**A blue red and yellow shield with lion and lion

AI-generated content may be incorrect.**

**NET-SENTINEL: A Lightweight, Rule-Based Intrusion Detection System for Raspberry Pi-based Small Networks**

**Chukwu Dominion Chidiebere**

**169066**

**CNS**

**Supervisor Name**

[**Humphrey Owuor**](mailto:howuor@strathmore.edu)

**Submitted in Partial Fulfillment of the Requirements of the Bachelor of Science in Computer Networks and Cybersecurity at the Strathmore University**

**School of Computing and Engineering Science**

**Strathmore University**

**Nairobi, Kenya**

**May 2025**

**Declaration and Approval**

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the research proposal contains no material previously published or written by another person except where due reference is made in the research proposal itself.

Student Name: Chukwu Dominion Chidiebere

Admission Number: 169066

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The Proposal of **Chukwu Dominion Chidiebere** has been reviewed and approved by **Mr. Humphrey Owuor**

Supervisor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Acknowledgement**

I would like to express my deepest gratitude to my supervisor, Mr. Humphrey Owuor, for his invaluable guidance, encouragement, and constructive feedback throughout the development of this proposal. His insights have played a vital role in shaping the focus and clarity of this project. I also extend my sincere thanks to the faculty and staff of the School of Computing and Engineering Sciences at Strathmore University for the knowledge and academic foundation they have provided. I am especially grateful to the School of Engineering for their commitment to supporting this project by providing access to Raspberry Pi hardware for testing and development, this practical resource has been critical to the feasibility of the proposed system. Lastly, I acknowledge the unwavering support of my family and friends, whose patience, motivation, and belief in my goals have sustained me throughout this academic journey.

**Abstract**

Small businesses and home networks are increasingly targeted by cyberattacks, yet most lack access to professional-grade intrusion detection systems (IDS) due to high costs and complexity. NET-SENTINEL addresses this gap by offering a lightweight, rule-based IDS that runs on affordable Raspberry Pi hardware, delivering enterprise-level protection in a user-friendly format. The system uses Scapy with Berkeley Packet Filters (BPF) to capture only suspicious traffic, employs threshold-based alerting to reduce false positives, and logs events in lightweight CSV files instead of complex databases. This approach ensures efficient resource usage keeping CPU usage below 50% and memory consumption under 500MB, while maintaining over 85% detection accuracy for common threats such as port scans, brute-force logins, and DDoS attacks. Prioritizing accessibility, NET-SENTINEL translates technical alerts into plain English, features a color-coded dashboard (green for safe, red for danger), and enables one-click CSV report generation. While currently optimized for Raspberry Pi 2, the system’s architecture is built with scalability in mind, allowing for future integration of advanced detection techniques, including machine learning, on newer models like Raspberry Pi 4 or 5. This project proves that real-time, effective cybersecurity doesn’t require expensive infrastructure or technical expertise, just smart, inclusive design focused on real-world needs. Due to the nature of the system, the methodology adopted will be the Prototyping methodology.

**Table of Contents**

[Declaration and Approval…………………………………………………………………….. ii](#_zgl836rkrbbz)

[Acknowledgement…………………………………………………………………………… iii](#_vvcn8b7li4i6)

[Abstract………………………………………………………………………………………. iv](#_lbni036lmp5h)

