Intrusion Detection and Network Protocol Analysis Application

COMP 8047 – Major Project Report

Adam Krawchuk – A00966518
3-23-2023

Table of Contents

1. Introduction

1.1. Student Background

I am a student of the Network Security Applications Development program at BCIT. Prior to this I graduated with distinction from the Information Systems option of the Computer Systems Technology program. This project has leveraged these programs to create a robust and complete application.

1.2. Education

British Columbia Institute of Technology 2019 - 2021

Computer Systems Technology – Graduated with Distinction

Information Systems Option – Diploma

GPA: 83%

British Columbia Institute of Technology 2021 - 2023

Network Security Applications Development – Graduating 2023

Bachelor of Technology

GPA: 82%

1.3. Project Description

The Intrusion Detection and Network Protocol Analysis Application is a unique network protocol analyzer that integrates components of an intrusion detection and prevention system (IDS/IPS) into a user-friendly graphical interface. Unlike traditional IDSs and IPSs, this project offers analysis of packet captures with graphing, filtering, and summarization features. The project highlights packets detected through Snort to allow the user to easily identify malicious activity within any packet capture. The application also allows users to kill ongoing connections or even block offending IP addresses. Several quality-of-life improvements over the industry-standard Wireshark are implemented. Filtering is improved to query from the previous result when appropriate, and options are available to allow for quicker and more ergonomic navigation of packet captures.

1.3.1. Essential Problems

Existing network protocol analyzers lack the ability to detect malicious activity, while IDS/IPSs typically lack graphing capabilities and graphical user interfaces. Although the foremost network protocol analyzer with a GUI, Wireshark, offers some benefits, it falls short when it comes to navigating a packet capture. Applying filters to large packet captures can be time-consuming,

and Wireshark applies filters to the entire packet capture, even when adding an "and" condition to a previous filter. This makes filtering a tedious process that encourages manual scrolling as an alternative. Additionally, Wireshark lacks an auto-scroll feature or key-combinations that would speed up the process of navigating a packet capture.

1.3.2. Goals and Objectives

This project's goal is to create a network protocol analyzer, complete with graphing, filtering, and summarization features that also notify users of malicious activity within their network traffic. This functionality sets it apart from existing network protocol analyzers that are incapable of identifying malicious activity. Further, the application seeks to improving filtering capabilities by allowing querying of previous filters results. Additionally, I wanted to improve the efficiency of analyzing packet capture by adding convenient options for navigation.

2. Body

2.1. Background

The initial idea for this project came while searching a large packet capture for malicious activity using Wireshark. Since Wireshark has no features that significantly improve speed of navigating large packet captures, this quickly became tedious. Using filters to narrow the packet capture helped, but applying these filters to the capture still took time. Worse yet, it took the same amount of time to apply an additional condition to a previous filter, implying that the query is being applied to the entire packet capture instead of the displayed packets.

The project changed direction to include IDS components alongside the quality-of-life improvements of the original idea. Since finding malicious activity -even with better navigation- is still a challenge. To address this an IDS should be incorporated to help identify packets that are part of the malicious activity.

This is not a company sponsored project.

2.1.1. Definitions

In this section, key terms used throughout the application and report are defined. While this report assumes some familiarity with these concepts, the aim is to clarify any confusion to ensure a clear understanding of this report and project.

Malicious activity: A suspicious connection made over a network. Attacks such as denial-of-service, brute-force password guessing, and port scanning activity are examples of malicious activity.

Packet: A formatted segment of data carried across a network. Packets are combined at the destination to facilitate communication between machines.

Network Protocol Analyzer: Software used to log traffic on a network. Often has features to aid in analysis of the traffic such as graphing, filtering, and summarization capabilities.

Intrusion Detection System (IDS): A system that monitors a network for malicious activity. If malicious activity is detected, an analyst is notified to investigate.

Intrusion Prevention System (IPS): A system that monitors a network for malicious activity. If malicious activity is detected the system takes action to block or timeout the source.

Rule: A set of conditions that match traffic patterns on a network that are considered malicious or suspicious. Each rule has a component to define how to respond if the conditions are met.

Ruleset: A collection of rules.

Snort: A highly configurable IPS that allows for custom made rulesets to be applied to network traffic.

Wireshark: Foremost network packet sniffer. Used to capture packets and has extensive options for analyzing packet captures.

2.2. Project Statement

The application should help users seeking to efficiently navigate packet captures. The application should also have features typical of tradition protocol analyzers such as filtering, graphing, and summarizing while also highlighting malicious packets. Additional options for blocking IPs and killing connections should also be included.

2.3. Possible Alternative Solutions

Below is a list of alternative technologies or solutions that were considered while creating the application.

2.3.1. Programming Language

Programming Language	Pros	Cons
Python	• Simple syntax • Large list of libraries • Object Oriented	• Pseudo-multithreading • Slower performance
C++	• Fast performance • Object Oriented • Large community	• Security issues • Pointers • No garbage collection
Java	• Fast performance • Object Oriented	• High memory use
C	• Best performance	• Security issues • Complex

2.3.2. GUI Library

GUI Library	Pros	Cons
Tkinter	• Previous experience • Simple to use • Most popular	• Issues with asynchronous code
PyQt	• Good documentation • User friendly • Extensive components list	• No previous experience
PySide	• Used commercially • Large number of resources for learning	• Requires payment

2.3.3. Filtering Library

Filtering Library	Pros	Cons
Create own filtering system	• Learning opportunity • Easier to visualize flow of logic • Likely to be less performant	• Issues with asynchronous code
PyParsing	• Good documentation	• No previous experience • Few tutorials
PLY	• Straightforward • Extensive debugging and reporting	• No previous experience • Few tutorials

2.3.4. Packet Gathering Library

Packet Gathering	Pros	Cons

Scapy	• Easy to use • Previous experience • Good documentation	• May miss packets if network is very active
Raw Sockets	• Bypasses network stack • Fewer packets missed on active network	• Must manually handle every byte

2.3.5. Operating System

Packet Gathering	Pros	Cons
Fedora	• Access to Iptables commands • Easier access to processes • Access to Snort	• May miss packets if network is very active
Windows	• Already installed on home machine	• Must find alternatives for blocking IPs • Must find alternative to Snort

2.4. Chosen Solution

Below is the list of technologies or solutions that were selected for creating the application. Most options were selected due to previous experience using the technology.

2.4.1. Programming Language

The application was ultimately written in Python due to my previous experience with creating network applications using it. The large number of available libraries for Python that I have previous experience using also contributed to this decision. A faster programming language would have likely been more appropriate to ensure no packets were dropped, but my lack of experience creating network applications with them would have led to an unfeasible development timeline.

2.4.2. GUI Library

Tkinter was chosen primarily because of its simplicity to create detailed applications. I knew from previous experience that everything I needed in a GUI was available in Tkinter. PyQt was also used briefly at the beginning of development but was found unintuitive to work with.

2.4.3. Filtering Library

I decided to create my own filtering system without the use of libraries such as PyParsing or PLY. This was mainly due to a lack of examples of comparable filtering languages produced by either.

Tutorials for these libraries were similarly difficult to find. By creating my own filtering system I was able to learn more about filtering languages and more easily track the flow of logic within them. However, it is likely that my implementation of filtering is less performant and could stand to be improved.

2.4.4. Packet Gathering Library

Scapy was selected for the purpose of sniffing packets on the network. This was primarily due to my previous experience with Scapy. The major downside to Scapy is that it may miss packets when used on a more active network. This was mitigated by having packet sniffing done on its own process. Raw sockets was also looked at as a possible solution to this problem but after testing it was discovered that raw sockets would also miss packets. Additionally, the use of raw sockets would require a complex system to create packet objects from the bytes sniffed on the network.

2.4.5. Operating System

Fedora was the easy choice since it meant that easy access to tools such as Snort and the Iptables command. Additionally, while my home computer is a windows machine it does have a Linux virtual machine available. Further, had I made a windows or a cross-platform application I would have had to find and learn a new IDS or created one myself which would have drastically increased development time.

2.5. Details of Design and Development

This application's development approach was an iterative, incremental process. The individual tasks were created by breaking down the required windows of the GUI into components. Trello was used throughout the project to track what tasks had been done, and what remained. The following Trello board shows the list of tasks tracked and accomplished throughout the project.

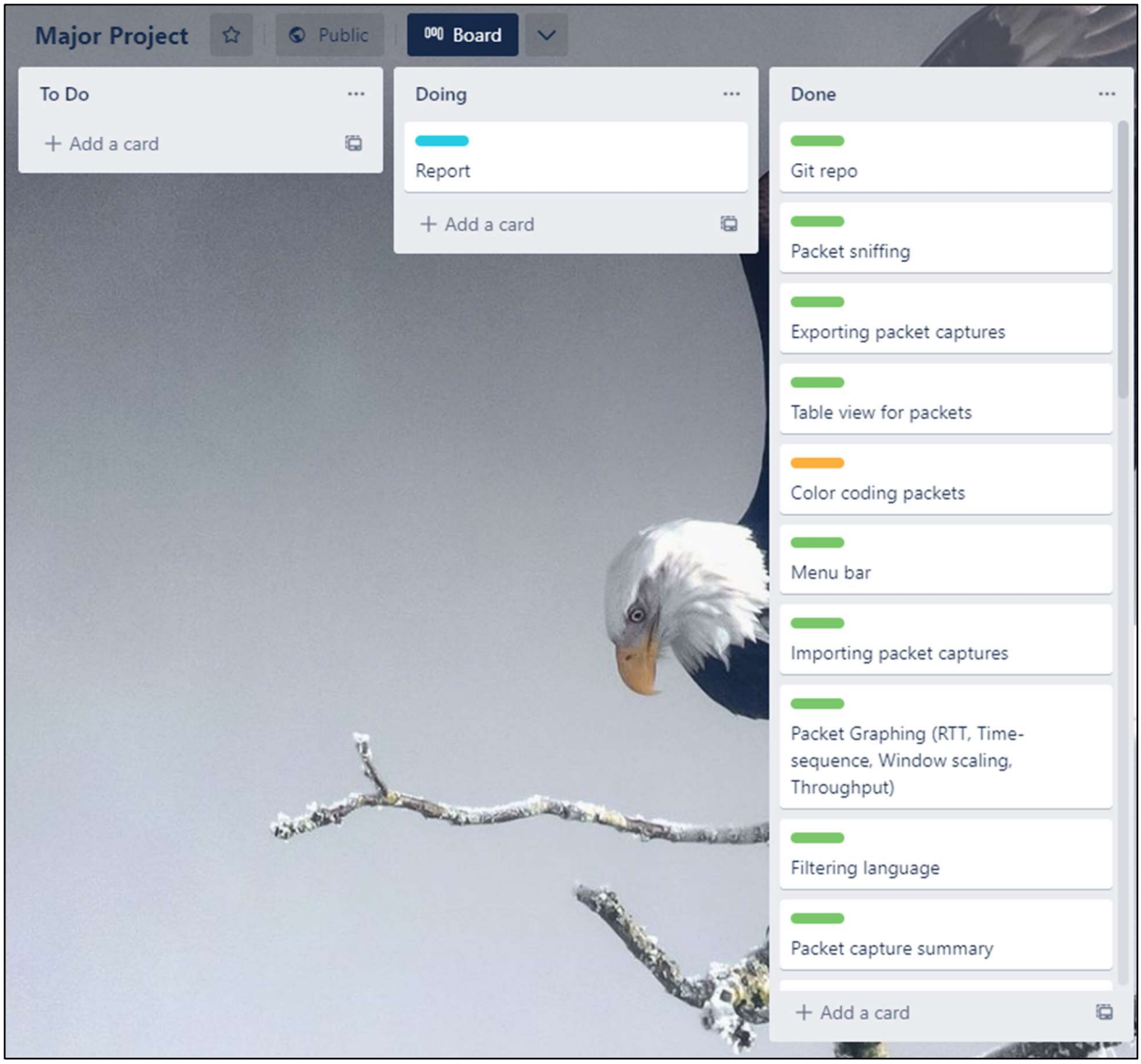


Figure 1: Trello board used for development

2.5.1. Deliverables

The delivered components of this project are as follows:

1. **Network Protocol Analyzer:** Tested and fully useable network protocol analyzer with graphing, GUI, summarization/collation of packet information, and filtering.
2. **IDS/IPS Integration:** A list of up-to-date rules applied to detect malicious activity within packet captures.
3. **Report**: Comp 8047 report detailing the project.

2.5.2. Initial Mock-up

This section provides a glimpse into how I initially envisioned the application. The purpose of this mock-up was to create the layout of the GUI and ensure everything would fit into place and look sensible. Place holder data was added to show how the display could maintain readability. Throughout the development process I strove to maintain the same layout initially envisioned.

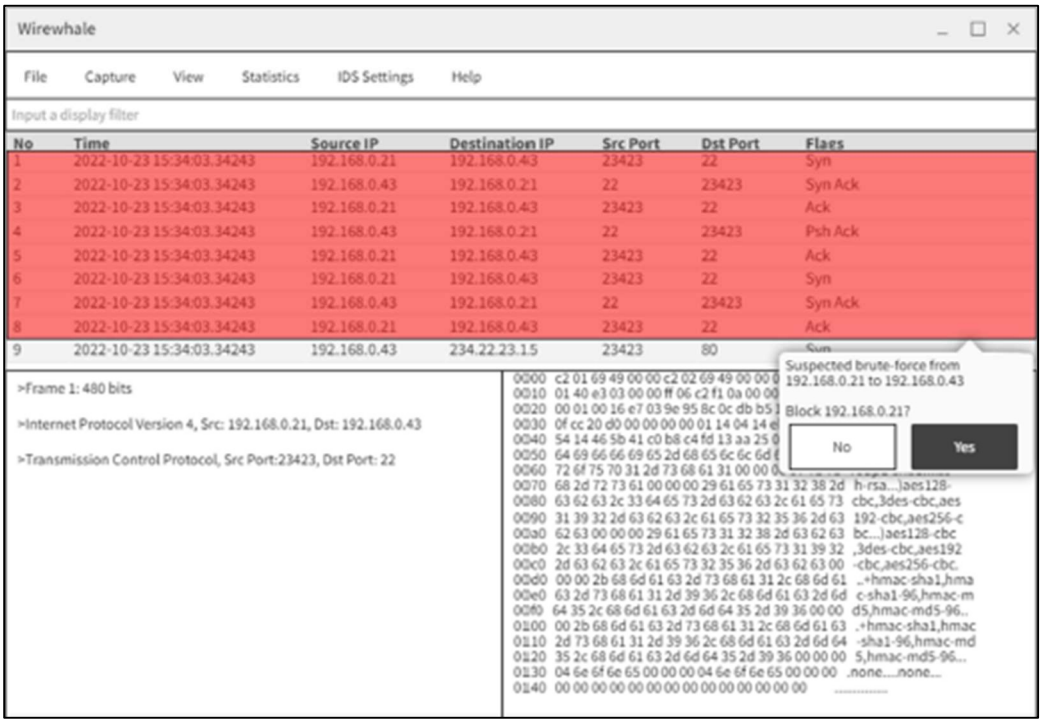


Figure 2: Initial Mock-Up

2.5.3. Context Diagram

The application relies on two components, the GUI application, and Snort. The context diagram below offers a top-down view of the most basic functionality of the application. The user loads or beings capturing packets, and the application updates the GUI with alerts found within Snort's alert file.

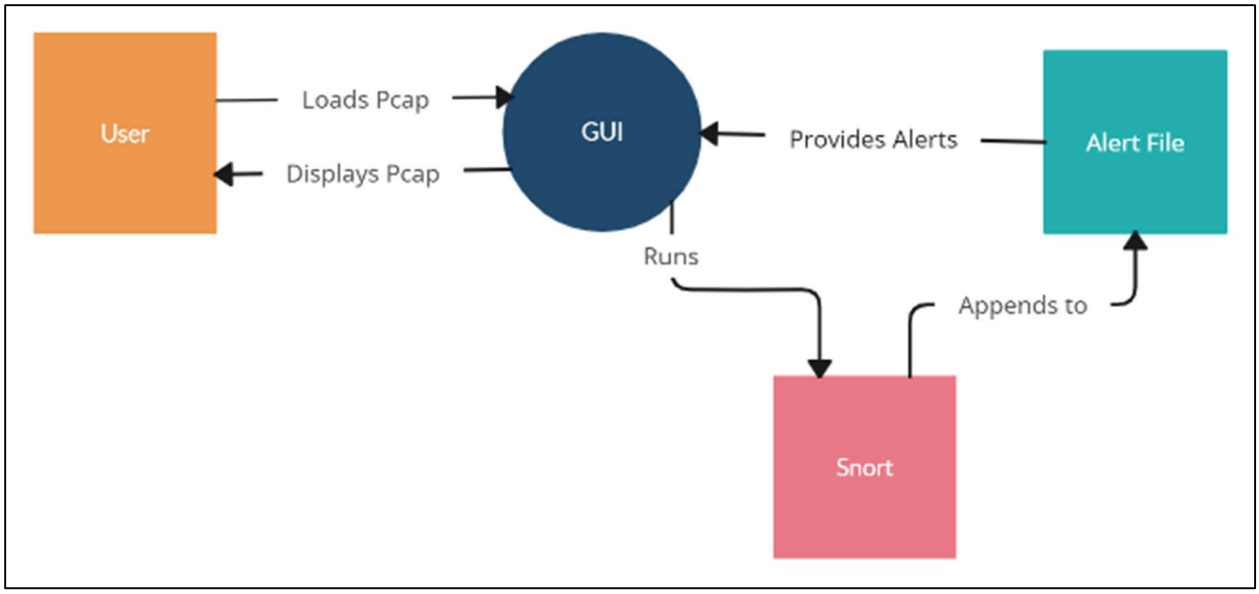


Figure 3: Context diagram of the application

2.5.4. Sequence Diagrams

This section's purpose is to go into further detail on exactly how the application handles packet captures and detects threats within them. When a user loads a packet capture into the application, every packet within it is sent to the packet handler. The packet handler appends the packet to the full packet list and checks the current filter before updating the display. When a scan is initiated, all packets are saved to a temporary packet capture file and sent to Snort for intrusion detection. The IDS handler then checks the Snort logs for new entries and attempts to match the entries to the list of packets. The sequence diagram below displays this process.

