906	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	99	
13	2024-03-27 02:48:29.462104	TCP	192.168.1.75	35758	443	142.232.230.210	1514	
14	2024-03-27 02:48:29.464293	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1292	
15	2024-03-27 02:48:29.464318	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1292	
16	2024-03-27 02:48:29.464345	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	299	
17	2024-03-27 02:48:29.464431	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1292	
18	2024-03-27 02:48:29.464449	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	103	
19	2024-03-27 02:48:29.464509	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	103	
20	2024-03-27 02:48:29.485031	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	228	
21	2024-03-27 02:48:29.485131	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	223	
22	2024-03-27 02:48:29.572161	TCP	142.232.230.210	443	35758	192.168.1.75	66	
23	2024-03-27 02:48:29.572193	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	1008	
24	2024-03-27 02:48:29.572233	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	88	
25	2024-03-27 02:48:29.572239	UDP	2607:fb80:400a:80b::2003	443	55946	2001:569:70a3:f100:6c90:8ba...	182	
26	2024-03-27 02:48:29.572764	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	95	
27	2024-03-27 02:48:29.575278	UDP	2001:569:70a3:f100:6c90:8ba...	55946	443	2607:fb80:400a:80b::2003	95	
28	2024-03-27 02:48:32.118126	TCP	192.168.1.75	33484	443	142.250.217.67	66	
29	2024-03-27 02:48:32.118150	TCP	192.168.1.75	42900	443	142.250.217.97	66	
30	2024-03-27 02:48:32.118154	TCP	192.168.1.75	51476	443	74.125.195.100	66	
31	2024-03-27 02:48:32.118157	TCP	192.168.1.75	49940	443	142.251.215.234	66	

The first action in this test is to observe the table where live packet data should be dynamically populated as they are captured. The expected result is that the packet data will stream into the table, updating in real-time without any interruptions or errors. This real-time updating is crucial for the effectiveness of live packet capture, as it ensures that the user can see ongoing network activity as it happens.

Inspect Live Captured Packet Data



Following the verification of the live packet stream, the next action involves clicking on a packet entry from the live capture table. The expected result of this action is that a detailed view of the selected packet's data should appear, displaying both the hexadecimal and ASCII contents. This allows the user to inspect the packet's detailed structure and content thoroughly, which is essential for detailed network analysis and troubleshooting in real time.

This test must be performed in a live network environment capable of generating sufficient packet traffic to effectively test the live capture feature. This environment should simulate typical or expected network conditions to ensure the functionality can handle real-world applications.

To pass this test, the live packet capture feature must successfully capture and display packet data in real-time. Additionally, the program should allow the user to click on a packet entry and view detailed information accurately, reflecting the current packet's data. Successful completion of these criteria will confirm that the live packet capture functionality operates reliably and meets the necessary standards for real-time network monitoring and analysis.

2.7. Implications of Implementation

1) **MainWindow**

The MainWindow class in the code snippet acts as the primary interface for the user. It initializes the user interface elements and handles user interactions. The class is set up to support drag-and-drop functionality, allowing users to add .pcap files by dragging them into the window. Upon dropping files, the application verifies the file type and populates a vector with the file paths if they match the expected .pcap extension. Additionally, the MainWindow contains slots connected to buttons like "Submit" and "Run". When the "Submit" button is clicked, it triggers processing of the .pcap files listed in the vector, involving reading the packet data and possibly interfacing with the SnortRunner for further analysis. The "Run" button initiates live packet capturing, which opens up the LiveCapture window and begins capturing packets in a new thread to maintain application responsiveness.

2) **PcapReader**

The PcapReader class plays a crucial role in the application's backend, specifically designed for the ingestion and analysis of .pcap files. Upon invoking its open method, PcapReader initializes the file reading process. If successful, it transitions to reading the .pcap file, meticulously extracting packet details while correlating them with a log map—presumably for cross-referencing with pre-existing threat intelligence or packet metadata. As packets are read, PcapReader categorizes and processes different protocols: Ethernet, ARP, IPv4, IPv6, and the transport layer protocols TCP, UDP, and ICMP, among others. This robust processing ability showcases the application's preparedness to handle a wide range of network traffic scenarios, making it a versatile tool for network analysis. Furthermore, the class also contains functionality to format timestamps from packet headers, ensuring the timestamps are in a uniform and readable format, crucial for accurate logging and event correlation. Lastly, PcapReader is designed with error handling in mind, clearly logging issues when file reading fails and when unexpected protocol types are encountered. This proactive approach in error management contributes to a more resilient and user-friendly application.