Table of Contents………………………………………………………………………………v

List of Figure………………………………………………………………………………….ix

List of Tables…………………………………………………………………………………..x

[List of Abbreviations………………………………………………………………………..](#_83dnwb9tyozt) xi

[Chapter 1:](#_bkguhui9uq3r) Introduction…………………………………………………………………….1

1.1 Background Information……………………………………………………………….1

1.2 Problem Statement……………………………………………………………………..2

1.3 Objectives……………………………………………………………………………...2

1.3.1 General Objectives………………………………………………………………..2

1.3.2 Specific Objectives……………………………………………………………..….2

1.4 Research Questions……………………………………………………………..………3

1.5 Justification……………………………………………………………..………………3

1.6 Scope, Delimitations and Limitations………………………………..………...……….4

1.6.1 Scope……………………………………………………….…………………..4

1.6.2 Delimitations…………………………………………………………………...4

1.6.3 Limitations……………………………………………………………………..4

[Chapter 2:](#_uau6q7gc5wf4) Literature Review…………………………………………………………....….5

2.1 Introduction……………………………………………………………………………..5

2.2 State of the art in current Scenario……………………………………………………...5

2.3 Related Works…………………………………………………………………………..6

2.4 Identified Gaps in Related Works……………………………………………….….7

2.5 Conceptual Framework……………………………………………………………...8

2.5.1 Traffic Monitoring and Packet Capture………………………………………....8

2.5.2 Detection Engine with Threshold-Based Rules………………………………..8

2.5.3 Alert and Notification Mechanism……………………………………………...9

2.5.3.1 Generates Human-Readable Alerts……………………………………….9

2.5.3.2 Triggers Visual Indicators on the Dashboard…………………………….9

2.5.4 Data Logging and Storage………………………………………………………9

2.5.5 System Performance Management……………………………………………...9

Chapter 3: [Methodology………………………………………………………………….11](#_Chapter_3:_Methodology)

3.1 Introduction……………………………………………………………………………11

3.2 Methodology…………………………………………………………………………..11

3.2.1 Planned phases of Development………………………………………………...12

3.3 Expected Deliverables…………………………………………………………………13

3.4 Tools and techniques…………………………………………………………………..14

Chapter 4: Analysis Diagram………………………………………………………………..15

4.1 Introduction……………………………………………………………………………15

4.2 Requirements…………………………………………………………………………..15

4.2.1 Functional Requirements…………………………………………………………15

4.2.2 Non-Functional Requirements……………………………………………………15

4.3 Analysis Diagram……………………………………………………………………...17

4.3.1 Use case diagram ………………………………………………………………...17

4.3.2 Sequence Diagram………………………………………………………………..18

4.3.3 Flowchart…………………………………………………………………………19

4.3.4 System Architecture………………………………………………………………20

4.4 Design Diagrams………………………………………………………………………21

4.4.1 System Architecture………………………………………………………………21

4.4.2 Network Architecture Diagram…………………………………………………..22

Chapter 5: Implementation and Testing……………………………………………………..23

5.1 Introduction…………………………………………………………………………..23

5.2 development Environment…………………………………………………………...23

5.2.1 Hardware Specifications…………………………………………………………23

5.2.2 Software Specifications………………………………………………………….23

Directory Structure…………………………………………………………………….24

5.3 System Implentation…………………………………………………………………24

5.3.1 Packet Capture and Filtering…………………………………………………….25

5.3.2 Detection Engine………………………………………………………………...26

5.3.3 Alerting and Dashboard…………………………………………………………28

5.3.4 Logging………………………………………………………………………….31

5.4 Testing Environment…………………………………………………………………32

5.5 Testing of Functional Requirements…………………………………………………33

5.6 Testing of Non-functional Requirements…………………………………………….34

5.7 Discussion of Results………………………………………………………………...34

Chapter 6: Conclusion and Recommendations……………………………………………..36

6.1 Conclusion…………………………………………………………………………...36

6.2 Recommendations……………………………………………………………………36

6.3 Future Work………………………………………………………………………….36

Biblography…………………………………………………………………………………38

Appendices………………………………………………………………………………….39

Appendix 1: Gantt Chart……………………………………………………………………39

Appendix 2: Chapter 1 Zerogpt AI Report………………………………………………….39

Appendix 3: Chapter 1 Zerogpt AI Report………………………………………………….40

Appendix 4: Chapter 1 Zerogpt AI Report…………………………………………………40

Appendix 5: Chapter 1 Zerogpt AI Report…………………………………………………41

Appendix 6: Chapter 1 Zerogpt AI Report…………………………………………………41

Appendix 7: Chapter 1 Zerogpt AI Report…………………………………………………42

Appendix 8: Turnitin Similarity Report

**List of Figures**

[Figure 2.5: Conceptual Framework](#_130u40c95n0h)