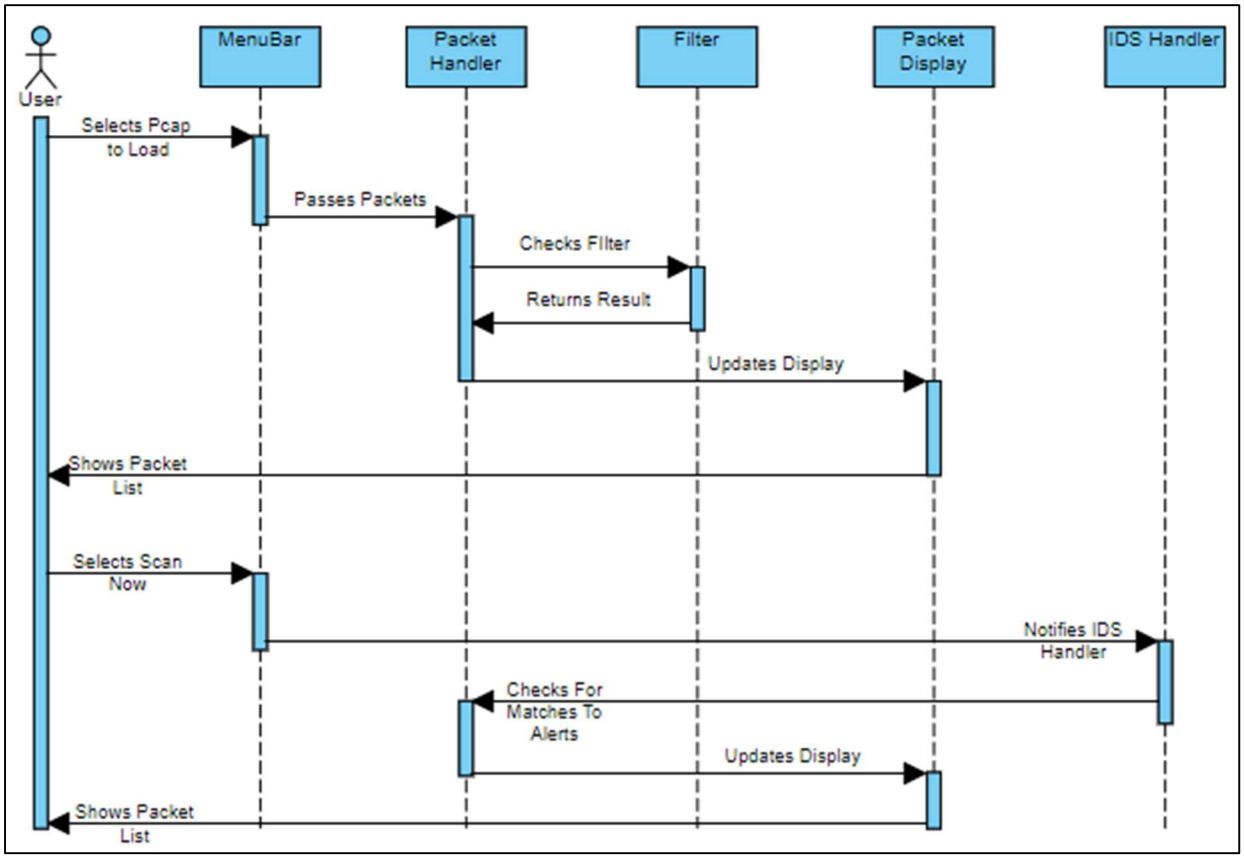


Figure 4: Sequence Diagram of loading and scanning a packet capture

If instead the user wants to sniff packets on their interface, a slightly different method is used to identify malicious activity. Once the user presses the start button, the Scapy begins asynchronously populating a queue with packets. The sniffed packets will follow the same flow of logic as the loaded packets. Where the process differs is that intermittently the Snort alert log directory is checked for new entries. If any are found the IDS handler will note them and inform the packet display to mark them in red. Once the user presses the “stop” button, the sniffer will stop getting new packets and the IDS handler will cease its intermittent scanning. This process is illustrated in the sequence diagram below.

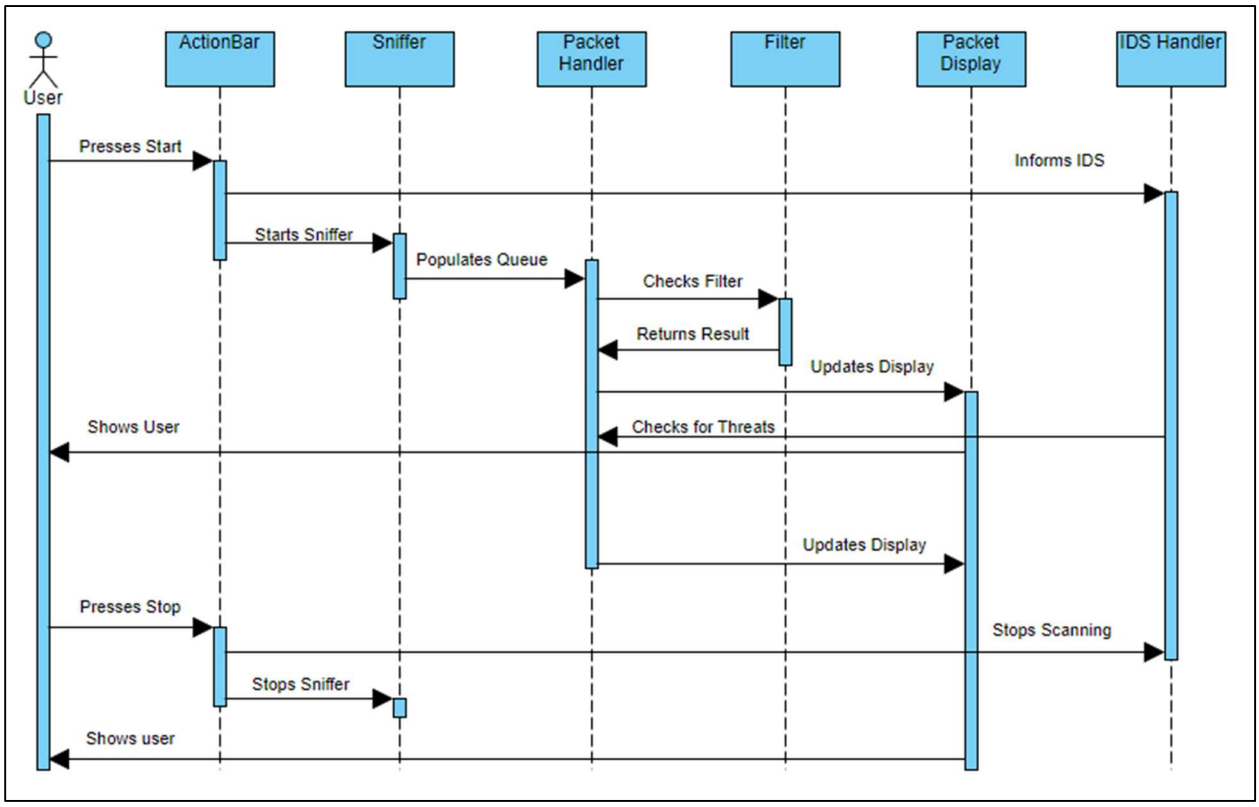


Figure 5: Sequence diagram for sniffing interface

2.5.5. Entity Relationship and Class Diagram

The entity relationship and class diagram below are intended to give a clearer understanding of the structure of the application. Most classes are connected to the main class since most components of the application are simply portions of the overall GUI. Exceptions to this are the graph, summary, and report classes that are created and managed by the menu bar component. The packet sniffer is also connected through the action bar as its action depends on the state of the action bar.

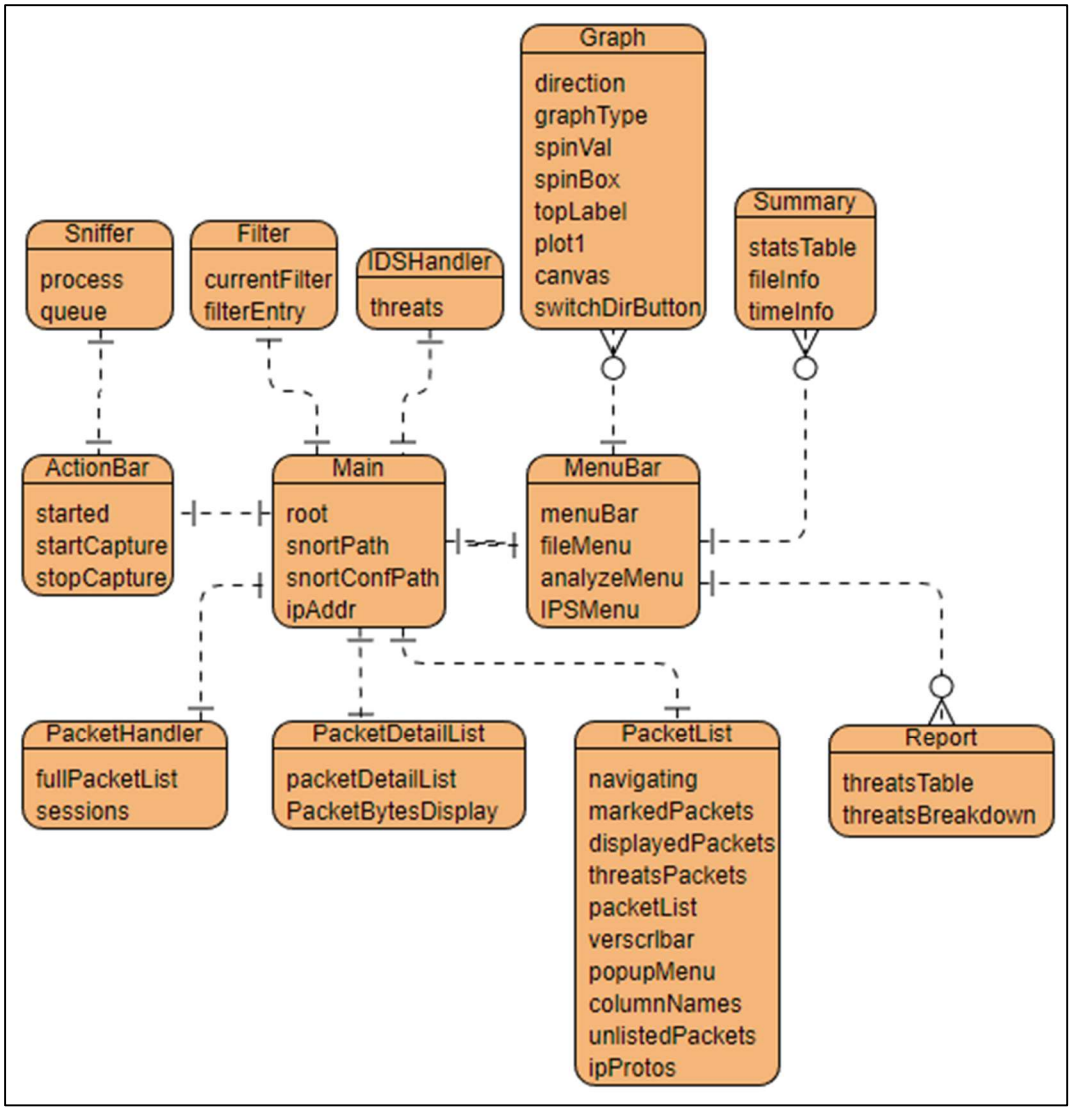


Figure 6: ERD Diagram of the application

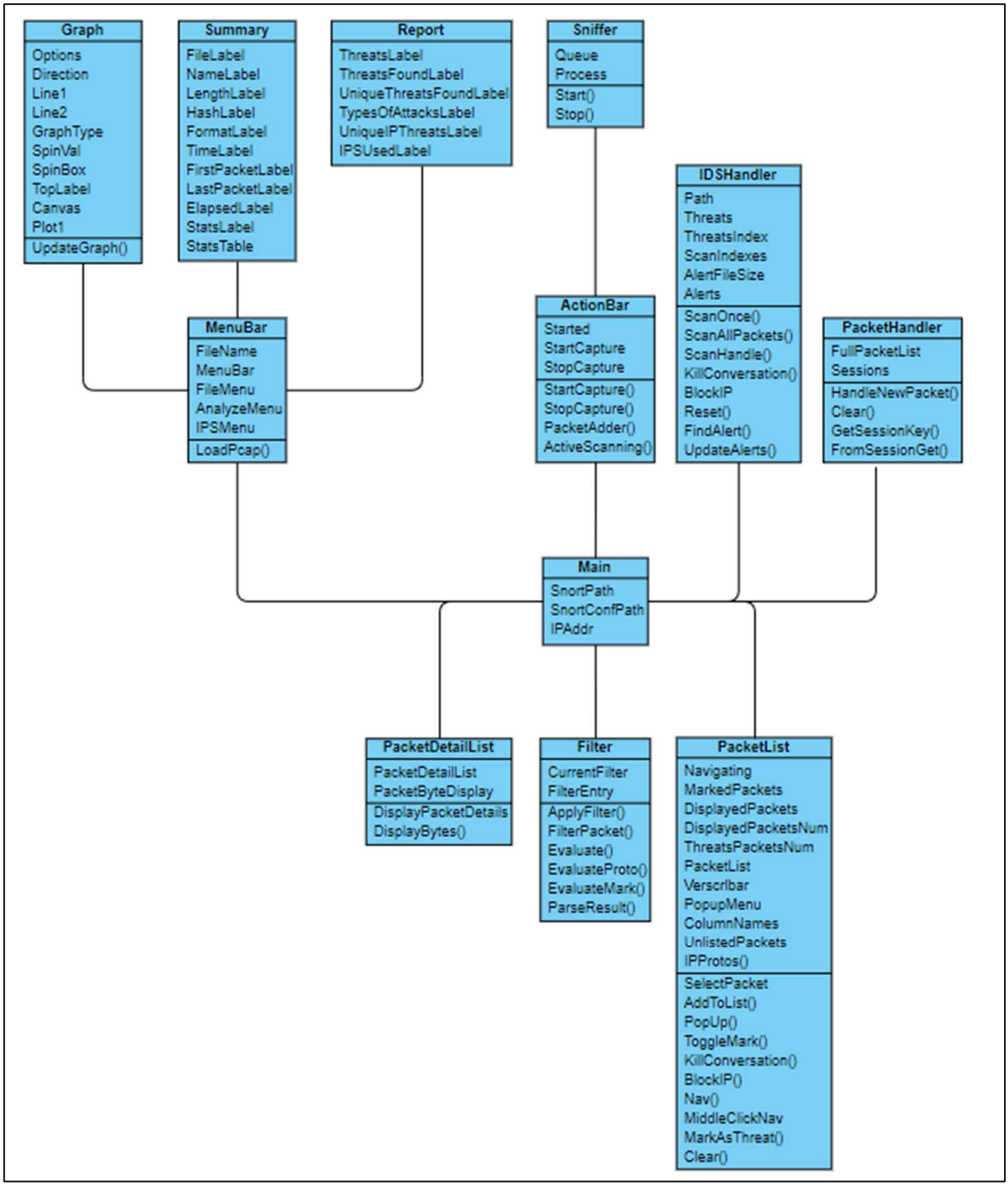


Figure 7: Class Diagram of the application

2.5.6. Installation Guide

This section will guide you through installing and setting up both Snort and the application. Snort is required for any malicious activity to be detected. Below is a step-by-step guide on installing both snort and the application.

2.5.6.1. Snort

1. Download both daq-2.0.7.tar.gz and snort-2.9.20.tar.gz from https://www.snort.org/downloads
2. Extract both to your preferred installation directory.
3. Move to the daq-2.0.7 directory.
4. run the following commands:
    a. dnf install bison -y
    b. dnf install flex -y
    c. ./configure
    d. make
    e. make install
5. Move to the snort directory.
6. Run the following commands:
    a. dnf install libdnet* -y
    b. dnf install pcre* -y
    c. dnf install libtirpc-devel -y
    d. dnf install zlib* -y
    e. dnf install luajit* -y
    f. ./configure
    g. make
    h. make install
7. replace the snort.conf file located in the snort/etc/ directory with the provided snort.conf file.
8. Change the snort.conf file's HOME_NET variable to your home network
9. Add the rules, so_rules, and preproc_rules to the snort/ directory
10. Run mkdir /var/log/snort
11. Run snort using the command: snort -A full -X -c <path to snort.conf> -D

2.5.6.2. Intrusion Detection and Network Protocol Analysis Application

1. Extract the project to the chosen directory
2. run the command: pip install -r requirements.txt
3. run the program using the command: python3 Main.py

2.5.7. User Guide

2.5.7.1. Getting Started

The application is started by running the command: python3 Main.py in the applications installation directory. The GUI should appear as seen below.

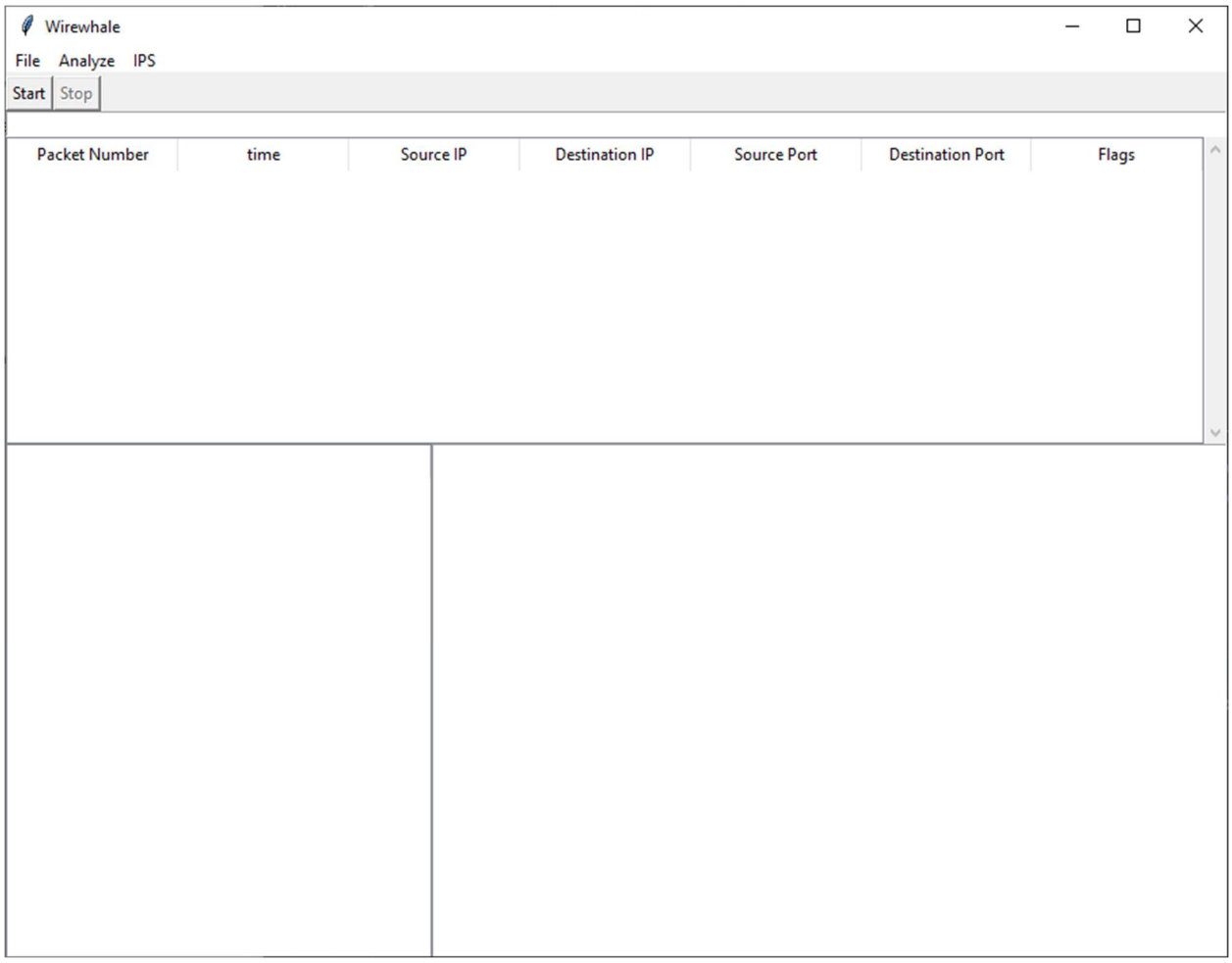


Figure 8: GUI of the application

The packet list can be populated by either loading a packet capture through the “File” -> “Load” menu option or through sniffing the interface by pressing the “Start” action button. Packets should appear as seen below.