3) **SnortRunner**

The SnortRunner class is a specialized component of the application designed to interact with Snort, a widely used network intrusion detection system (IDS). The class is tasked with running Snort against .pcap files and parsing the generated logs to extract and store relevant security information. The generateSnortLog function constructs a command line string to execute Snort with specified parameters, including the configuration file, ruleset, and the .pcap file to be analyzed. The command directs Snort's output to a log file named after the .pcap file, facilitating organized storage and retrieval of intrusion detection data. The processLog function then reads this generated log file, using regular expressions to parse significant details such as the timestamp, classification of the alert, and its priority. This data is stored in an unordered_map, linking the time of each alert to its corresponding priority and classification, thus providing a quick reference to the severity and type of network events detected by Snort.

4) PacketCatcher

The PacketCatcher class is designed to capture network packets in real-time, which is crucial for live network monitoring and analysis. Upon execution, it initiates a live packet capture using the `pcap_open_live` function, which sets up the network interface for capturing packets. Once the interface is ready, it starts a capture loop with `pcap_loop`, specifying `packetHandler` as the callback function to be called for each captured packet. The `packetHandler` function is where the bulk of packet processing occurs. For each packet received, it constructs a `CustomPacket` object, which likely serves as a container for structured packet information. This function also handles different network protocols by inspecting the Ethernet type and branching accordingly to process ARP, IPv4, IPv6, and the inner protocols such as TCP, UDP, and ICMP. Time formatting functions, `formatTime` and `removeYear`, are used to standardize the timestamp format for logging or display purposes, an important aspect for accurate timing information in network analysis. Finally, the `notifyPacketCaptured` method signals that a new packet has been captured and processed, which is likely used to update the user interface or trigger further analysis in real-time.

5) AnalyzeWindow

The `AnalyzeWindow` class serves as a specialized dialog within the application, responsible for displaying and interacting with packet data that has been read and processed. The class is constructed with Qt's user interface elements, which are set up and customized in its constructor. It initiates a priority mapping to color-code packet data based on their priority, enhancing data visibility and user understanding. Upon setting up the UI, the `fillTable` method populates the table with packet information, aligning the content and setting the background color to reflect the associated priority. It also sets up dropdown filters for protocols, IPs, and ports, which facilitates the user's ability to filter the displayed packets according to these criteria. Interactivity is enabled through signals and slots, where clicking a table item invokes the `onItemClicked` method. This interaction is likely to prompt the display of more detailed packet information, possibly including a hexadecimal dump of the packet's contents. Destruction of the UI elements is cleanly handled by the destructor to ensure that no resources are leaked. Overall, `AnalyzeWindow` functions as an informative and interactive panel where users can sift through packet data, a cornerstone feature for any network analysis tool.

2.8. Innovation

The innovations introduced in this program aim to significantly enhance the user experience and functionality of network packet analysis tools, setting it apart from traditional packet analyzers like Wireshark. One of the standout features is the enhanced threat visualization, which not only reads .pcap files but also incorporates the capability to display warnings and risk assessments based on Snort 3 rules, allowing users to quickly identify and respond to potential security threats with insights that go beyond basic packet data. Additionally, unlike Wireshark, which relies on complex command-line syntax for packet classification, this program offers a user-friendly drag-and-drop interface that eliminates the need for

memorizing and inputting intricate commands, thus making network security analysis accessible to a broader audience, including those with limited technical expertise. The program also features a simplified user interface designed to be intuitive and streamlined, significantly reducing the learning curve and making it less daunting for beginners compared to many other packet analyzers. A core innovation is the program's commitment to making network security analysis accessible to non-experts by removing technical barriers and providing visual insights, thereby empowering a wider user base to engage in network monitoring and threat detection. Looking to the future, there are plans to potentially include a real-time threat monitoring tool, which would continuously assess incoming packets and provide immediate warnings for suspicious activity, offering a proactive approach to threat detection that surpasses the capabilities of most traditional packet analyzers, including Wireshark.