[Figure 2.2 Prototype Methodology](#_973061rqvber)

Figure 3.4.1: NET-SENTINEL Gantt chart

Figure 4.3.1: use case diagram

Figure 4.3.2: sequence diagram

Figure 4.3.3: flowchart

Figure 4.3.4: System Architecture

Figure 4.4.1 System Architecture

Figure 4.4.2 : Network Architecture Diagram

Figure 5.3.1.1: Packet\_sniffer.py detecting traffic

Figure 5.3.1.2: NetSentinel.py running

Figure 5.3.2.1: 2nd Virtual machine for flood simulation

Figure 5.3.2.2: Virtual Machine Flagging flood traffic and alerting on terminal

Figure 5.3.3.1: Dashboard running on terminal (Flask server)

Figure 5.3.3.2: Dashboard UI showing live alerts, IP status and action buttons.

Figure 5.3.3.3: Dashboard log viewer page showing all blocked and unblocked IP addresses

Figure 5.3.3.4: Dashboard log viewer page for all suspicious traffic

Figure 5.3.3.5: Dashboard accessed from smartphone

Figure 5.3.3.6: Alert email received in admin inbox

Figure 5.3.4.1: Contents of alert\_log.csv after testing simulation

Figure 5.3.4.2: block\_log.csv showing blocked and unblocked IP entries

**List of Tables**

Table 5.2.1 : Hardware Specifications

Table 5.2.2 : Software Specifications

Table 5.5.1: Functional Requirements and Testing Outcomes

Table 5.6.1: Non-Functional Requirements and Testing Outcomes

**List of Abbreviations**

IDS - Intrusion Detection System

BPF - Berkeley Packet Filter

CSV - Comma Separated Values

UI - User Interface

DDoS - Distributed Denial of Service

RAM - Random Access Memory

CPU - Central Processing Unit

IoT - Internet of Things

ML - Machine Learning

SQL - Structured Query Language

HTTP – Hyper Text Transfer Protocol

HTTPS – Hyper Text Transfer Protocol Secure

# Introduction

**1.1 Background Information**

Now, even home workers and small businesses are under threat from major cybersecurity threats. The twist is, large corporations can afford to fit high-level security, but smaller businesses have to make do with ultra-low-level protection, or worse, with no protection at all. That leaves them vulnerable. In truth, the 2024 Verizon Data Breach Report finds that roughly 43% of cyberattacks are aimed at small organizations. Not because they are the biggest targets, but because they're typically the easiest.

Products such as Snort and Suricata are very good at identifying threats but have a learning curve. They usually need robust hardware and a reasonable amount of technical know-how to get going. Even the free versions can be tough to figure out. There are simpler, cloud-based tools out there, but their subscription fees tend to put them out of reach for most small operations.

Some recent studies have introduced lightweight detection methods. For example, Khan et al. (2023) proposed techniques that work efficiently on small devices like ARM boards. Liu and Zhang (2022) also found that using threshold-based methods helps cut down on false alarms. But even with these promising results, most of the ideas are still stuck in research papers there aren’t many tools that regular people can actually use yet.

NET-SENTINEL was created to change that. It brings together proven detection techniques and makes them simple enough for anyone to use. Designed originally for Raspberry Pi but tested and proven on virtual machines and Windows hosts, it monitors traffic, flags suspicious behavior, and lets the user block threats through a clear, color-coded dashboard. Alerts are easy to understand and actions can be taken without needing any security training.

More than just a project, NET-SENTINEL represents an effort to bridge the cybersecurity gap. It allows local shops, home offices, and schools to defend themselves without spending a fortune or hiring experts. In a world where everyone is connected, protecting small networks makes the whole internet a little safer. NET-SENTINEL proves that good cybersecurity doesn’t have to be complicated or expensive, it just needs to be smart and accessible.