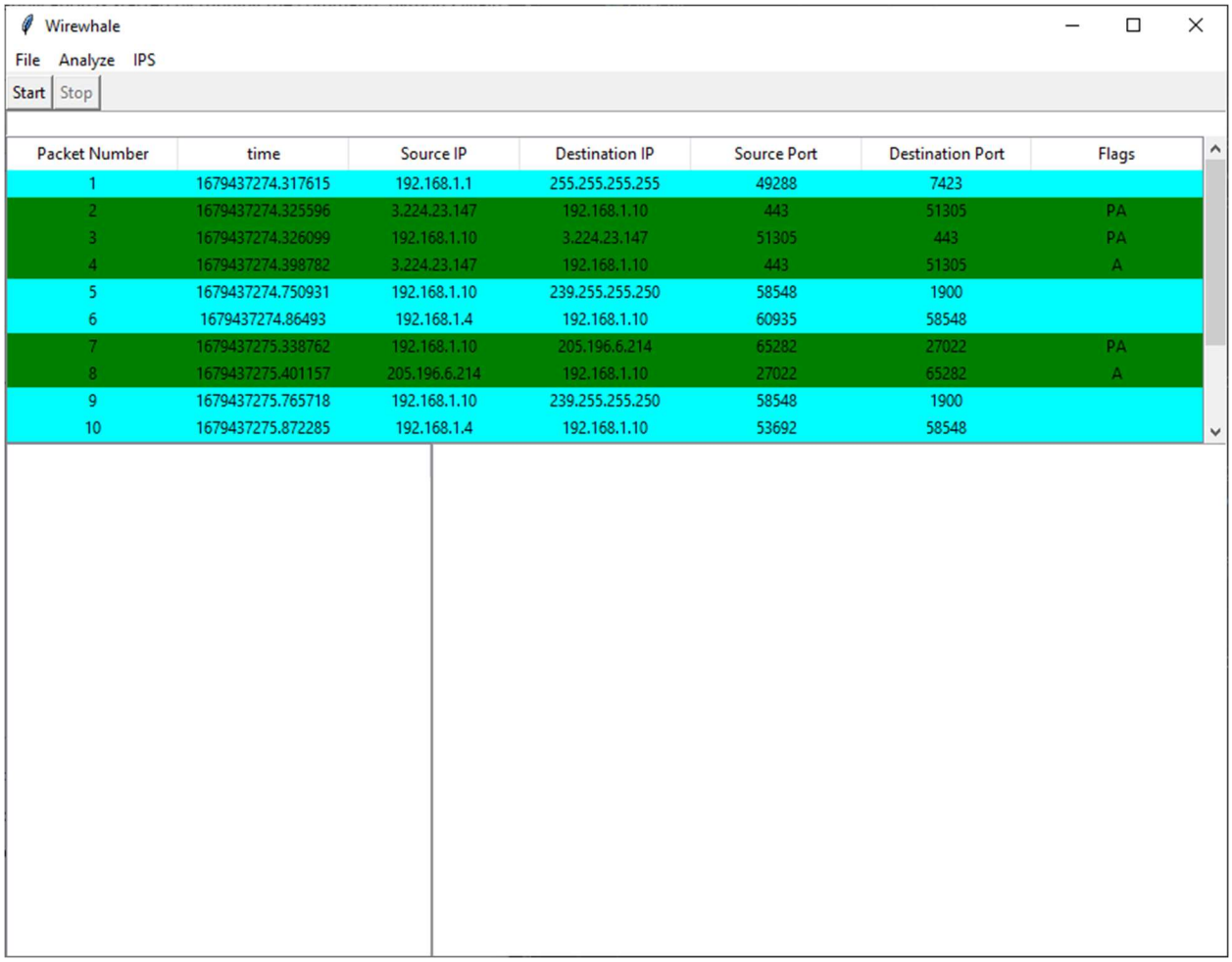


Figure 9: Packets displayed through the application

The color of the packet is determined by the following:

- Red: Indicates suspected malicious activity
- Yellow: Means the packet is marked by the user
- Orange: The packet has an ARP layer
- Green: Indicates a TCP packet
- Cyan: Indicates a UDP packet

The packet capture can be navigated by either using the arrow keys, scrolling, the scrollbar or by middle-clicking the list to enable auto-scrolling. Right-clicking the packet will display a

menu with options for marking the packet, killing the connection, and blocking the source IP. Left clicking on the packets will display their details and a breakdown of the bytes within the packet as seen below.

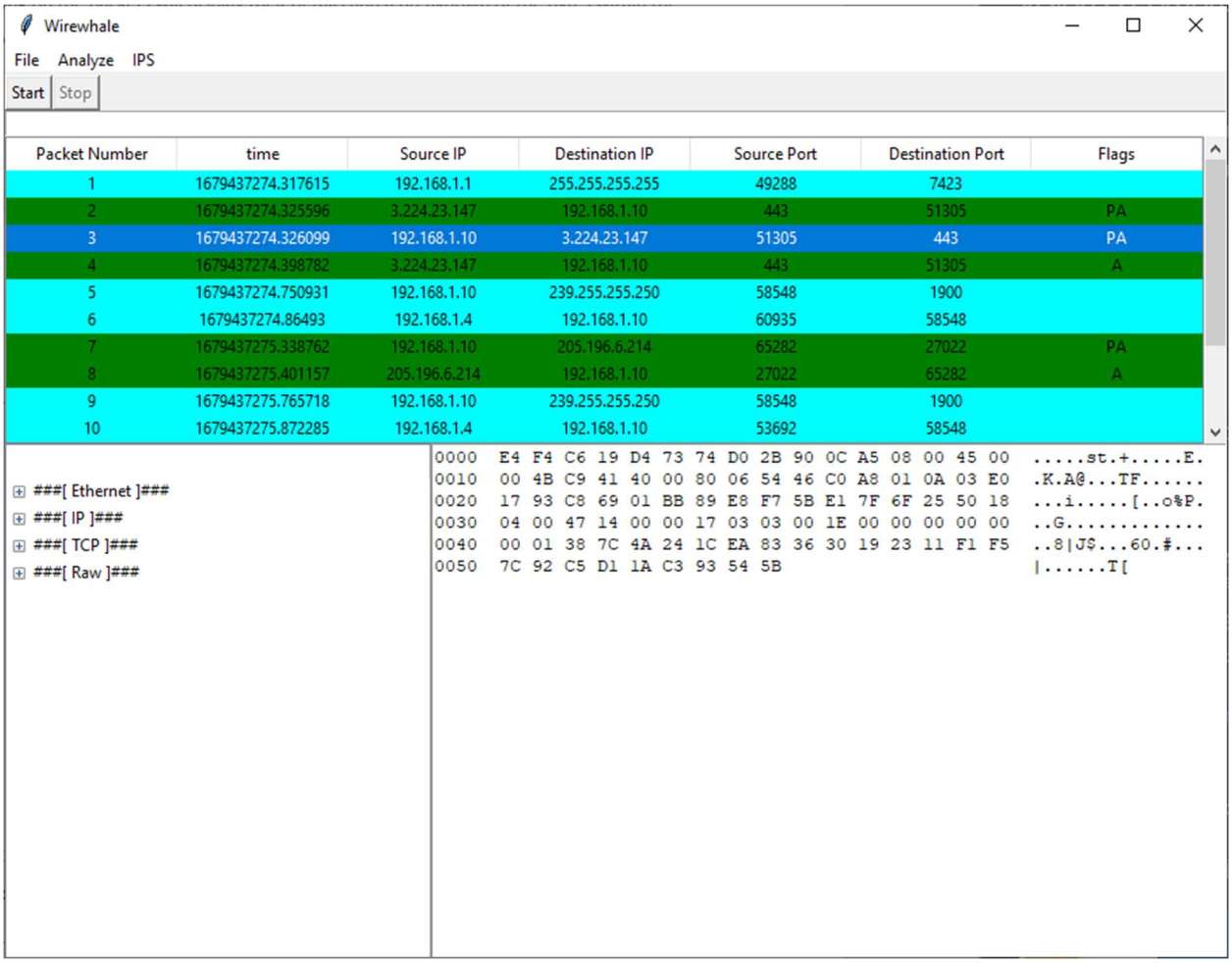


Figure 10: Selected packet details displayed

The layers can be expanded on the left to show the different fields belonging to that layer while the packet bytes can be seen on the right. A summary containing various statistics of the packet capture can be seen by selecting the “Analyze” -> “Summary” menu option as seen below.

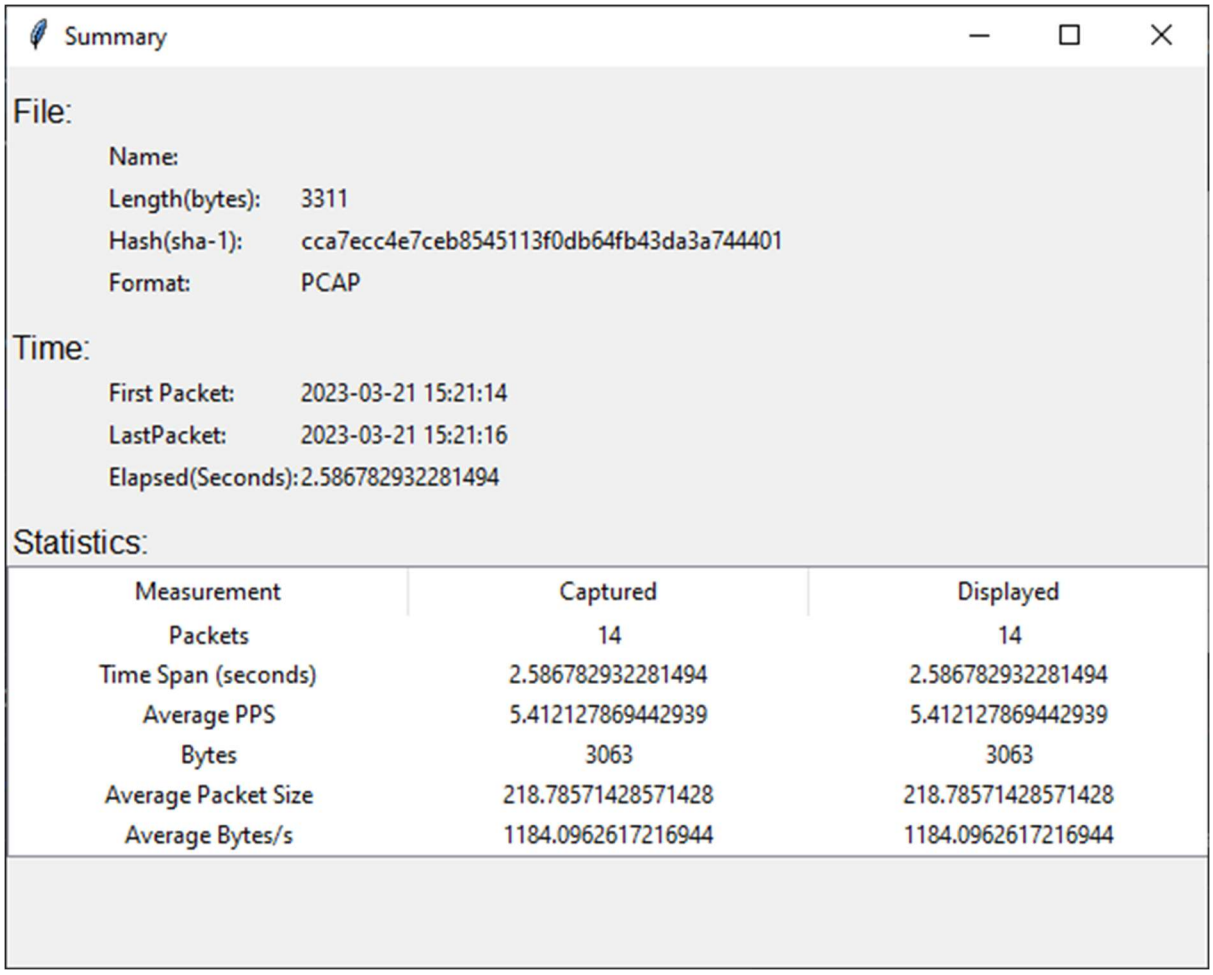


Figure 11:Packet Summary

The “Analyze” menu also contains options for generating a report of threats from the packet capture as well as throughput, window scaling, time sequence, and round trip time graphing options. Selecting the time sequence graphing option provides a graph of sequence numbers over time as seen below.

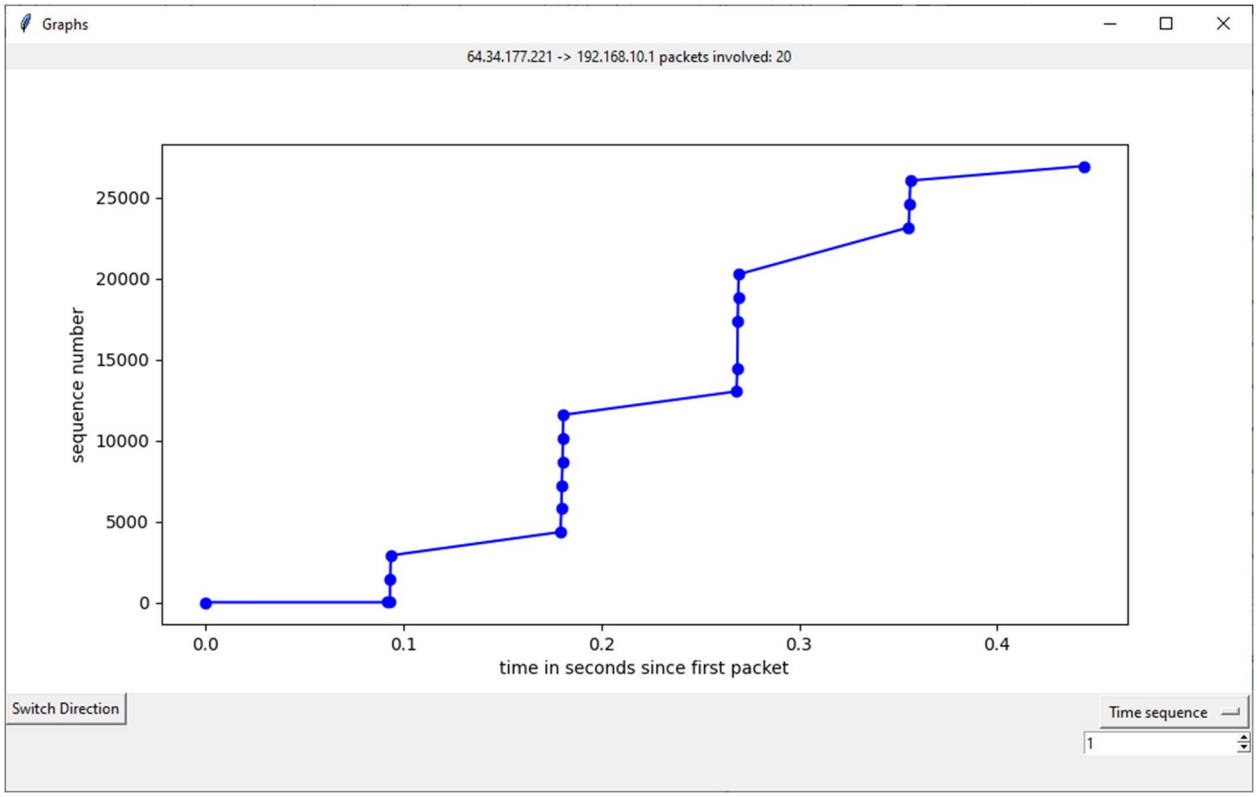


Figure 12: Time sequence graph

The header at the top shows the direction of the packets and the number of packets graphed, while the bottom has options for selecting different streams to follow. There is also a drop-down list available for selecting other types of graphs.

The packet capture can also be saved by using the "File" -> "Save All" or "Save Filtered" options. This will save the packet capture in its entirety or only the displayed packets.

2.5.7.2. Filtering

Filtering options are also provided. Filters should be written in space separated fashion using a "*term comparator value*" pattern. The following table shows the different terms and their expected comparators and values.

Term	Valid Comparators	Valid Value Types
srcip	==, !=	String
dstip	==, !=	String
srcport	==, !=, <=, <, >=, >	Integer
dstport	==, !=, <=, <, >=, >	Integer

time	==, !=, <=, <, >=, >	Integer
flags	==, !=	String
protocol	==, !=	String
marked	==, !=	Boolean

Multiple term-comparator-values can be evaluated using the “and” and “or” logic operators can be used to create more elaborate filters. Parentheses are also supported to give priority to different parts of the filter. An example of filtering can be seen below.