2.9. Complexity

The integration of multiple sophisticated tools like C++, Qt, CMake, Wireshark, and Snort is an intricate endeavor. Each of these tools, in its own right, is akin to a vast universe with unique functionalities and intricacies. Achieving a harmonious confluence where each tool's capabilities are synchronized and fully utilized is a significant hurdle. Additionally, it's not just about superficially deploying these components; there's an imperative to delve deep. Truly understanding the foundational principles, nuances, and potential synergies of each component is an endeavor filled with complexities. Further complicating matters is the project's need to manage and interpret a plethora of data types and protocols. Ranging from raw network packets to detailed IDS logs, each data set presents its own unique syntax, structure, and significance. Parsing and prioritizing this diverse array of information while maintaining the integrity and objective of the analysis adds another layer of challenge to the project.

The complexity and intricacies of the proposed project render it more appropriate for a BTech level rather than a diploma level. One fundamental reason is the limited exposure diploma students have to advanced programming languages like C++. While they might be introduced to the rudiments, the advanced functionalities crucial for this project are typically reserved for BTech curricula. Furthermore, specialized tools such as Qt, CMake, and Snort, essential for this venture, might be entirely absent from diploma syllabi. The depth of understanding required for effective packet analysis, encompassing the knowledge of network protocols, packet structures, and the nuances of tools like Wireshark, is often beyond what a diploma offers.

The crux of the problem lies in achieving a harmonized operation of various tools. Ensuring that packets filtered by Wireshark are then meticulously assessed by Snort for potential threats is a challenge. Additionally, tools like Snort generate specialized log outputs, which demand another set of tools for effective interpretation. Integrating this array of outputs into a user-friendly, accessible interface is a daunting task. Another non-trivial challenge is ensuring cross-platform compatibility. With the multitude of operating systems, each with its distinct architecture and behaviors, ensuring a flawless operation across platforms becomes complex.

To tackle these challenges, students need a repertoire of specialty knowledge. This includes an advanced grasp of C++ programming, expertise in GUI development using the Qt framework, proficiency in build and dependency management with CMake, a deep understanding of network protocols, and expertise in intrusion detection using tools like Snort.

Even with these competencies, there are areas where students might find themselves at a crossroads. Delving deeper into advanced C++ features, understanding multi-threading for coordinated operations, and achieving proficiency in frameworks like Qt and CMake would demand additional exploration. Furthermore, students would need to dive deeper into intrusion detection systems, specifically understanding the mechanisms of tools like Snort. Learning the nuances of cross-platform development, adopting advanced optimization techniques, and ensuring effective communication between tools are areas that would require substantial research. Lastly, a rigorous testing and validation phase, entailing both unit and integration testing, would be paramount to ensure the system's robustness and reliability. This holistic approach to problem-solving, combining prior knowledge with research-driven exploration, encapsulates the essence of a BTech project.

2.10. Research in New Technologies

The design and development

User Interface (UI) Design: The UI is crafted using the Qt framework, renowned for its robust capabilities in user interface design. A notable innovation in this area is the implementation of drag-and-drop functionality, simplifying the user interaction and making the tool more accessible, especially for users with limited technical expertise. The technical challenge here is ensuring that the UI remains responsive and intuitive, despite the complexity of the backend operations.

File Handling and Processing: This component relies on File I/O in C++ for manipulating .pcap files. An innovative feature here is the real-time verification of file types, which streamlines the data processing pipeline and enhances efficiency. The primary technical challenge is managing large or corrupted files without compromising the performance or stability of the application.

Network Packet Analysis: The tool uses libpcap, a standard library for network packet capture and analysis. Innovation in this domain includes deep packet inspection capabilities that allow for detailed categorization and processing of various network protocols. The challenge lies in efficiently parsing and analyzing a wide variety of network protocols, ensuring accuracy and speed in data handling.

Integration with Network Intrusion Detection Systems (NIDS): Utilizing Snort for intrusion detection, this area focuses on seamless integration, allowing users to run and interpret Snort directly from the application. This integration represents a significant step forward in usability, although it poses the technical challenge of parsing and correlating Snort logs with packet data in a manner that remains clear and user-friendly.

Real-Time Data Acquisition: The technology employed here is pcap_live, which is instrumental in capturing live network traffic. The innovation includes the ability to perform concurrent packet capturing and processing, which is crucial for maintaining real-time performance. The main technical challenge is minimizing latency and ensuring accurate timestamping for the live data, which is vital for effective real-time analysis.