**1.2 Problem Statement**

Small networks are being left behind in cybersecurity. While big organizations use sophisticated intrusion detection systems, the solutions available to smaller setups are either too complex, too expensive, or simply don’t work in limited-resource environments. This isn’t just a theoretical issue, it’s something that puts real people at risk of losing money, having their personal data stolen, or even falling victim to identity theft.

While researchers have made strides in building lighter, more efficient detection systems, most of those ideas haven’t made their way into simple, usable tools that everyday people can actually set up and trust. The need for such a solution is growing rapidly, especially as more people rely on small networks for remote work, online business, and personal data exchange

This project is an attempt to move from theory to practice. It combines lightweight detection ideas with an easy-to-use interface to create a tool that works in the real world. Today, with tools like Scapy and the growth of platforms like Raspberry Pi and virtual machines, it's finally possible to build reliable security systems without spending a fortune or needing deep technical skills.

**1.3 Objectives**

**1.3.1.General**  
 This main aim of this study is to develop a lightweight intrusion detection system that provides effective security for small networks without requiring technical expertise or expensive infrastructure.

**1.3.2.Specific Objectives:**

1. To create a lightweight packet capturing module using Scapy and BPF filters that can efficiently monitor essential network ports like 22, 80, and 443, all while keeping CPU usage low and performance stable.
2. To design a simple but effective detection engine that uses threshold rules to spot suspicious activities such as DDoS attacks or repeated failed login attempts with a target of achieving at least 85% accuracy during tests.
3. To develop a simple web-based dashboard that displays alerts in plain language with clear visual indicators, tested with non-technical users.
4. To evaluate how flat-file logging using CSV compares to database storage in terms of memory efficiency and speed of retrieval.
5. To assess overall system performance, including CPU, memory, and network usage during different attack simulations using virtualized environments.

**1.4 Research Questions**

1. How can Scapy and BPF be utilized to successfully capture only relevant packets on low-resource systems?
2. How traffic levels are attacks like DDoS or brute-force attempts can be identified without creating too many false positives?
3. Do clear language and simple visual cues (e.g., color codes) in alerts actually help facilitate the non-technical user to better understand and respond appropriately to threats?
4. How does CSV logging perform compared to database logging under constrained conditions?
5. Can the IDS system sustain efficient operation under real-time attack scenarios in a virtualized network?

**1.5 Justification**

Most intrusion detection systems are simply not practical for everyday users. They are too expensive, too complicated, and need hardware that small setups can’t afford. Ironically, these are exactly the networks that are attacked most. That's why building security tools that are lightweight, powerful, and accessible to anyone isn't just useful, it's downright crucial.

Studies show that simple threshold-based systems can perform well when properly implemented. CSV-based logging has also proven efficient for limited-resource environments. NET-SENTINEL takes these concepts and transforms them into a usable product, one that works not just in theory, but in practice.

By packaging known techniques into a lightweight, testable, and friendly interface, this project addresses a real need and demonstrates that small networks don’t have to be defenseless.

**1.6 Scope and Delimitations**

**1.6.1 Scope**

The goal of this project is to develop and design NET-SENTINEL, a simple and lightweight intrusion detection system that will keep small offices and home networks safe. While the system is meant to be capable of running efficiently even on low-powered hardware like the Raspberry Pi, development and testing are going to be conducted first on virtual machines and regular PCs. The focus will be on detecting common network threats like brute-force logins and DDoS attacks, and alerting them on an easy-to-use and straightforward dashboard. The goal is to make something that works, does not require extensive technical knowledge, and can offer strong protection to the average user.

**1.6.2 Delimitations**

The project specifically rules out advanced security features such as deep packet inspection or encryption analysis due to performance limitations. Neither does it deal with large business networks or integrate with cloud-based control systems. Mobile apps and automatic updates are outside the scope currently. The focus is solely on delivering a working, standalone IDS that is usable by general users.

**1.6.3 Limitations**

Since the system will mostly be tested on a virtual machine setup rather than a real network, there could be slight differences in how it works. It's set up to work well under low-resource conditions, but extremely high traffic could potentially impact its performance. And while it does an excellent job of catching typical threats, some newer or more advanced attacks might slip through. Nonetheless, the design choices like the use of simple detection methods and avoiding extensive processing are made to ensure affordability and simplicity, especially for those who need strong security but do not have access to expensive tools.