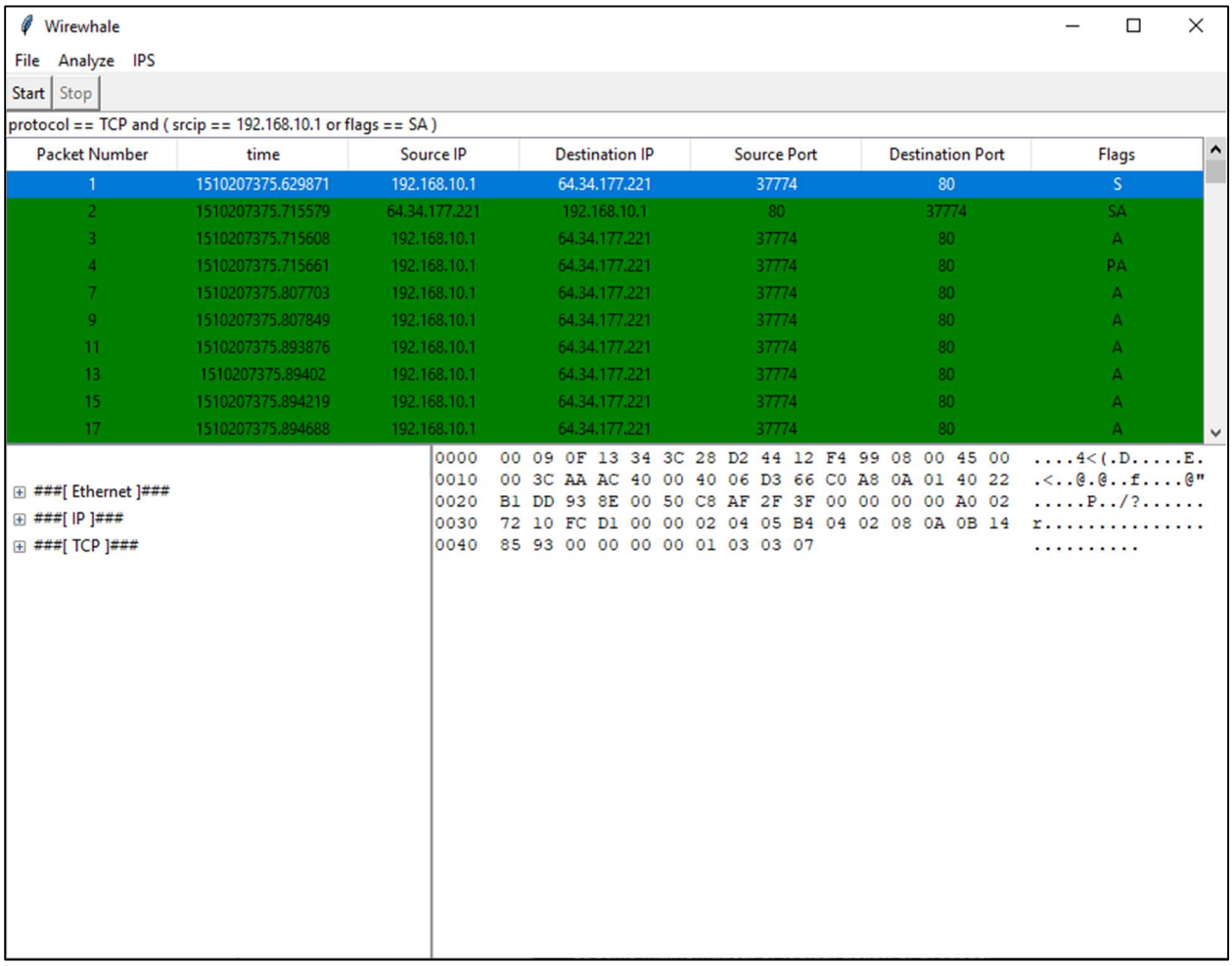


Figure 13: Filtering a packet capture

2.5.7.3. Detecting Malicious Activity

If Snort is not logging to the /var/log/snort directory, then the directory must be first specified by selecting the “IPS” -> “Alert Directory Select” option. The snort.conf file should also be specified in a similar manner using “IPS” -> “Snort Conf Select”. If the application is

actively capturing packets, then it will check the specified alert directory every 5 seconds for changes and attempt to connect alerts given by Snort to any packets within the packet capture. Snort must be running for this to occur.

If you are instead loading a packet capture, you can select the “IPS” -> “Scan Now” option to have the application hand Snort the packet capture to scan. If any malicious activity is detected within the capture, it will be highlighted red as seen below.

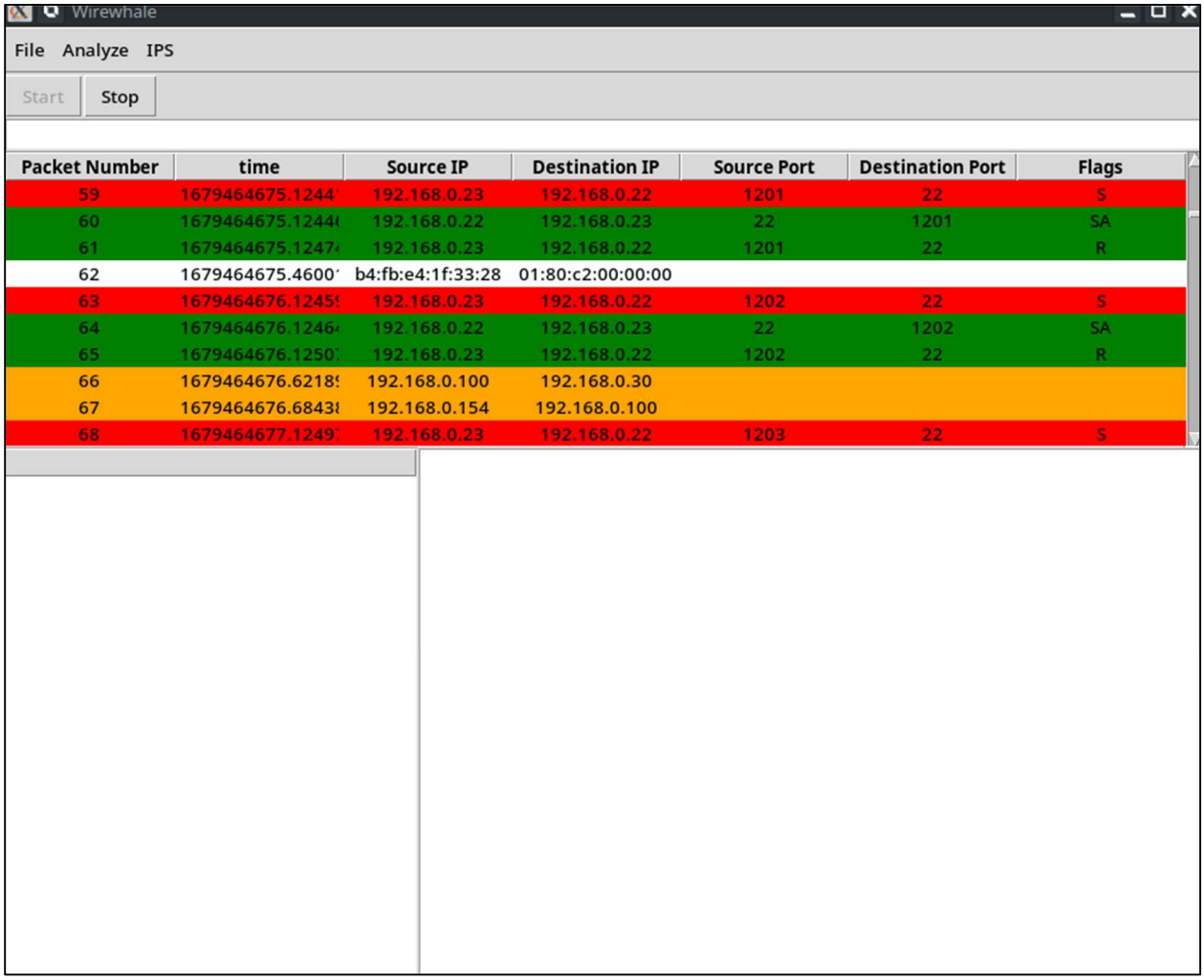


Figure 14: Malicious packets highlighted

2.5. Testing Details and Results

The testing for the application consisted of unit, manual, and stress testing. The testing was done throughout the project and helped to identify bugs early. This has contributed to the reliability of the application.

2.5.8. Unit Testing

This section details the unit testing approach that utilized to test small components of the application. This helped to ensure that changes made to the code didn't adversely affect previous portions. Additionally, writing these tests provided a useful reminder of their corresponding section and helped to maintain a clear understanding of the codebase throughout development.

The goal behind writing each test was to verify that the code tested still operated as expected. Since this is a GUI, inputs given to the application changed multiple portions of the GUI. This meant each test case often had multiple assertions to validate. Below are the results of running the test and a detailing of each test.

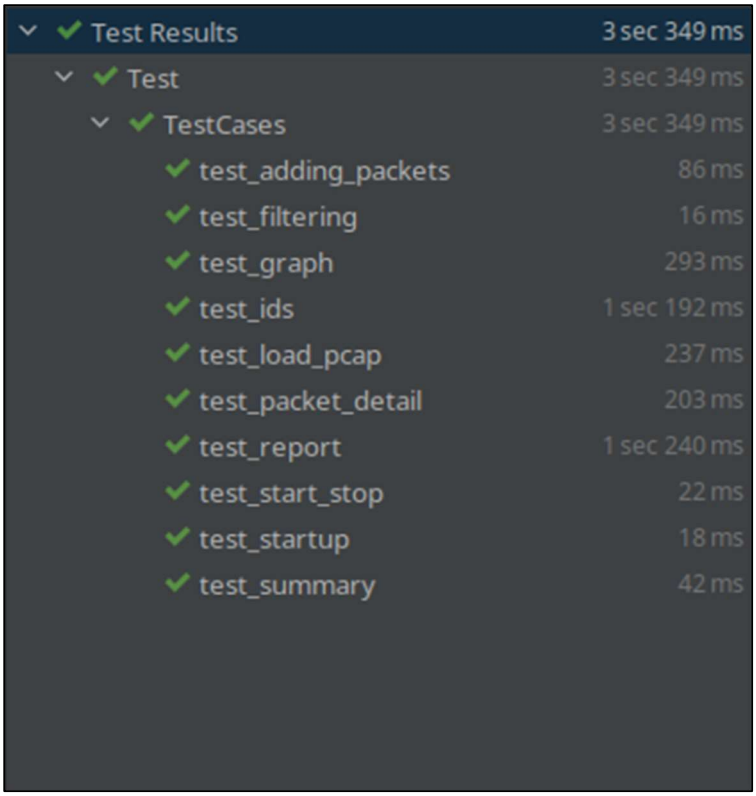


Figure 15: All Unit tests passing

Test Name	Purpose	What was checked
Test_adding_packets	To validate that packets provided to the packetHandler did end up populating the packetList.	First confirmed that the packetList was empty, then after the packets were added, checked that packetList was no longer empty.
Test_filtering	To validate that applied filters caused the proper packets to be displayed.	First confirmed that all packets provided were bring displayed. Next, after applying the filter, the display was checked to ensure that the packets excluded by the filter were no longer there. Finally, after the filter was removed, the display was checked to ensure all packets were once again displayed.

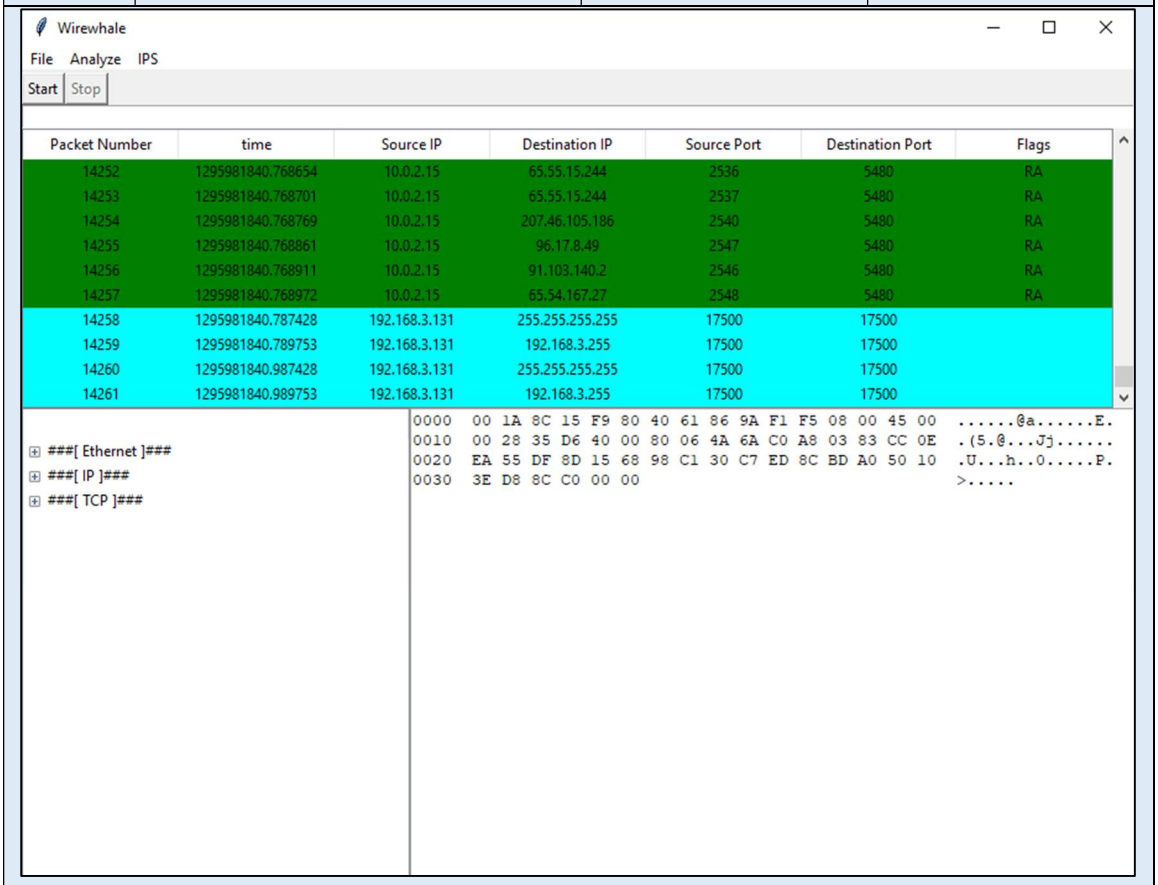
Test_graph	To test that the graphs were displaying information as expected.	After loading an example packet capture, the graph was checked to ensure the correct number of points appeared.
Test_ids	To test that provided a packetList containing malicious activity, that the IDSHandler would detect and catalog the threat.	The IDSHandler was first checked to confirm no threats were listed, then after scanning the packet capture, the IDS was rechecked to ensure that the threat was listed.
Test_load_pcap	To test that loading a packet capture was functioning as expected and that all packets were being properly displayed.	The packetList was first checked to confirm it contained no packets and that the display showed no packets. Next, after loading a test packet capture, the packetList and packet display were checked to confirm that they both contained the same number of entries as listed within the packet capture.
Test_packet_detail	To test the functionality of displaying individual packet details once selected.	The packet detail and bytes display were first checked to ensure they contained no packets. Once a packet was selected, the displays were rechecked to confirm that they now displayed information.
Test_report	To test that the report would provide accurate details of malicious activity provided.	After the application was provided a packet capture containing malicious activity, the labels for number of threats, unique threats, unique IPs involved, type of threats, and IPs used were displaying correct information.

Test_start_stop	To test that the action buttons were starting and stopping the sniffer as expected.	First the state of the application was checked to ensure it was not actively sniffing packets. Once the “start” button was pressed, the state of the application was rechecked to confirm that it was now in an active state. Finally, after the “stop” button was pressed, it is confirmed that the application is no longer in a started state.
Test_startup	To ensure that the application provided a GUI as expected.	Once the application was started a simple check of its displayed name is done to prove that it is running.
Test_summary	To test that the summary portion of the application was displaying accurate information pertaining to the packets provided.	After providing several packets to the application and filtering them, the summary is checked to validate that the labels for the number of packets total and displayed are correct. The times for both displayed and total packets are also confirmed.

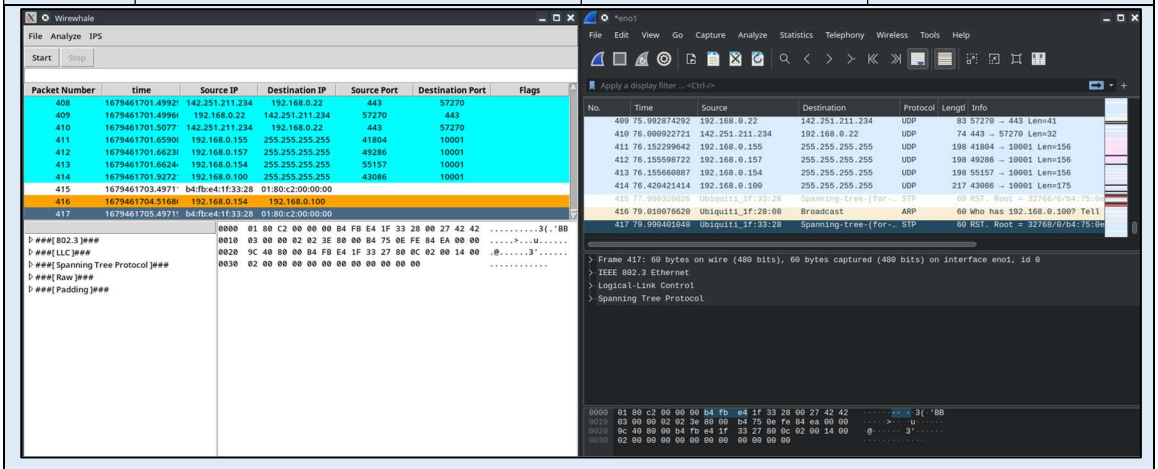
2.5.9. Manual Testing

Manual testing consisted of tests designed to prove the functionality of the application. The below table describes the tests done and their expected and actual outcome. Additionally, a screenshot is included with each to prove the results when possible.