Potential Areas and Technologies for Future Development

Artificial Intelligence (AI) and Machine Learning (ML): Future enhancements may include implementing AI to predict and classify network anomalies. This approach would involve introducing AI-based algorithms for anomaly detection and threat prediction, with a focus on researching ML models that are suitable for real-time data processing and pattern recognition.

Advanced Data Visualization: There is also potential for developing more sophisticated visualization tools for network traffic. Innovations could include real-time dashboards with interactive elements that provide a comprehensive overview of network health. Research in this area would likely focus on best practices in data visualization to represent complex network data efficiently and effectively.

2.11. Future Enhancements

The development of the network analysis project has reached a pivotal stage with the successful creation and implementation of a Minimum Viable Product (MVP), which includes the foundational feature of live packet capturing. This initial phase has paved the way for identifying several potential enhancements and improvements that could significantly augment the project's capabilities and effectiveness in real-world applications.

While the MVP and the basic live packet capturing tool have been realized within the project's timeline, future enhancements such as integrating live packet capture with the Snort Intrusion Detection System (IDS) and improving concurrency handling are now seen as critical next steps. These enhancements, though optional, are highly valuable for boosting the tool's functionality in a live environment.

Looking ahead, the project is poised for several targeted improvements. A key development will be the seamless integration of live packet capturing with Snort IDS, enabling real-time analysis and alerting. This integration is crucial as it significantly enhances the tool's utility for immediate threat detection and response. Additionally, since live packet capturing and analysis are resource-intensive, there is a plan to optimize these processes for concurrency. This will improve the performance and responsiveness of the tool, making it more efficient in handling high volumes of network traffic.

Another area of focus will be the user interface, which is set to include more intuitive controls for live packet analysis such as buttons to start, stop, and pause capture sessions. These improvements aim to enhance user interaction and control during live monitoring sessions. Furthermore, the tool will feature advanced filtering and search functionalities that allow users to efficiently locate specific packets within a large stream of captured data, enhancing the tool's usability.

Automated alerting mechanisms that trigger based on specific detections or anomalies identified by Snort will also be integrated. These will provide instant notifications to users or systems, fostering a proactive approach to network security. Lastly, enhancing the logging and reporting capabilities to include more detailed analytics will provide deeper insights for forensic analysis and offer a thorough review of historical data.

These planned improvements are designed to not only expand the functionality of the network analysis tool but also to refine its efficiency and usability, thereby ensuring it meets the evolving needs of network security monitoring and threat management.

2.12. Timeline and Milestones

December: Setup and Learning Phase

Initial Project Setup and Requirement Gathering		
Description	Day	Hours
Install basic requiremetns	20-Dec	3
Install Snort3	21-Dec	3
Learn Qt6 using C++	22-Dec	3
	23-Dec	SAT
	24-Dec	SUN
Learn Qt6 using C++	25-Dec	4
Learn Qt6 using C++	26-Dec	5
Learn Qt6 using C++	27-Dec	5
Learn Qt6 using C++	28-Dec	3
Learn Qt6 using C++	29-Dec	3
	30-Dec	SAT
	31-Dec	SUN
Total Days/Hours	12 Days	29 Hours

The initial phase of the project focused on setting up the necessary tools and learning the key technology required for development. The main tasks included installing basic requirements, setting up Snort3, and learning Qt6 with C++.

- **Timeframe:**

Start Date: December 20, 2024

End Date: December 31, 2024

Total Duration: 12 Days


- **Allocated Hours:**

Total Hours Spent: 29 Hours

- **Activity Breakdown:**

- 1) Install Basic Requirements: Essential software and tools were installed, ensuring the development environment was correctly set up.
- 2) Install Snort3: Snort3 was successfully installed, which is crucial for the network intrusion detection aspect of the project.
- 3) Learn Qt6 using C++: Progressive learning of Qt6 using C++. I now have a foundational understanding of the framework to proceed with UI development for the project.