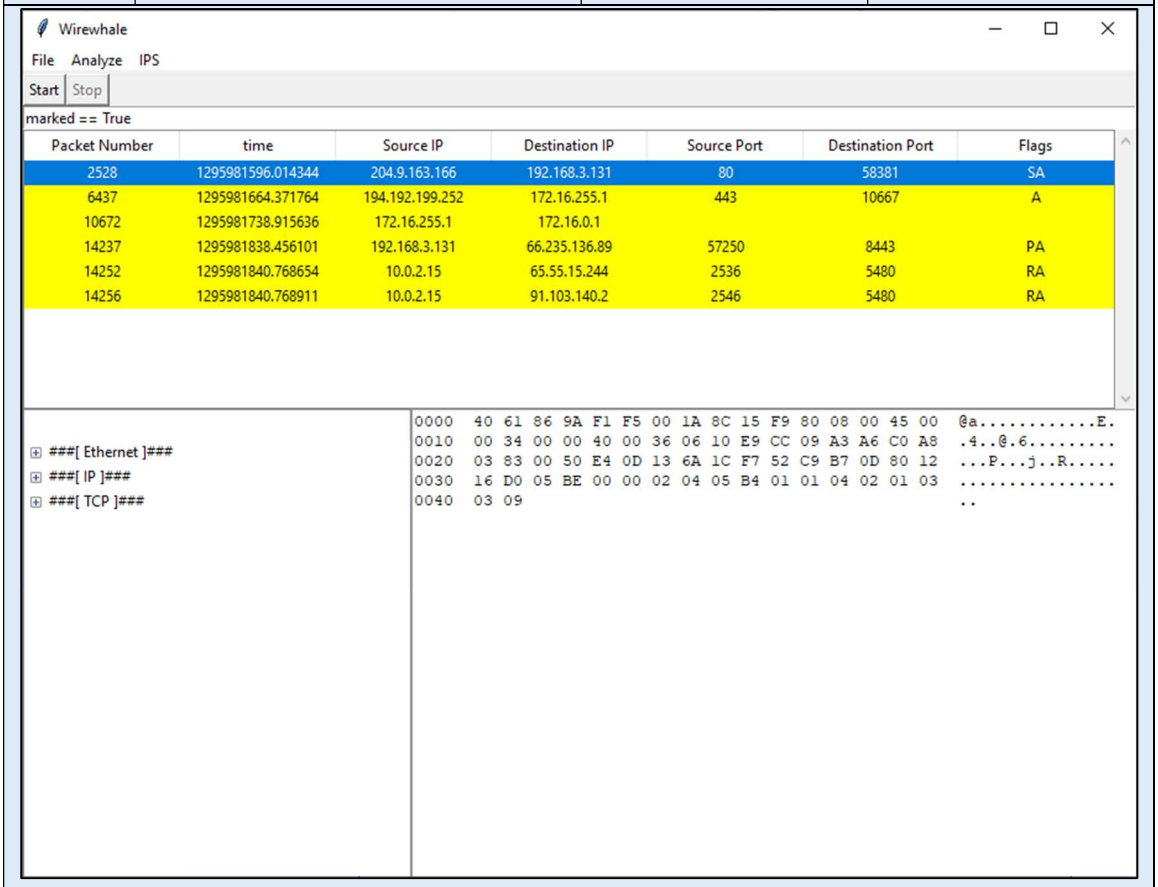
Test No.	Description	Expected Result	Result
1	Opening a packet capture	All packets within the capture displayed	All packets within the capture displayed



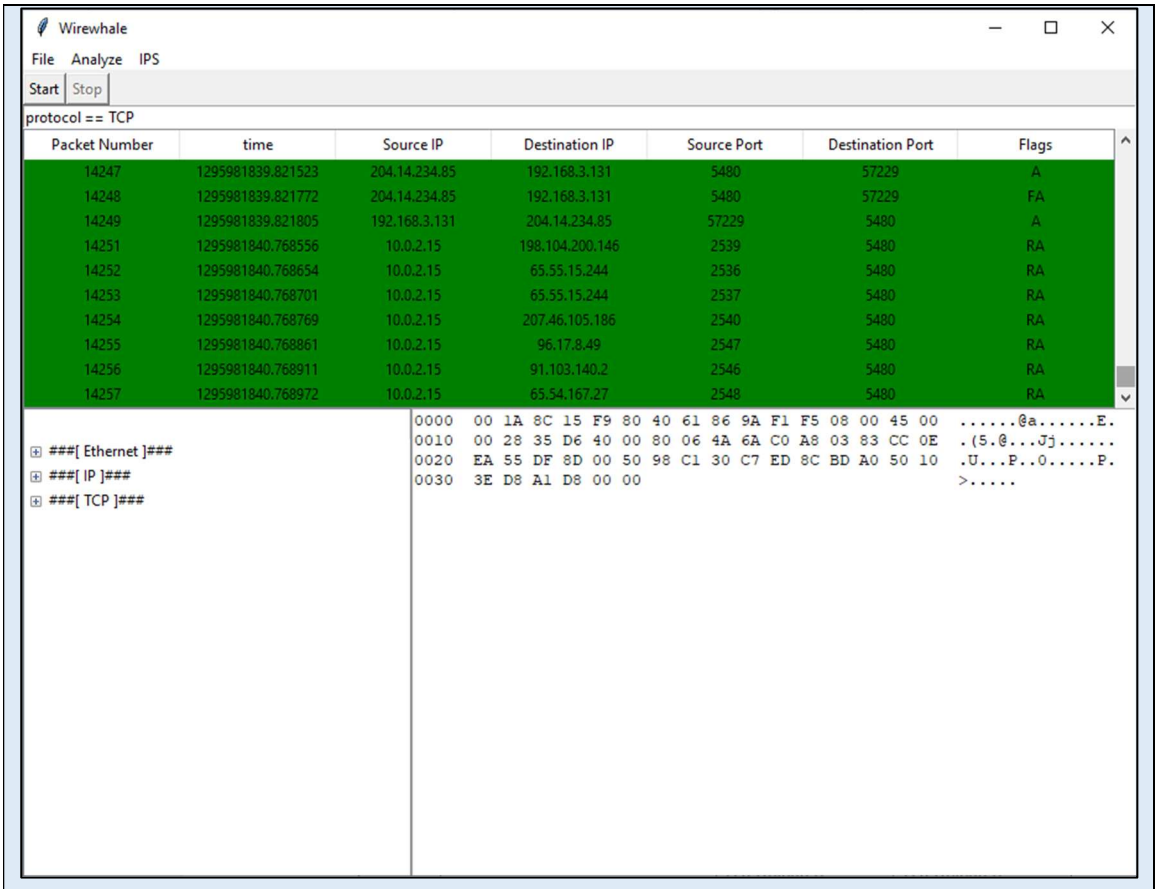
2	Sniffing on the interface provides same number of packets as other network Sniffers	Same number of packets displayed on both captures	Same number of packets displayed on both captures



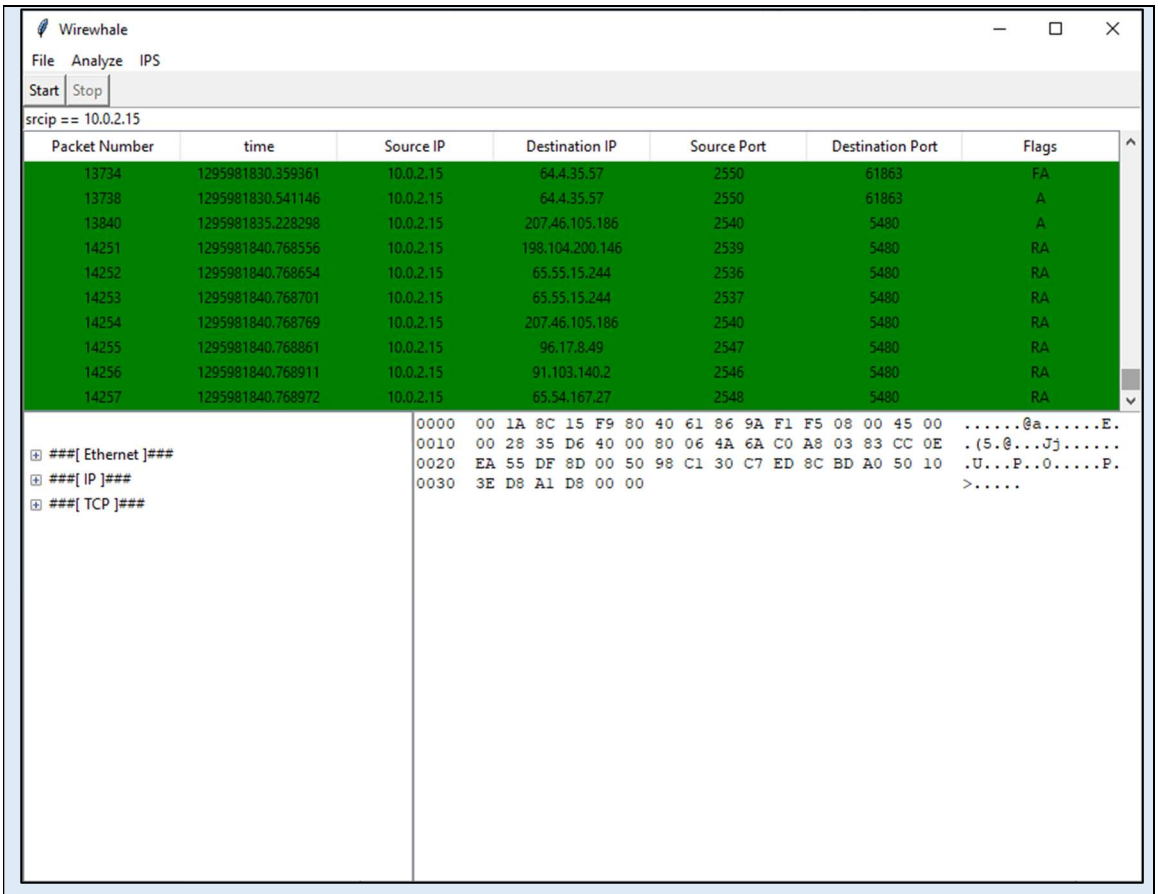
3	Marking packets and filtering to only include them	Marked packets are visually distinct and filterable	Marked packets are visually distinct and filterable



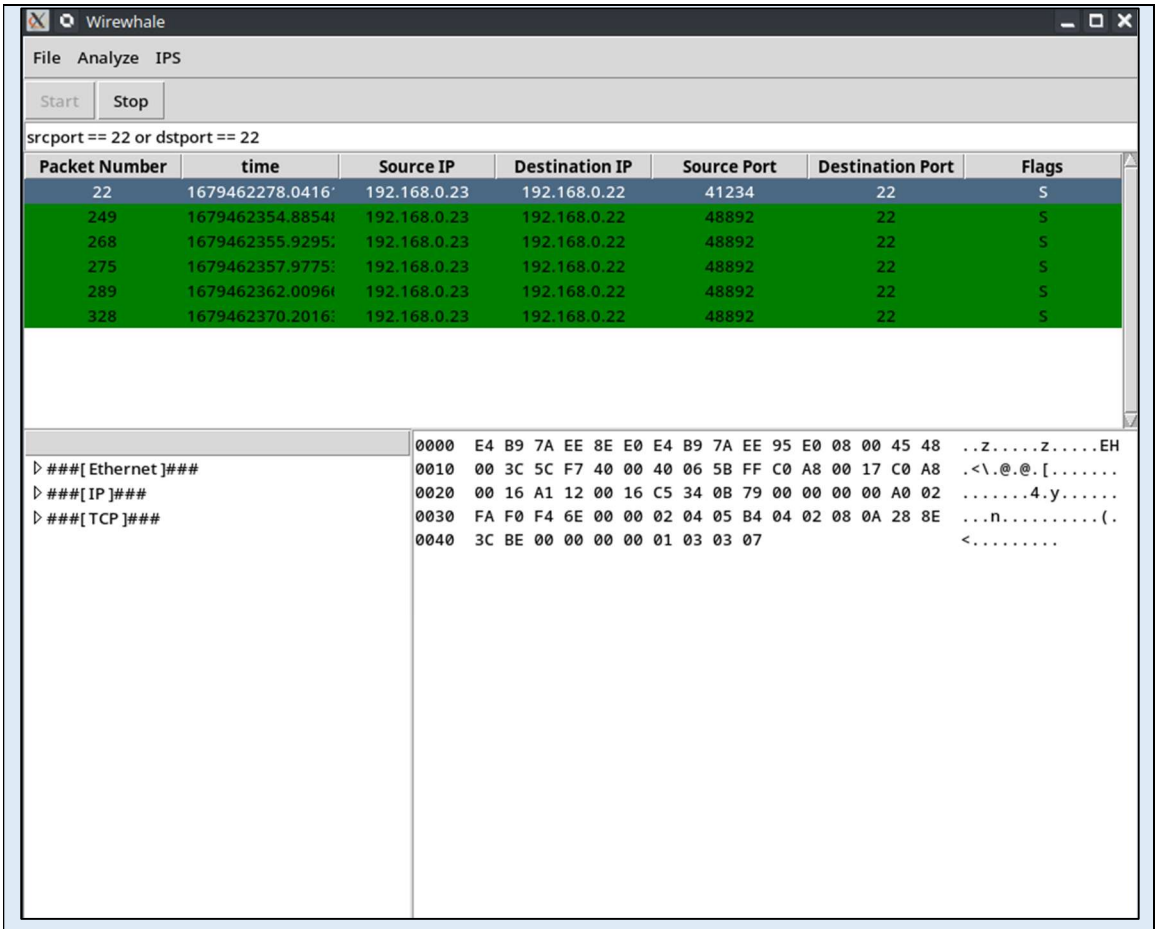
4	Filter only TCP packets	Only TCP packets are visible	Only TCP packets are visible



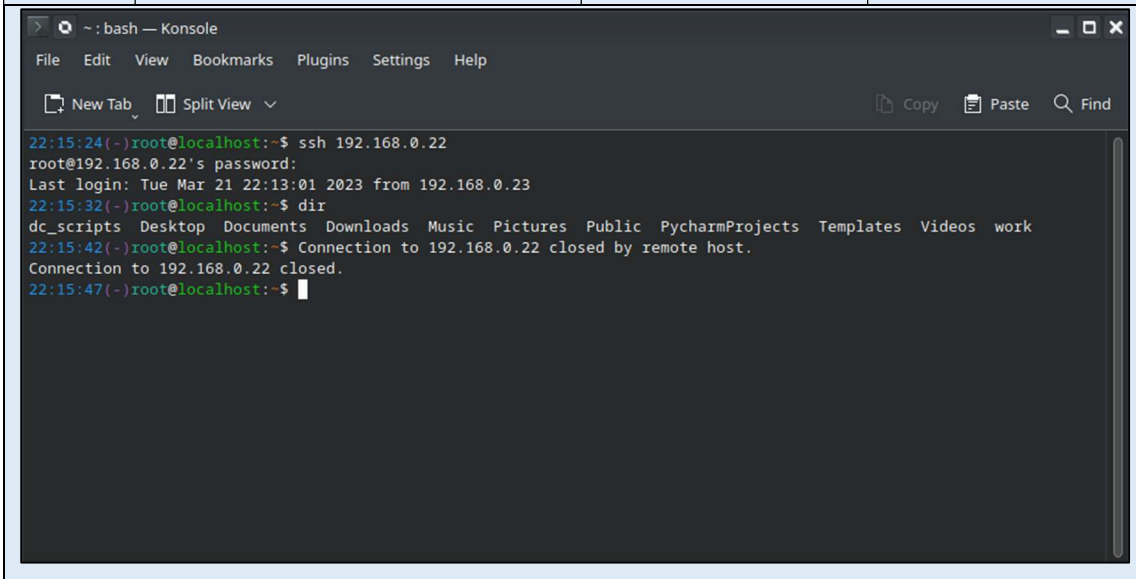
5	Filter only from IP address 10.0.2.15	Only packets from 10.0.2.15 are visible	Only packets from 10.0.2.15 are visible

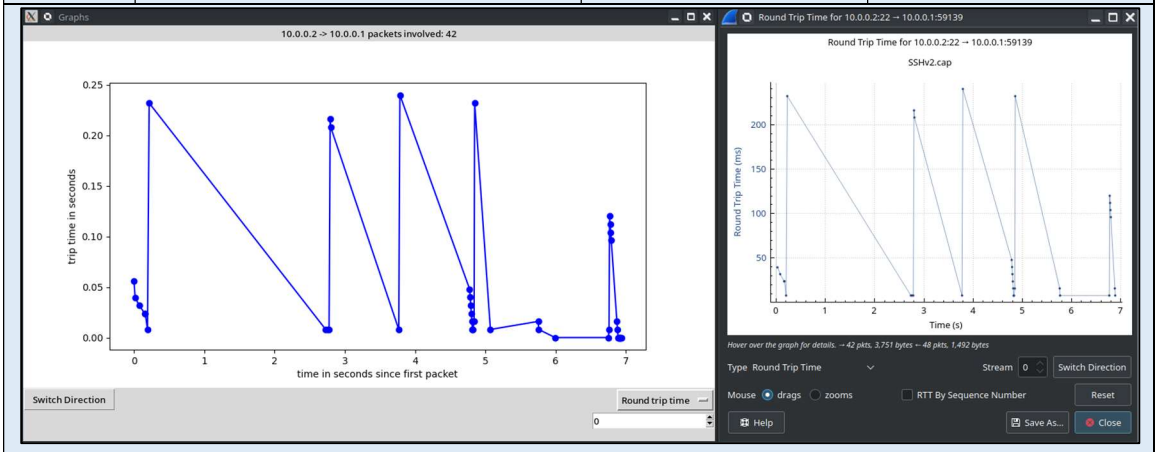
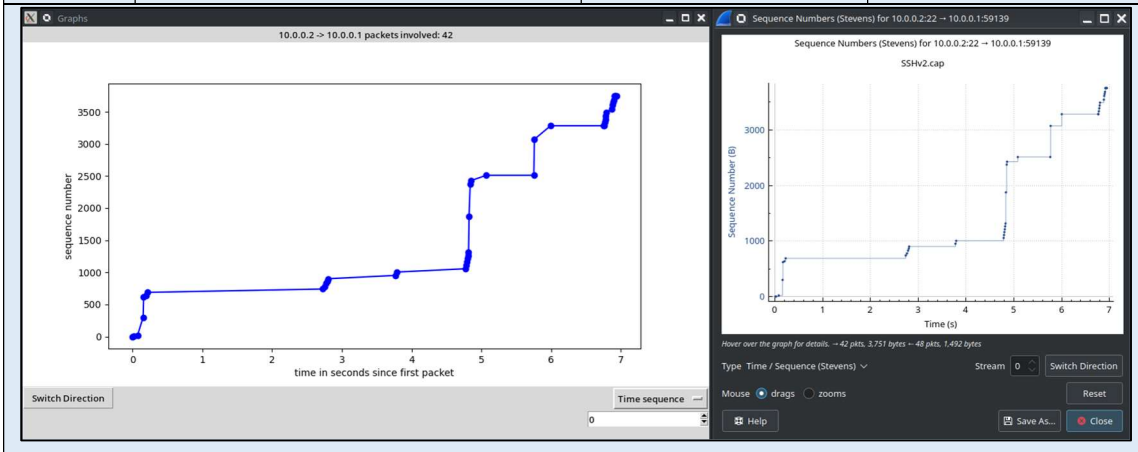


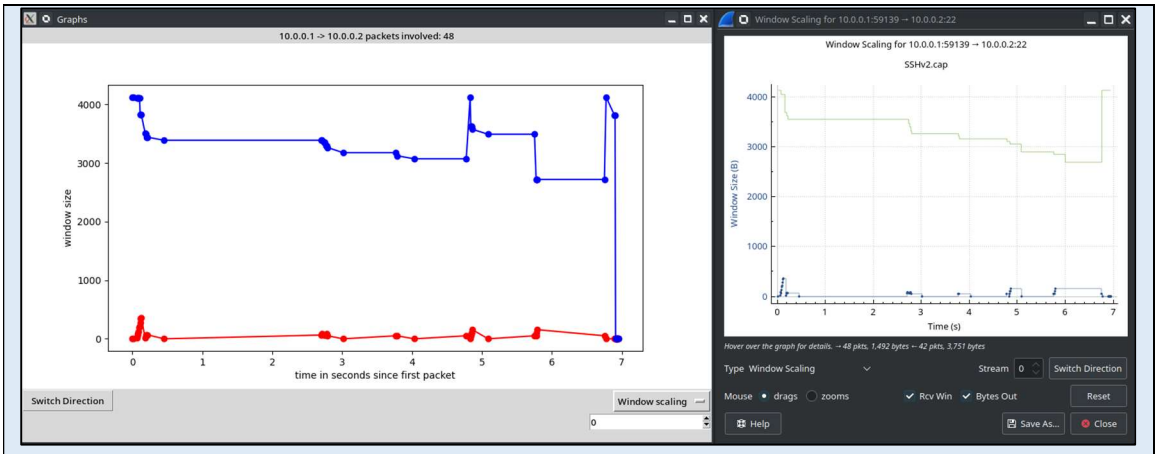
6	Block an IP address	Packets sent from the blocked address are ignored	Packets sent from the blocked address are ignored



7	Kill an SSH conversation	The SSH connection is terminated	The SSH connection is terminated

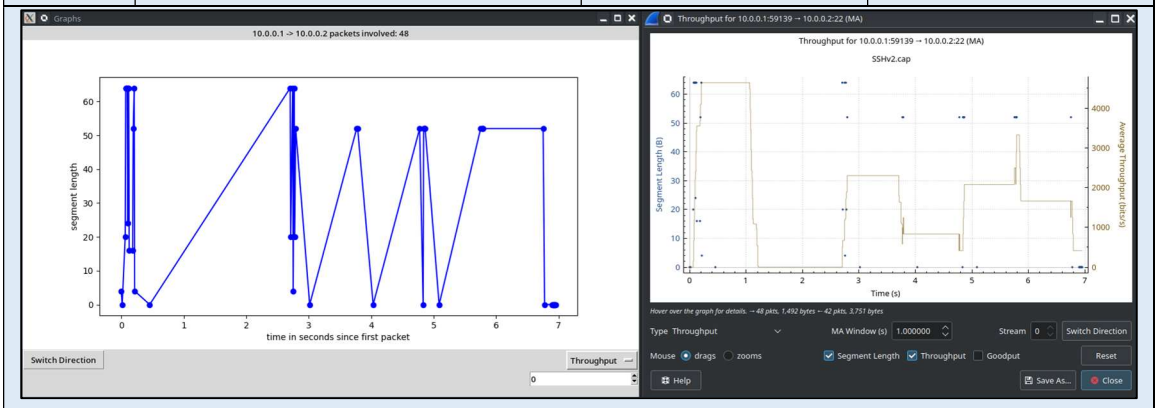


8	Middle click the packet display to navigate quickly	Rapid navigation of the packet capture	Rapid navigation of the packet capture
9	Sample packet capture loaded into both Wireshark and the application and round-trip time graph selected	Both graphs display identical points	Both graphs display identical points
			
10	Sample packet capture loaded into both Wireshark and the application and time sequence graph selected	Both graphs display identical points	Both graphs display identical points
			
11	Sample packet capture loaded into both Wireshark and the application and window scaling graph selected	Both graphs display identical points	Both graphs display some variations (explanation below)



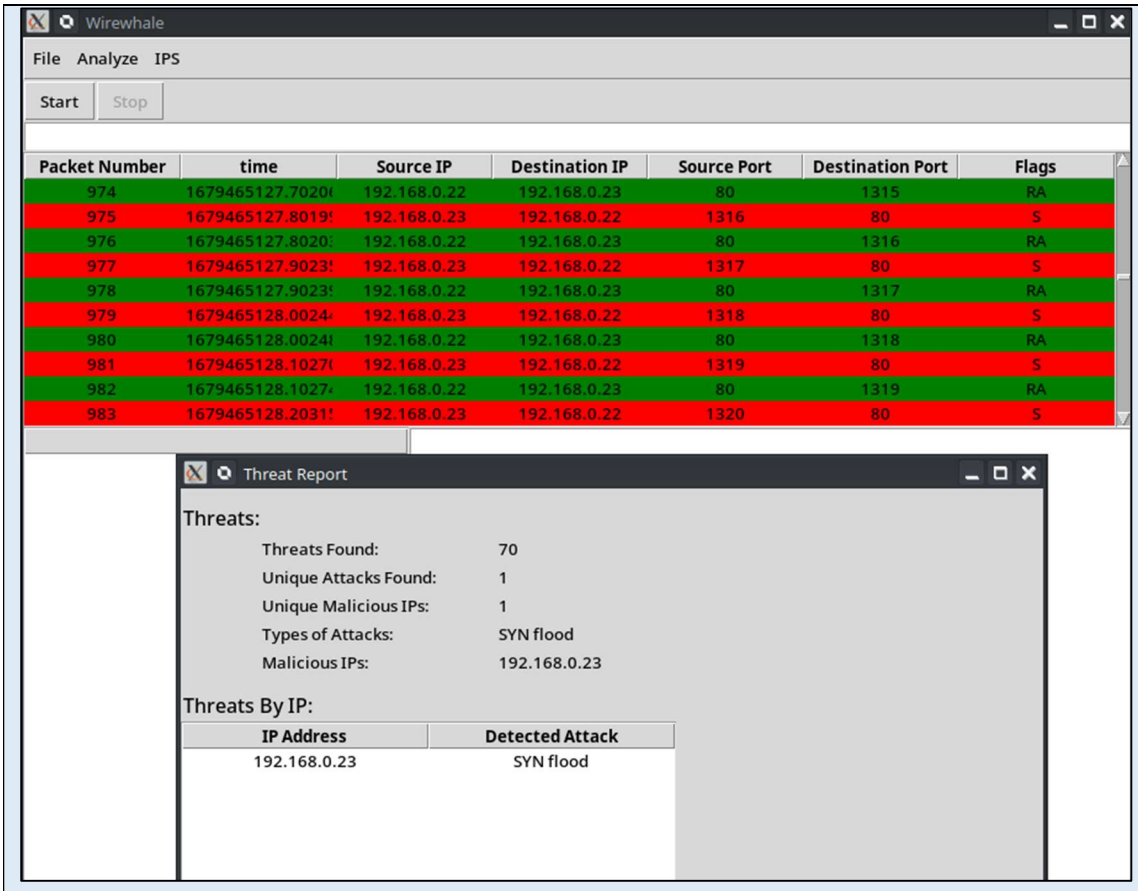
Discrepancy near 5 second mark is noted in the packet capture. The application displays the proper window size for this time, while Wireshark's graph doesn't account for this increase in window size. This could be due to Wireshark not checking every packet's window size and instead, providing a more generalized view of the window size. Wireshark also doesn't display drop in window size at end from FIN packet.

12	Sample packet capture loaded into both Wireshark and the application and throughput graph selected	Both graphs display identical points	Both graphs display some variations (explanation below)

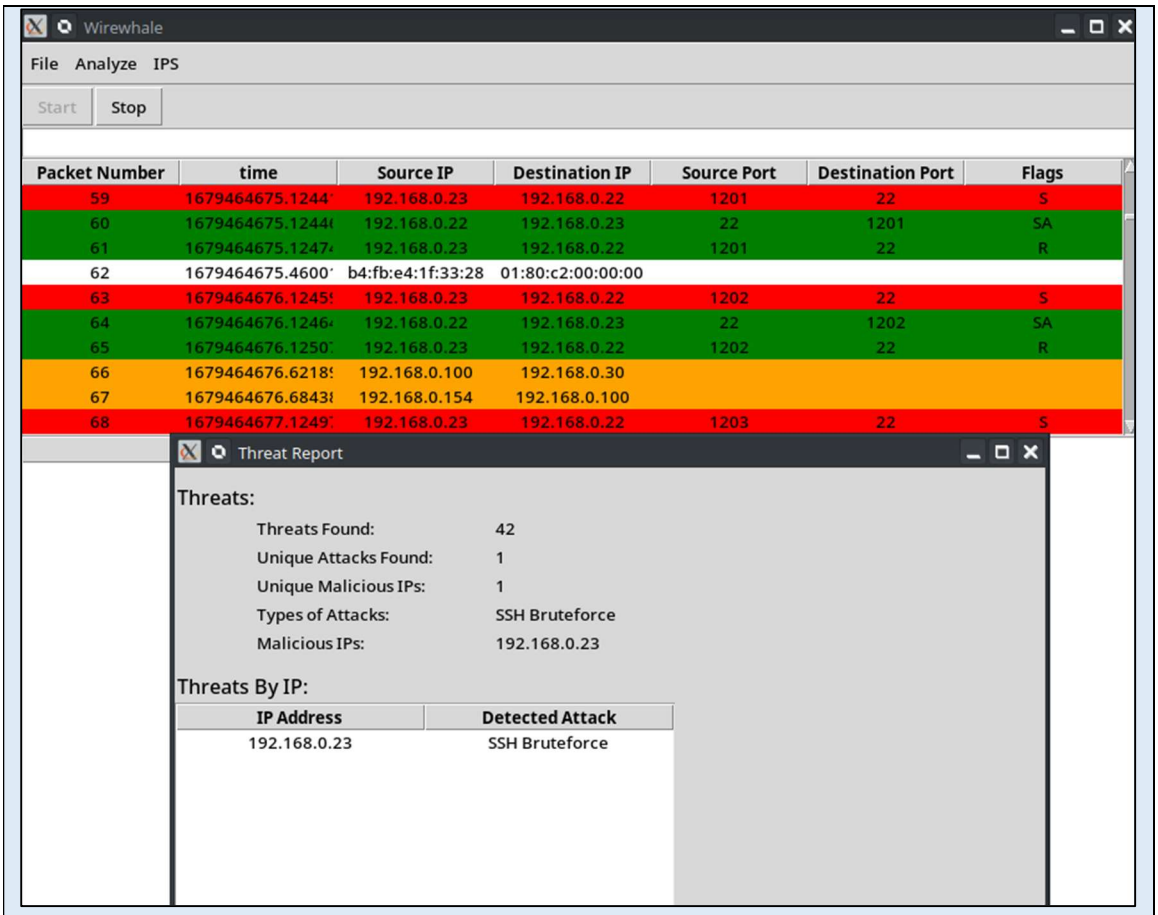


Discrepancy in graphs due to the fact that Wireshark does not connect its points on the throughput graph.

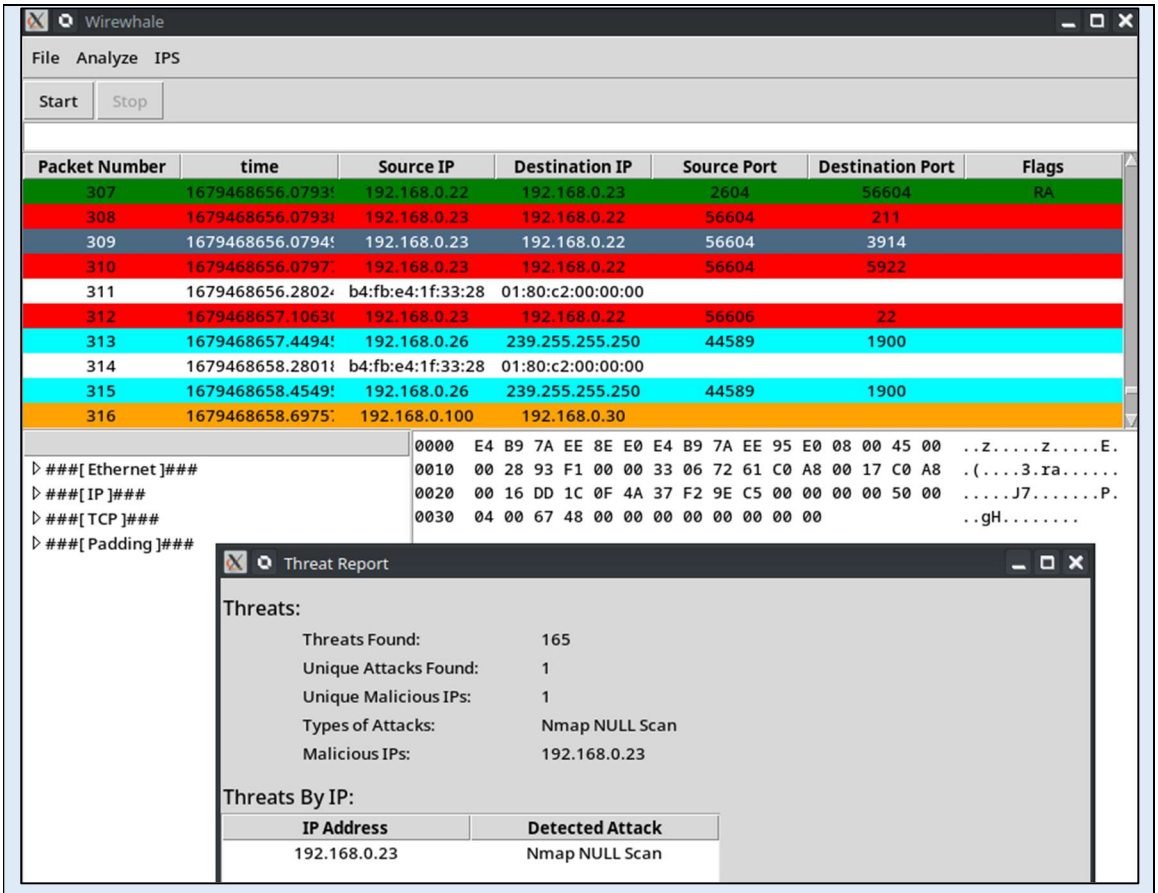
13	Hping3 used to conduct a SYN flood against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



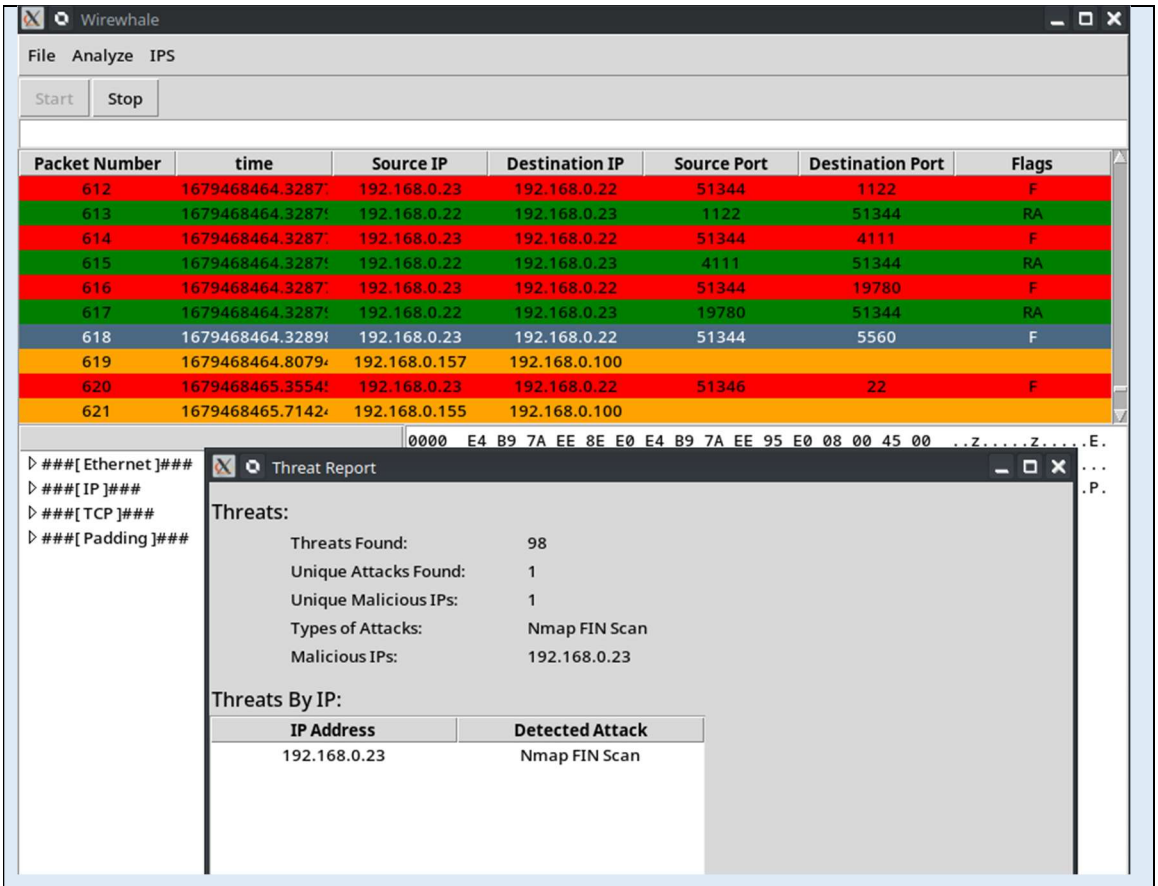
14	Hping3 used to simulate an SSH brute force password cracking against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



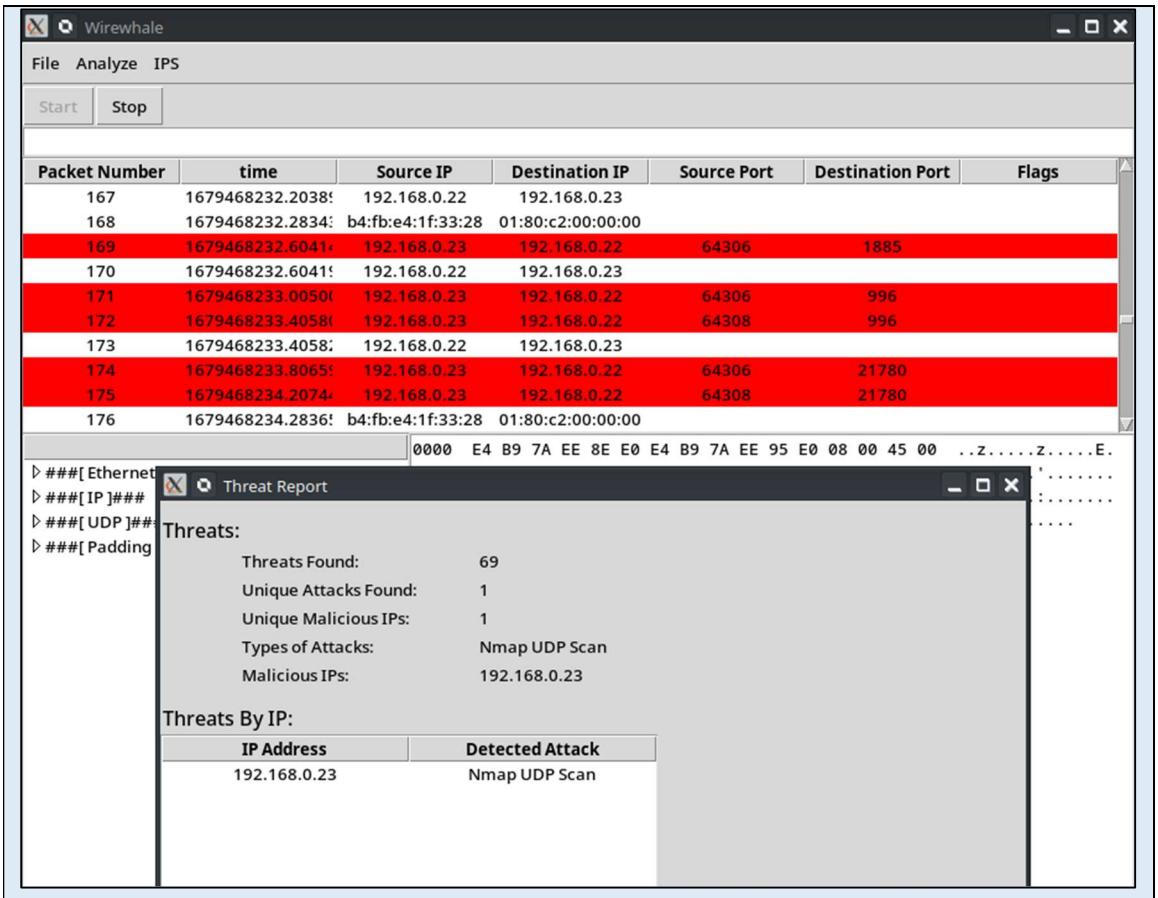
15	Nmap used to conduct a NULL scan against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