January: MainWindow & Snort Integration Phase

MainWindow & Snort Integration		
Description	Day	Hours
Design MainWindow.cpp	01-Jan	4
Implement a basic layout of MainWindow	02-Jan	5
Implement a basic layout of MainWindow	03-Jan	4
Add a background in MainWindow	04-Jan	4
Implement a function that change MainWindow size dynamically	05-Jan	5
	06-Jan	SAT
	07-Jan	SUN
Implement a layout of 'Drop' area in MainWindow	08-Jan	5
Implement a 'Drag' function	09-Jan	6
Test and Debug 'Drag' & 'Drop'	10-Jan	5
Implement a file read function for .pcap	11-Jan	4
Implement a 'Submit' button	12-Jan	4
	13-Jan	SAT
	14-Jan	SUN
Implement button slot and activate when clicked	15-Jan	6
Implement button slot and activate when clicked	16-Jan	5
Design 'Submit' button and add 'Capture' button	17-Jan	6
Test slot for each buttons and debug	18-Jan	4
Add an icon for 'Capture' button	19-Jan	5
	20-Jan	SAT
	21-Jan	SUN
Install Snort 3	22-Jan	6
Install Snort 3 cont, test, and confirm alert logs	23-Jan	3
Implement process that runs Snort 3	24-Jan	4
Implement process that runs Snort 3 cont	25-Jan	3
Create SnortRunner class and design as a Singleton	26-Jan	4
	27-Jan	SAT
	28-Jan	SUN
Connect SnortRunner with 'Submit' button	29-Jan	3
Test & Debug SnortRunner when pcap dropped	30-Jan	5
Test & Debug SnortRunner when pcap dropped	31-Jan	5
Total Days/Hours	 31 Days	105 Hours

The second phase of the project was dedicated to the design and development of the MainWindow component and its integration with Snort3. Key functionalities such as layout design, drag-and-drop features, file reading, and the execution of Snort3 were the primary focus.

- **Timeframe**

Start Date: January 1, 2025

End Date: January 31, 2025

Total Duration: 31 Days

- **Allocated Hours:**

Total Hours Spent: 105 Hours

- **Activity Breakdown:**

- 1) Design and Development of MainWindow.cpp: Established the primary window structure, allowing further design and feature implementation.
- 2) Implementation of Drag-and-Drop and Submit Features: Enabled the 'Drop' area for file input and designed the 'Drag' function, followed by the 'Submit' button setup, enhancing user interaction with the application.
- 3) Testing and Debugging: Conducted thorough testing and debugging for the drag-and-drop and button functionalities, ensuring reliable operation.
- 4) Snort3 Installation and Configuration: Snort3, confirmed alert logs, and implemented processes to run the intrusion detection system.
- 5) SnortRunner Class Implementation and Integration: Developed the SnortRunner class as a Singleton pattern for better resource management and linked it to the 'Submit' button to analyze .pcap files.

- **Challenges and Mitigations:**

- 1) Encountered difficulties with integrating Snort3 due to compatibility issues, resolved by thorough testing and configuration adjustments.
- 2) Debugging the drag-and-drop feature took additional time, mitigated by allocating extra hours and reviewing Qt documentation.

February (PcapReader & CustomPacket & AnalyzeWindow Development)

PcapReader & CustomPacket & AnalyzeWindow Implementation		
Description	Day	Hours
Read PcapReader additional implementation	01-Feb	5
Design CustomPacket Class	02-Feb	6
	03-Feb	SAT
	04-Feb	SUN
Implement reading .pcap data funtion	05-Feb	4
Create CustomPakcet Class	06-Feb	5
Connect PcapReader with 'Submit' button	07-Feb	4
CMakeList.txt update for Libpcap Library	08-Feb	3
Implement PcapReader call pcap_loop() and print data till the .pcap file ends	09-Feb	5
	10-Feb	SAT
	11-Feb	SUN
Implement processing ethernet header	12-Feb	7
Implement processing ipv4 header	13-Feb	5
Implemnet Processing ipv6 header	14-Feb	6
Implement processing udp header	15-Feb	6
Implement processing tcp header	16-Feb	7
	17-Feb	SAT
	18-Feb	SUN
Implement processing arp header	19-Feb	4
Implement a map for handling other protocols	20-Feb	5
Test each data save in CustomPacket variables	21-Feb	4
Format Timestamp for UTC both pcap data and log	22-Feb	4
Create priority map and match with CustomPacket	23-Feb	6
	24-Feb	SAT
	25-Feb	SUN
Design AnalyzeWindow	26-Feb	5
Create AnalyzeWindow and add packet info table	27-Feb	5
Link each CustomPacket item to table	28-Feb	3
Link each CustomPacket item to table cont	29-Feb	3
Total Days/Hours	29 Days	102 Hours

This phase was dedicated to the development and implementation of the PcapReader functionality, the creation of the CustomPacket class for structured packet handling, and the establishment of the AnalyzeWindow component for displaying packet information.

- **Timeframe:**

Start Date: February 1, 2025

End Date: February 29, 2025

Total Duration: 29 Days

- **Allocated Hours:**

Total Hours Spent: 102 Hours

- **Activity Breakdown:**

- 1) PcapReader Implementation: Enhanced PcapReader with additional features and linked it to the 'Submit' button for file reading actions.
- 2) CustomPacket Class Creation: Designed and implemented the CustomPacket class to standardize packet data handling and ensure consistent data structure across the application.
- 3) Header Processing Implementation: Developed processing capabilities for Ethernet, IPv4, IPv6, UDP, TCP, and ARP headers, as well as a map for handling other protocols.
- 4) Testing and Timestamp Formatting: Conducted testing for data storage in CustomPacket variables and implemented UTC timestamp formatting for both pcap data and logs.
- 5) AnalyzeWindow Development: Designed and created the AnalyzeWindow, linked packet information to the table display, and ensured proper connectivity between the CustomPacket items and the interface.

- **Challenges and Mitigations:**

- 1) Integrating multiple header processing required extensive research and precise implementation, which was achieved through incremental development and continuous testing.
- 2) Standardizing timestamp formats across different data sources required careful consideration of timezone conversions and formatting methods due to Snort3 time calculation is slightly different.

March (Sorting Packets and Enhancement Phase)

Sorting packets and additional implementation based on supervisor's feedback		
Description	Day	Hours
Color item in the table based on priority from log	01-Mar	4
	02-Mar	SAT
	03-Mar	SUN
Show packets on the table	04-Mar	6
Test & Debug uncovered protocols	05-Mar	5
Test & Debug uncovered protocols cont	06-Mar	5
Test & Debug with dropping multiple .pcap files	07-Mar	7
Test & Debug table item match with .pcap packets	08-Mar	8
	09-Mar	SAT
	10-Mar	SUN
Implement Set for sorting protocol, ip, port	11-Mar	7
Implement Set for sorting protocol, ip, port cont	12-Mar	7
Link Set to AnalyzeWindow	13-Mar	5
Test for sorting drop down menu ip	14-Mar	5
Test for sorting drop down menu port	15-Mar	5
Test for sorting drop down menu protocols	16-Mar	SAT(6)
	17-Mar	SUN
Debug minor errors for memory leaks	18-Mar	5
Write a final project report	19-Mar	9
Final testing before meeting supervisor and continue to write the report	20-Mar	10
Implement Set for sorting warnings (Feedback)	21-Mar	5
Implement Qdialog for showing packet data (Feedback)	22-Mar	8
Implement Live Capture mode (Feedback)	23-Mar	SAT(10)
Implement Live Capture mode (Feedback) cont	24-Mar	SUN(12)
Test Live Capture mode	25-Mar	8
Test for sorting drop down menu warnings	26-Mar	5
Continue to write the report	27-Mar	12
	28-Mar	
	29-Mar	
	30-Mar	SAT
	31-Mar	SUN
Total Days/Hours	31 Days	154 Hours

This phase centered on refining the application's functionality based on the supervisor's feedback. Tasks included implementing features for sorting packets, enhancing the user interface with color-coding based on priority, and developing additional components like the Live Capture mode.

- **Timeframe:**

Start Date: March 1, 2025

End Date: March 31, 2025

Total Duration: 31 Days

- **Allocated Hours:**

Total Hours Spent: 154 Hours

- **Activity Breakdown:**

- 1) User Interface Enhancement: Items in the table were color-coded based on log priorities, and improvements were made to show packets more clearly on the table.
- 2) Testing and Debugging Protocols: Completed extensive testing and debugging of uncovered protocols and multiple .pcap file drops.
- 3) Sorting Feature Implementation: Implemented and linked a set for sorting data based on protocol, IP, and port to the AnalyzeWindow, along with testing sorting capabilities.
- 4) Finalization and Reporting: Addressed minor memory leaks, conducted final tests, and compiled a comprehensive project report.
- 5) Live Capture Mode Development: Developed and tested the Live Capture mode, enabling real-time packet capturing and sorting.