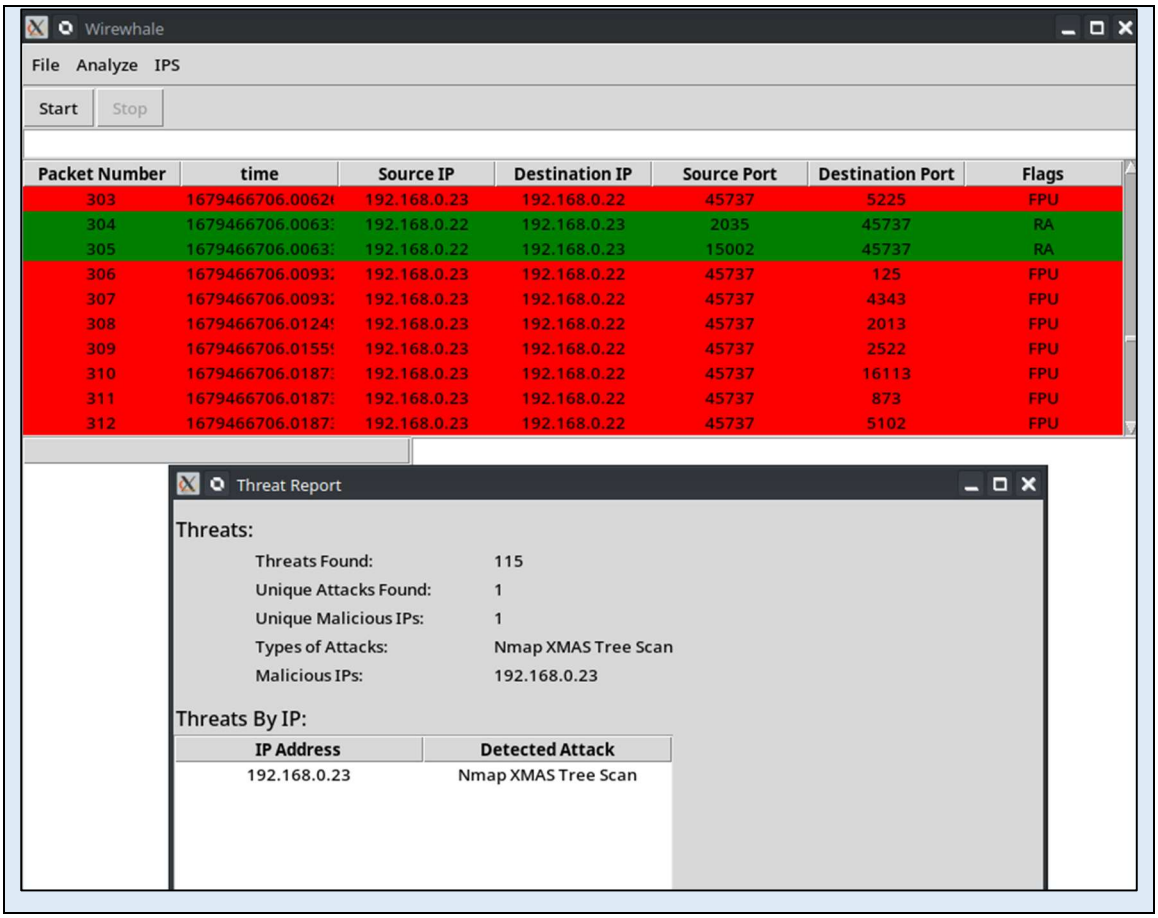
16	Nmap used to conduct a FIN scan against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



15	Nmap used to conduct a UDP scan against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



16	Nmap used to conduct a XMAS scan against machine running application	Malicious packets highlighted and reported	Malicious packets highlighted and reported



2.5.10. Stress Testing

Stress testing was done on the application to ascertain the applications limits and provoke improvement. All stress test results were compared to Wireshark for evaluation.

Test No.	Task	Time taken Wireshark	Time taken Project
1	Load a packet capture of size 1,000	0.05 sec	0.96 sec
2	Load a packet capture of size 10,000	0.5 sec	5.83 sec
3	Load a packet capture of size 100,000	2.03 sec	4 min 58 sec
4	Filtering 100,000 packets for UDP	2.05 sec	7.27 sec
5	Filtering 100,000 packets for UDP and source port 1853	2.12 sec	4.17 sec

While the difference in loading and filtering times listed above are disappointing, they are understandable as Wireshark was written using C. However, the differences in time are clearly not completely due to variances in the performance of the programming languages used. The

application keeps all packets captured in memory, which is likely the primary cause of the gap in loading performance times.

Filtering times, -while still higher than Wireshark's- did see improvement when adding additional conditions to previous filters. Wireshark's filtering times were static as it is likely applying the filter to the entire packet capture instead of only the displayed packets.

2.6. Implications of Implementation

An Agile methodology was used in the development of this application. The project was broken into sub-components based on portions of the GUI. Throughout the project, Trello was used to manage the development of these components. The application itself was implemented in two portions: the GUI, and the IDS.

GUI: The GUI portion was delivered using the Tkinter library. Scapy was used to sniff packets from the network and to read packet capture files. Matplotlib was used to embed graphs within the GUI.

IDS: Rules were created and used by Snort 2.9.20 for the IDS. Iptables was used to block connections while PsUtils were used to provide users the option to kill existing connections.

2.7. Innovation

The application is innovative in that it combines the protection provided by IDS/IPS with the analysis tools of a network protocol analyzer. Traditional network protocol analyzers lack the ability to alert users of malicious activity, while IDS/IPS do not offer significant analysis tools. Additionally, the application innovates by introducing quality-of-life improvements to enhance the packet capture navigation and incorporate some IPS-like features, such as killing conversations or blocking IP addresses. The primary innovations from this project are the detection of suspected malicious activity, generation of a report, and providing the user with mitigation options while still allowing for network traffic analysis.

2.8. Complexity

Creating this project was challenging due to the complexity involved in integrating the summarization, filtering, and graphing features of a network protocol analyzer with an IDS/IPS. While IDS/IPS libraries are widely available, the integration with a network protocol analyzer and optimization of the performance add to the complexity. To develop this application successfully, in-

depth knowledge of protocols, packets, and networks was essential, as creating effective IDS rules and network protocol analyzers depend heavily on this knowledge.

Further, this project would be inappropriate for a diploma level student since knowledge from the Network Security Applications Development program was extensively used. This project requires in-depth knowledge of packet structure, intrusion detection systems, and a familiarity with penetration testing tools such as Nmap.

One of the technical challenges in this project involved implementing a filtering language for the users. Prior to this project, I had no experience with parsing the logic of user-defined filters. Furthermore, crafting algorithms to efficiently execute these filters over large datasets presented another significant obstacle.

2.9. Research in New Technologies

The innovation for this project is derived from the creative combination of existing IDS/IPS features and network protocol analysis tools. It is noteworthy that no new technologies were introduced to implement the application. However, I did consider several new technologies. For example, machine learning was briefly examined as a replacement for Snort in identifying malicious activity within packet captures. Unfortunately, due to the lack of experience with this technology and the potential risk of introducing uncertainty into the project, it was ultimately not pursued. In addition, PyQt6 was tested as a replacement for Tkinter as the GUI library. However, early difficulties with QT6 prevented further usage, and Tkinter was used instead.

2.10. Future Enhancements

The list below details future enhancement that could be applied to the project.

- Filtering can be improved to incorporate short-circuit evaluation.
- The memory issues can be resolved by appending and reading from a file of packets instead of keeping every packet in memory.
- Scapy could be replaced by PyShark or TCPdump to grab packets at a faster rate from the interface.
- The handling of the IDS could be improved by having Snort send alerts directly to the application instead of reading from the alert file.
- Filtering can also be improved by utilizing PyParsing or PLY libraries.

- Additional filter terms could be added to improve UX.
- Searching the packet list for specific packets could be done using a binary search algorithm to improve speed.
- Cross platform implementations could increase user base.
- Additional Snort rules.
- Replace Snort's rule-based detection with a machine learning substitute.

2.11. Timeline and Milestones

This section details the development schedule and milestones of the Intrusion Detection and Network Protocol Analysis application. It took approximately 368 hours to complete the project and report. A large portion the time was spent researching and re-working code. As expected, milestone 1 took the longest to complete, however it also went over its allocated time. This was due to a poor initial design, as I had originally intended for the GUI component to be a single class. After working with GUI as a single class for weeks I realized that it needed to be split up into different components as it became difficult to work with. Once the GUI was split into different components, I was able to make consistent progress in completing and testing features. Another issue that slowed development during this phase was the discovery that Scapy will drop packets if they arrive too fast. As mentioned previously, alternative methods of sniffing were investigated at this time. The memory issue also became apparent during this time leading to even further re-working.

Milestone 2 suffered as it took longer than anticipated to complete the GUI. However, by the end of this phase the application could detect and highlight malicious activity within packet captures.

Milestone	Time Frame	Breakdown		Total Hours
1: Basic Protocol Analyzer GUI	Jan 02 – Mar 06	Wireframing	4 hours	
		Basic GUI	55 hours	
		Packet sniffing	11 hours	
		Graphing	23 hours	
		Filter research	42 hours	
		Filter implementation	24 hours	
		Packet summarization	12 hours	
		Saving and loading	2 hours	
		Stress testing	10 hours	209 hours

		Auto scrolling	4 hours	
		Unit testing/ bug fixing	22 hours	
2: Basic IDS/IPS Integration	Mar 06 – Mar 21	Scanning for Malicious packets	12 hours	
		Snort configuration	4 hours	
		Snort rules research	10 hours	
		Snort rules	7 hours	
		Killing connections	5 hours	
		Blocking IPS	4 hours	
		Unit testing/ bug fixing	10 hours	52 hours
3: Final Polish	Mar 10 – Mar 22	Marking packets	5 hours	
		Generating reports	15 hours	
		Improving GUI	23 hours	
		Unit testing/ bug fixing	17 hours	
		Color coding	4 hours	64 hours
4: Report	Mar 10 – Mar 22	Report	33 hours	
		Report polish	10 hours	43 hours
			Total:	**368 Hours**

3. Conclusion

In conclusion, the experience of combining a network protocol analyzer and an IDS has proven to be a valuable learning experience. This integration allowed for a convenient analysis of network traffic, while also enabling identification and response to malicious activity. The application developed for this project successfully leveraged the strengths of both technologies to create a tool for network security. Through the integration of existing features, the application was able to achieve innovation and improve the overall effectiveness of the security system. I believe, the combination of a network protocol analyzer and an IDS represents a promising option for enhancing network security, and further exploration and development of this integration could provide even greater benefits in the future.

3.5. Lessons Learned

While developing this application I had a chance to further my knowledge of the staggering number of protocols and packet structures. This highlighted the importance of in-depth background research

on topics surrounding future projects. Further, I learned the need for examination into options for alternative libraries. Finally, I learned the importance of identifying early-on potential memory issues and proper stress testing.

3.6. Closing Remarks

Although the Intrusion Detection and Network Protocol Analysis application lacked in terms of performance, it was a valuable learning experience and a great addition to my portfolio of applications.

4. Appendix

4.5. Approved Proposal

Table of Contents

3. Student Background

I am a student of the Network Security Applications Development program at BCIT. Prior to this I graduated with distinction from the Information Systems option of the Computer Systems Technology program. This project will leverage these programs to create a robust and complete application.

3.1. Education

British Columbia Institute of Technology 2019 - 2021

Computer Systems Technology – Graduated with Distinction

Information Systems Option – Diploma

GPA: 83%

British Columbia Institute of Technology 2021 - 2023

Network Security Applications Development – Graduating 2023

Bachelor of Technology

GPA: 82%

4. Project Description

The goal of the project is to combine a network protocol analyzer with an intrusion detection and prevention systems (IDS/IPS). This will be achieved by scanning any packet capture analyzed for malicious activity. Once malicious activity is detected the relevant packets will be highlighted and labeled, the user will then be presented options to block or timeout the offending IP address. This splits the project into two primary functions, analyzing packet captures, and detecting/reacting to malicious activity. The network protocol analyzer component of the project implements quality-of-life improvements. Specific improvements to existing network protocol analyzers include:

- applying filtering to the previous results, when appropriate,
- implementing auto-scrolling,
- allowing users to kill ongoing conversations,
- and keyboard shortcuts for scrolling.

The project will maintain features from existing network protocol analyzers such as graphing, statistics gathering, filtering, and a graphic user interface.

5. Problem Statement and Background

A network protocol analyzer monitors packets going to and from a network to offers statistics and visual representations often used for troubleshooting or monitoring purposes. An IDS works to scan packets and apply them to preset rules to identify malicious activity. An IPS goes one step further and acts against any detected threats often by blocking the offending IP address. For the purposes of this project "malicious activity" is defined as a suspicious connection made over a network. Attacks such as denial-of- service, brute-force password guessing, and port scanning activity are examples of malicious activity.

Existing network protocol analyzers lack the ability to detect malicious activity, while existing IDS/IPS's typically lack graphing capabilities and graphical user interfaces.

The foremost network protocol analyzer with a GUI -Wireshark- has shortcomings when it comes to navigating a packet capture. Applying filters to large packet captures takes a significant amount of time. This is unavoidable but, Wireshark takes the same amount of time to filter even when adding an "and" condition to a previous filter. This indicates that Wireshark applies filters to the entire packet capture when it could instead apply it to the previous result. This makes filtering a tedious process that inadvertently encourages scrolling to navigate the packet capture. Furthermore, Wireshark does not have an auto-scroll feature or any key-combinations that significantly affect the speed of navigating a packet capture which leads to a painful process of manually scrolling.

6. Scope and Depth

The minimum scope of the project is to combine a network protocol analyzer with an IDS/IPS for Linux operating systems. The application will:

- Generate reports of malicious activity
- Capture traffic from an interface
- Generate accurate graphs
    - Time sequence
    - Round trip time
    - Window scaling
    - Throughput
- Display a list of packets from packet captures
- Display a summary of packets

- Detect malicious activity including but not limited to:
    - SSH password guessing
    - Scanning activity
    - Denial of service attacks
    - Buffer overflow attacks
- Allow connections to be killed
- Block specific IP addresses
- Allow for auto-scrolling
- Have a graphical user interface
- Have filtering options for packets
- Allow packets to be marked
- Have fast filtering when adding “and” conditional to previous filter
- Have keyboard shortcuts for scrolling

The following is out of scope for the project:

- Cross-platform support
- Mobile application
- Wireshark plugin (due to inexperience with creating plugins)
- Automatic generation of Snort rules

7. Test Plan

Unit Testing

The application will use python’s unittest framework to do unit testing. All methods will utilize unit testing to verify their results. Unit testing is to be created concurrently as the project is being coded and not left to the end. The unit tests should also be run after every major update to the project.

Stress Testing

Stress testing will be used primarily to ensure filtering of packets will be done in a timely manner. Additionally, large packet captures should be viewable without issue. This will be done through timing

how quickly large packet captures are loaded, filtered, and scanned when compared to other implementations such as Wireshark.

Manual Testing

The manual testing for this project will be debugging, and bug fixing without the use of tools outside of the IDE's built-in debugger. This will involve running the application and manually verifying that everything in scope is implemented as expected.

Test Cases

These are examples of manual tests that will be conducted throughout the project.

Test Description	How Test Will Be Conducted	Passing Criteria
Open a packet capture and ensure that all packets expected are visible	A sample packet capture will be loaded into the project and checked visually to ensure all packets are displayed.	All packets within capture visible
Generate a packet capture and ensure that all packets are captured	The project's packet capture and tcpdump will be started simultaneously. The contents of their respective packet captures will then be checked to ensure they are identical.	All packets sent to and from machine visible within GUI
Marked packets visually distinct and filterable	Packet will be marked within the project and visually checked to ensure they are highlighted and filterable.	Marked packets become visually distinct and show up when filtering by marked packets
Filter packets by various headers and ensure that the output is correctly filtered	Filters such as "tcp" and "ip.addr == "192.168.0.1" will be entered into the project's filter bar before visually confirming that only relevant packets are being displayed.	Only expected packets remain visible within the GUI

Ensure blocked IP addresses are blocked	After blocking an IP address through the project, SSH will be used from the blocked IP address to attempt to connect to the machine. The SSH attempt should be ignored.	Packets sent from the offending IP address are ignored
Ensure auto-scrolling works to quickly navigate packet captures	Opening a large packet capture, middle-clicking, and dragging the mouse up/down to navigate the packet capture.	Packets are navigable with moderate speed by middle clicking
Ensure round trip time graph show accurate information	A packet capture will be loaded into the project and into Wireshark. Select round trip time graph for both and visually confirm that both graphs are identical.	Graph displays expected information
Ensure time sequence graph show accurate information	A packet capture will be loaded into the project and into Wireshark. Select time sequence graph for both and visually confirm that both graphs are identical.	Graph displays expected information
Ensure window scaling graph show accurate information	A packet capture will be loaded into the project and into Wireshark. Select window scaling graph for both and visually confirm that both graphs are identical.	Graph displays expected information
Ensure throughput graph show accurate information	A packet capture will be loaded into the project and into Wireshark. Select throughput graph for both and visually confirm that both graphs are identical.	Graph displays expected information

Ensure connections that have not yet ended can be killed	Start the project's packet capture. SSH from a separate machine into the machine running the project. Select one of the SSH connection's packets within the project and select end connection. Visually confirm the SSH connection is unusable.	Appropriate packet sent to both parties finishing connection
Ensure denial of service attack is detected	Hping3 or similar tool will be used on a separate machine. The tool will begin a SYN flood targeting the machine running the project. The offending packets should then be highlighted, and the user should be prompted to block the IP.	Application detects and warns user of specific malicious activity within packet capture
Ensure SSH password guessing is detected	A password cracker such as Hydra will be run on a separate machine targeting the machine running the project's SSH service. The multiple SSH connections should become highlighted, and a pop-up should ask the user if the offending IP should be blocked.	Application detects and warns user of specific malicious activity within packet capture
Ensure port scanning activity is detected	Nmap will be run on the network from a separate machine. The project should highlight the offending packets and prompt the user to block the IP.	Application detects and warns user of specific malicious activity within packet capture
Ensure packet summary is functional and accurate	A packet capture will be loaded into the project and into Wireshark. Packet summary will be	Packet summary displayed with accurate data pertaining to packet capture

	selected for both programs. Visually confirm that statistics are identical.	

8. Methodology

Methodology and Approach

The methodology to be used for this project will be the agile scrum methodology with two-week sprints. The sprints will focus on building out the most important features and creating usable prototypes. This will help to reduce the risk of discovering a portion of the application is not possible to code late into the project. Each prototype is to be tested using a complete set of unit tests and manual testing prior to being considered complete.

Technologies Used

The project will be managed using:

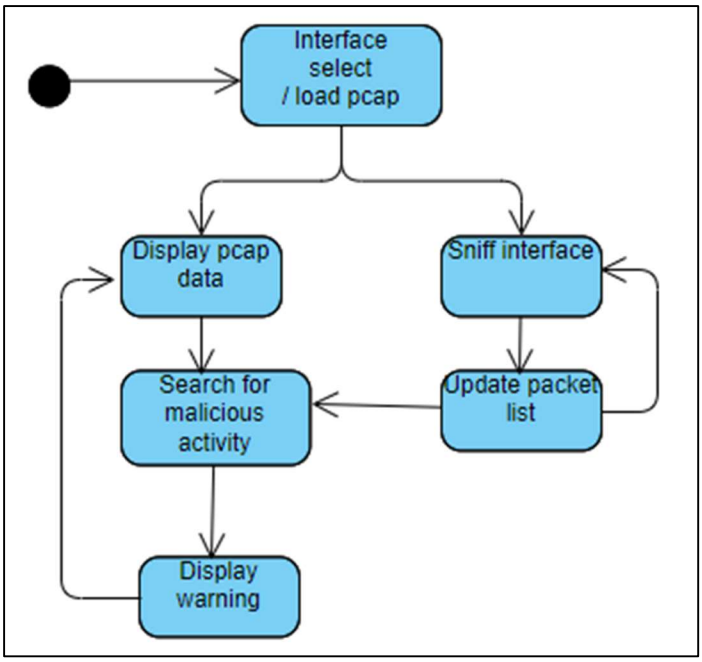
- Github for version control
- Trello for tracking progress
- MS Word for documentation
- MS Excel for test results

9. System/Software Architecture Diagram

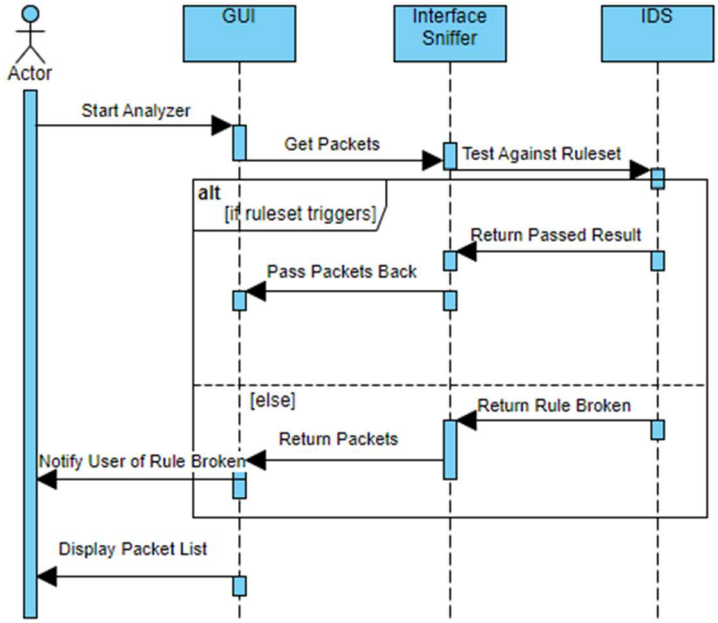
The system will consist of two portions, the IDS/IPS and the network protocol analyzer. If a packet capture is provided, it will be scanned for malicious activity while the user is able to graph and filter packets within the packet capture. If the network protocol analyzer is actively sniffing the network, then the scan for malicious activity will be run intermittently. If any malicious activity is detected, the user will be prompted to block offending IP addresses.

The network protocol analyzer will use the python Tkinter framework to display a GUI, the graphs will be embedded into the GUI using the matplotlib library. Incoming packets will be sniffed using Scapy and a robust set of Snort rules will be utilized to detect malicious activity. Malicious activity will be blocked using the iptables utility.

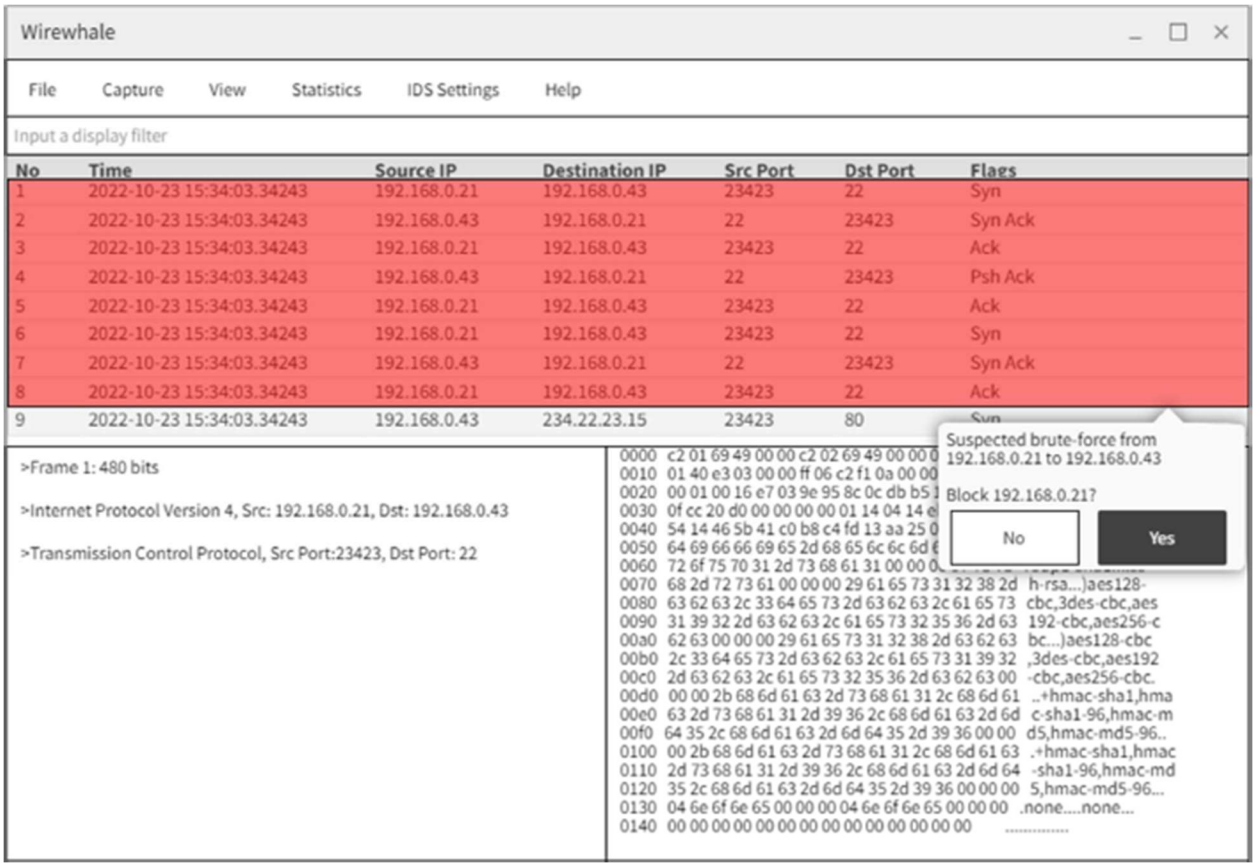
The graphic below shows a state machine diagram of the primary functionality of the application.



The sequence diagram below illustrates the primary functionality of the project. Upon starting the analyzer, packets will be retrieved and checked against the IDS. If malicious activity is detected, the user will be notified, and relevant packets will be highlighted.



The UI mockup below shows the intended end-product detecting an attacker attempting to brute force an SSH password. The relevant packets are highlighted, and the user is given the option to block the offending IP address. If yes is selected, an iptables rule will block the offending IP address.



10. Innovation

This project aims to innovate by combining the protection provided by IDS/IPS's with the analysis tools of a network protocol analyzer. Existing network protocol analyzers do nothing to alert users of malicious activity while IDS/IPS's do not have significant analysis tools. Further innovation comes from the quality-of-life improvements that will be made to the network protocol analysis portion of the project. The quality-of-life improvements primarily focus on improving the navigation of a packet capture, but also allow for some IPS like tools such as killing conversations or blocking IP addresses.

Research into fingerprints of malicious activity will be required to make a robust IDS. I have previously made basic IDS rules but do not yet know how to detect a wider array of malicious activity. A good starting point into this research would be to investigate existing IDS rules or through simulating and analyzing malicious activity.

Existing network protocol analyzers such as Wireshark and Ettercap don't have built in IDS/IPS protection (Porup, 2018). Packet captures analyzed using these tools will not detect malicious activity. This is the primary innovation intended for this project. Packet captures analyzed using the project will generate a report of suspected malicious activity and will prompt the user with options to mitigate the threat while still allowing for analysis of network traffic.

11. Complexity

The project is complex because of the difficulty of recreating the numerous tools and features of a network protocol analyzer such as Wireshark and combining them with an IDS/IPS. These features include:

- a graphical user interface,
- graphing capabilities (time sequence, round trip time, etc.)
- filtering language,
- and following of conversations.

Additional complexity comes from implementing the features of an IDS/IPS and combining them with the network protocol analyzer. While IDS/IPS's are both well-known and already have multiple libraries available, combining it with a network protocol analyzer and ensuring it runs efficiently introduces additional difficulty. The IDS/IPS will also need to have configurable settings to ensure it applies rules appropriate to its environment. For instance, a packet capture from a large web-based business should have a higher threshold for incoming requests when compared to a typical home network.

My education on protocols, packets, and networks will be crucial in creating this application as creating either an IDS or a network protocol analyzer both heavily rely on this knowledge.

12. Technical Challenges

One of the technical challenges comes from having to implement a filtering language for the users. I have not previously created any projects that have required parsing the logic of user given filters. Additionally, crafting algorithms to efficiently execute these filters over sizable datasets poses a unique challenge.

Another challenge is going to be identifying malicious activity within packet captures. While I do have previous experiences creating IPS's, the types of malicious activity I know how to detect currently does not match the scope of what I plan to implement. Research into existing solutions will be required to complete this project.

13. Development Schedule and Milestones

Task #	Task	Duration
1	Basic GUI	20 hours
2	Packet sniffer	10 hours
3	Auto scrolling / navigation improvements	20 hours
4	Graphs	20 hours
5	Filtering language research	30 hours
6	Filtering language implementation	20 hours
7	Saving and loading packet captures	10 hours
8	Stress testing and improve if needed	20 hours
9	Summarize packet captures	20 hours
Milestone 1: Basic protocol analyzer completed	**Total time required: 170 hours**	
11	IDS/IPS scanning packets	5 hours
12	IDS/IPS rules research	30 hours
13	IDS/IPS rules implementation	20 hours
14	Killing ongoing connections	5 hours
15	Blocking IP's	5 hours
Milestone 2: Basic IDS/IPS integrated	**Total time required: 65 hours**	
16	Marking packets	5 hours
17	Generate reports from IDS/IPS rule results	10 hours
18	Improved GUI	20 hours
19	Color coding packets	10 hours
Milestone 3: Final polish completed	**Total time required: 55 hours**	
20	Report	20 hours

21	Polish report	10 hours
22	Unit testing and bug fixing (to be done throughout entire project)	50 hours
Milestone 4: Project completed	**Total time required: 70 hours**	
	Total:	360 hours

14. Deliverables

The deliverables for this project are as follows:

4. **Network Protocol Analyzer:** Tested and fully useable network protocol analyzer with graphing, GUI, summarization/collation of packet information, and filtering.
5. **IDS/IPS Integration:** An extensive list of up-to-date rules will be applied to detect malicious activity within packet captures.
6. **Report**: Comp 8047 report detailing the project.

15. Conclusion and Expertise Development

This project will develop my expertise in recognizing signs of an attack. I will need to develop my ability to recognize malicious activity in a network to not only create IDS/IPS rules but to understand and apply them appropriately to the project. By researching existing IDS/IPS rules I will be able to apply them to improving the project and develop my skills. Additional expertise development will come from developing the filtering language used to narrow down packet captures. It's my hope that this will become an accessible tool for network analysts to identify malicious activity within packet captures.

16. References

DANJOU, J. (2019, April 1). *Writing Your Own Filtering DSL in Python.* Retrieved from julien danjou info: https://julien.danjou.info/writing-your-own-filtering-dsl-in-python/

KEARY, T. (2022, June 6). *Wireshark Cheat Sheet – Commands, Captures, Filters & Shortcuts.* Retrieved from Comparitech: https://www.comparitech.com/net-admin/wireshark-cheat-sheet/

Markus, O. a. (2022, September 8). *What are the best Wireshark alternatives?* Retrieved from alternativeto: https://alternativeto.net/software/wireshark/

MattJonkman. (2013, April 19). *New ET Users Guide.* Retrieved from emergingthreats: https://doc.emergingthreats.net/bin/view/Main/NewUserGuide

Nederkoorn, C. (2022, August 9). *Top 10 Python GUI Frameworks Compared.* Retrieved from activestate: https://www.activestate.com/blog/top-10-python-gui-frameworks-compared/#:~:text=The%20Python%20GUI%20Project%2C%20aka,other%20frameworks%20on%20this%20list.

Porup, J. (2018, September 17). *What is Wireshark? What this essential troubleshooting tool does and how to use it.* Retrieved from csoonline: https://www.csoonline.com/article/3305805/what-is-wireshark-what-this-essential-troubleshooting-tool-does-and-how-to-use-it.html#:~:text=While%20Wireshark%20is%20a%20network,and%20analyze%20encrypted%20TLS%20traffic.

Python Software Foundation. (2022, October 26). *unittest — Unit testing framework.* Retrieved from python.org: https://docs.python.org/3/library/unittest.html

Taylor-Morgan, S. (2020, April 3). *The 7 most popular ways to plot data in Python.* Retrieved from opensource: https://opensource.com/article/20/4/plot-data-python

17. Change Log

V1

Feedback

1. The proposal is very general and lacks technical depth.
2. The proposal makes several mentions of malicious activity, but you have not explained what constitutes “malicious activity” and associated indicators. How will these indicators will be used in the IDS/IPS module? How will the application mitigate false negatives and false positives?
3. Will the IDS and firewall rulesets be generated automatically? Is so, explain how. The proposal mentions the design of a custom protocol analyzer, but no details have been provided. For example, which libraries do you intend to use and how?
4. The test plan is very general and incomplete. Clearly explain the individual tests you will perform to validate the rule sets and malicious activity detection components.

V2

Changes

Changes	**Reason for change**	**Section altered**
Added definition for malicious activity	Feedback from D’Arcy/ feedback 2	Problem Statement and Background
Added mock-up of UI	Feedback from D’Arcy	System/Software Architecture Diagram
Added specific technical detail describing implementation	Feedback 1, 2, 3	System/Software Architecture Diagram Project Description
Addressed alternative solutions	Feedback from D’Arcy	Scope and Depth
Added specifics to test plan and added additional tests explaining how detection of malicious activity will be performed	Feedback 1, 2, 4	Test Plan
Clarified automatic generation of IDS rules as out of scope and updated deliverables to reflect	Feedback 3	Scope and Depth Deliverables

4.6. Project Supervisor Approvals

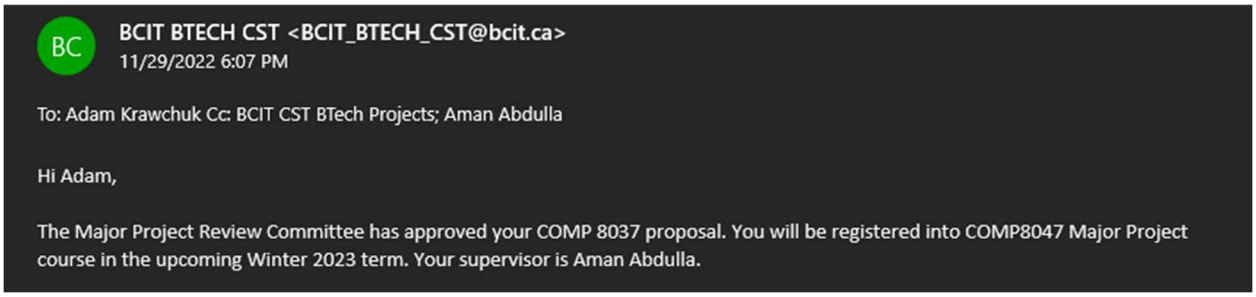


Figure 16: Proposal approval

5. References

Bukunmi, O. (2021, November 10). *Comparing the top python gui frameworks*. LogRocket Blog. Retrieved March 19, 2023, from https://blog.logrocket.com/comparing-top-python-gui-frameworks/

Getting started with Trello. Trello. (n.d.). Retrieved March 20, 2023, from https://trello.com/guide

Hping man page. HPING3(8) - linux man page. (n.d.). Retrieved March 22, 2023, from https://linux.die.net/man/8/hping3

Nmap. (n.d.). Retrieved March 23, 2023, from https://nmap.org/

Ply (python lex-yacc). (n.d.). Retrieved March 10, 2023, from https://www.dabeaz.com/ply/

Pyparsing. PyPI. (n.d.). Retrieved March 13, 2023, from https://pypi.org/project/pyparsing/

Snort documentation. Snort setup guides for emerging threats prevention. (n.d.). Retrieved March 23, 2023, from https://www.snort.org/documents#OfficialDocumentation

Tkinter - Python interface to TCL/TK. Python documentation. (n.d.). Retrieved March 22, 2023, from https://docs.python.org/3/library/tkinter.html

Welcome to Scapy's documentation! - Scapy 2.5.0 documentation. (n.d.). Retrieved March 14, 2023, from https://scapy.readthedocs.io/en/latest/

Wireshark · documentation. Wireshark. (n.d.). Retrieved March 13, 2023, from https://www.wireshark.org/docs/