- **Challenges and Mitigations:**

- 1) Adapting to uncovered protocols required iterative testing, resolved by allocating more time to the debugging process.
- 2) Ensuring memory efficiency posed challenges, which were mitigated by reviewing and optimizing code.

3. Conclusion

Incorporating a major project like the development of a network analysis tool with packet capturing and integration with Snort IDS into my educational curriculum significantly enhanced my expertise in network security and software development. This project immersed me in a deep understanding of network protocols and standards, allowing me to gain insights into the inner workings of protocols such as TCP/IP, UDP, ICMP, and others, which are fundamental to cybersecurity and network engineering.

The integration with Snort, an industry-standard intrusion detection system, enabled me to develop skills in configuring and utilizing IDS for threat detection. I learned how to interpret alerts and understand attack vectors, which are crucial skills in today's cybersecurity landscape. Implementing

live packet capture equipped me with the ability to handle and analyze high-velocity data streams, a skill highly sought after in fields like real-time security monitoring and incident response.

Furthermore, this project enhanced my understanding of concurrency and multithreading, which are essential for developing concurrent systems that involve synchronization and performance optimization in software design. These are key areas in the development of complex systems and were integral to my technical growth.

Designing and developing a large-scale project from scratch also deepened my expertise in software architecture and object-oriented design, while teaching me how to write maintainable, efficient code. This hands-on experience with software engineering best practices proved invaluable. Additionally, using modern development tools and languages such as the Qt framework for UI design and C++ for backend development not only enhanced my technical skills but also made me proficient with powerful development tools.

I also faced unique problem-solving challenges associated with network security, such as ensuring the integrity and confidentiality of data. This project encouraged me to develop user-centric designs, enhancing my ability to create user-friendly interfaces that simplify complex processes, a critical aspect of software usability.

Lastly, managing this project from proposal to completion, including reporting on progress and outcomes, improved my project management skills. It also refined my ability to communicate complex technical information clearly, preparing me for future professional challenges in technical and project management roles. This comprehensive project was not only a cornerstone of my academic experience but also a significant step in preparing for a career in technology and cybersecurity.

3.1. Lessons Learned

During the development of the network analysis project, I immersed myself in several advanced skills and knowledge areas that typically extend beyond the scope of a standard classroom setting, providing a deep dive into practical and applied aspects of network security and software development.

I gained an in-depth understanding of network intrusion detection through practical use of Snort. This experience went beyond theoretical concepts to include real-world configurations of intrusion detection systems (IDS) and hands-on alert handling. Furthermore, the project provided advanced network protocols analysis, where I not only studied various protocols theoretically but also gained hands-on experience with packet analysis and manipulation.

A significant area of learning was real-time data processing. I developed techniques to handle and analyze live network data, a crucial skill in network monitoring and security but not often covered extensively in traditional academic courses. Another key skill area was concurrency in practice. Implementing multithreading and understanding the challenges of concurrent programming in real-world applications gave me practical insight beyond the basics often taught in programming courses.

The project also emphasized software architecture, involving architectural decisions to create a scalable and maintainable system. I applied design patterns like Singleton and Observer, which are typically discussed in theory but were fully realized in this project. In user interface design, I applied UI/UX principles using the Qt framework, gaining insights into event-driven programming and responsive design techniques.

Practical debugging and problem-solving were critical throughout the project as I developed systematic approaches to debugging complex systems and refined problem-solving skills through live testing and issue resolution. Project and time management were also crucial, as I learned to balance multiple development tasks, meet deadlines, and adapt to unexpected challenges that required reprioritizing features or adjusting the project's scope.

Performance optimization was another vital learning area, with a focus on the importance of performance in security applications and learning how to optimize code for efficiency. This is especially critical in tools that handle large volumes of data. Finally, comprehensive reporting was emphasized, where I developed the ability to document the project process, decisions, and outcomes effectively, translating technical progress into comprehensive reports suitable for various audiences.

Through this experience, I not only enhanced my technical skills but also developed a robust set of soft skills, including communication and critical thinking, vital for my future career in technology and cybersecurity.

3.2. Closing Remarks

Throughout the course of developing the network analysis tool, the project transcended traditional classroom learning, offering a comprehensive and immersive experience in network security, software engineering, and system design. This hands-on project emphasized the importance of practical skills in real-world scenarios, enabling me to gain a deeper understanding of network protocols, intrusion detection systems, and live data analysis. It fostered an appreciation for the complexities of cybersecurity, highlighting the necessity of continuous learning and adaptation to keep pace with evolving threats.

Key learnings from the project included an in-depth exploration of network protocols, uncovering their vulnerabilities, and ways they can be exploited or protected. Integrating with Snort provided firsthand experience with intrusion detection systems, offering insights into how such tools are configured and utilized in professional environments. Handling live packet captures illustrated the challenges and techniques for analyzing high-velocity network data, crucial skills in cybersecurity operations. The project also advanced my programming skills, particularly in C++ and Qt, demonstrating the importance of robust software design in creating efficient and user-friendly applications. Moreover, it underscored the importance of concurrency in software development, especially for performance-critical applications like network monitoring tools.

This project was not merely an academic exercise but a bridge to the professional world of network security, offering a preview of the challenges and satisfactions of working in this dynamic field. The hands-on experience gained is invaluable, providing a solid foundation for a career in cybersecurity.

or network engineering. Moreover, it emphasized the importance of a proactive and inquisitive approach to learning, encouraging continuous exploration and self-improvement to keep pace with the fast-evolving technological landscape.

In conclusion, the project was a testament to the power of practical application in solidifying theoretical concepts and sparking a deeper interest in network security. It served as a reminder that in the world of technology, the classroom is everywhere, and learning is an ongoing journey.

To meet supervisor requirements for further improvement, several enhancements are suggested for the MainWindow including implementing additional file type checks beyond the .pcap extension to verify file content, ensuring robustness against incorrect file formats. Offering multi-language support can cater to a global user base, and integrating a progress bar will inform users about the status of file processing operations. For the PcapReader, increasing the efficiency of packet reading by implementing multi-threading, which allows simultaneous reading and processing of multiple .pcap files, introducing a caching mechanism for frequently accessed data to reduce read times, and upgrading the log map to a more performance-oriented data structure like a hash table would improve lookup times. Improvements to SnortRunner Interoperability could automate Snort updates and rule-set downloads to keep the IDS up-to-date with the latest threat intelligence. Providing customization options for Snort rules within the application will enable users to tailor the intrusion detection process, and incorporating AI-based algorithms to predict and classify potential false positives would enhance the accuracy of threat detection. Enhancements to PacketCatcher Robustness should include developing a mechanism to sort live capture sessions for later analysis or for audit trails, and implementing network traffic anomaly detection in real-time, triggering alerts for unusual patterns. For the AnalyzeWindow, incorporating an advanced search with support for regular expressions to filter through packet data efficiently and introducing customizable dashboards to monitor network traffic and threats in a user-friendly manner are recommended. General Application Improvements should focus on providing comprehensive documentation and tutorials to assist new users in navigating the application, performing regular security audits to update the application to patch vulnerabilities, and maintaining robust security, and adding an incoming/outgoing packet graph to visualize network traffic.

4. Appendix

4.1. Approved Proposal

https://drive.google.com/file/d/1EpHlbR28ohHo6gAAWLQO20pX7ct_jf31/view?usp=sharing

4.2. Project Supervisor Approvals

[Include written approvals from the project supervisor indicating that they've approved both the proposal and report. Also include any changes that have been approved by the project supervisor. Please note that without written approvals from the project supervisor, the committee may not review the final report.]

5. References

- Deitel, H. M., & Deitel, P. (2017). *C++ How to Program (10th ed.)*. Pearson International.
- Martin, K. & Hoffman, B. (2015). *Mastering CMake: A Cross-Platform Build System*. Kitware, Inc.
- Qt Company. (2022). *Qt 6 Documentation*. Retrieved from [<https://doc.qt.io/qt-6/classes.html>]
- Cisco Systems, Inc. (2023). *Snort User's Manual*. Retrieved from [<https://www.snort.org/documents>]

6. Change Log

< Supervisor feedback >

- “write it in paragraphs, not bullet points, and be clear about the benefits/reasons for doing it”
 - ⇒ removed most of the bullet points and changed into paragraphs (p.3-49, except timelines and milestones)

< Committee's feedback >

- “For all the diagrams, please add description on how those interfaces/systems interact with each other - currently we only have images, but no reasoning behind that design”
 - ⇒ Added p.12(Design), p.13-14(System/Software Architecture Diagram), p.15-16 (Class Diagram)