# Chapter 2: Literature Review

**2.1 Introduction**

This chapter discusses the state of intrusion detection systems (IDS) today, with an emphasis on solutions applicable to low-resource environments such as small businesses and home networks. The chapter begins with a discussion of the state of IDS technology at present and how it falls short in resource-poor contexts. It continues by reviewing pertinent academic literature, determining methodological orientations and strengths and weaknesses. The chapter concludes by pointing out some of the gaps in literature that the NET-SENTINEL project will address.

**2.2 State of the Art in the Current Scenario**

Intrusion Detection Systems (IDS) are the secret to the integrity of today's networks, and they provide security against malicious access, malware attacks, and other cyber threats. Solutions like Snort and Suricata are widely employed in corporate networks due to their strong performance and dependability. IDS employs signature-based detection and anomaly-based detection techniques to categorize malicious activity. However, the price of having such a capability is their reliance on high-performance hardware and the need for complex configurations. As Sharma et al. (2021) explain, the generic resource profile of such platforms makes them not desirable for small networks, as those found in small businesses or home office.

In the background of growing occurrences of Internet of Things (IoT) devices and the need for scalable security, the research has inclined towards lightweight IDS models. Light IDS models are specifically designed for resource-constrained environments and can be installed based on ARM-based systems like Raspberry Pi. Alshamrani et al. (2019), for example, demonstrated the efficacy of a Raspberry Pi-based IDS on correlation-based feature selection and machine learning algorithm. Their approach managed to achieve high accuracy with low CPU and memory usage, proving that secure systems do not have to execute on enterprise-grade hardware.

Parallel to this, threshold-based detection mechanisms have gained attention as efficient alternatives to deep packet inspection. These systems define behavioural limits (e.g., maximum failed logins per minute) and trigger alerts when those thresholds are crossed. Laszka et al. (2014) argue that, while this model reduces false positives compared to static signatures, careful tuning is essential to avoid under- or over-triggering alerts.

Another significant innovation in this space is efficient logging. Traditional IDS platforms log data in relational databases, which can be heavy for low-resource devices. In contrast, CSV-based flat-file logging offers a streamlined alternative that retains critical information while consuming significantly less memory and processing power. BitLyft (2023) notes that flat-file logging reduces system overhead and improves log retrieval performance in lightweight environments.

Finally, a key limitation in traditional IDS is their lack of user-friendly interfaces. Most alerts and configurations are highly technical, posing a barrier to adoption for non-expert users. Research by Lee et al. (2022) reveals that systems incorporating plain-language alerts and color-coded dashboards increase user responsiveness, particularly among those with minimal cybersecurity training. This shift toward usability is particularly relevant for community and personal deployments, where system administrators may not have formal IT backgrounds.

**2.3 Related Works**

Several notable works have explored how IDS can be adapted for lightweight, real-world deployment on hardware such as Raspberry Pi. Alshamrani et al. (2019) present one of the most comprehensive examples, with a model that uses machine learning and feature selection to minimize processing time. Their deployment on Raspberry Pi achieved a strong balance between accuracy and system load, demonstrating that IDS can be both effective and efficient on entry-level hardware.

Threshold-based IDS models have also seen promising development. Laszka et al. (2014) proposed a method that optimizes threshold values in real-time based on observed traffic behaviour. Their findings suggest that static threshold values can be too rigid, leading to either desensitized or hyper-sensitive systems. Adaptive thresholds, tailored to specific environments, offer a more dynamic and accurate detection method.

From the usability point of view, Lee et al. (2022) created an IDS with non-technical users in mind. They had simple-language notifications and simple-to-use dashboards in their system. The users were more likely to understand and respond appropriately to warnings during usability tests, which illustrates the role played by accessible design in enhancing adoption and efficiency.

Log management has also been reconsidered in correlated research. BitLyft (2023) advocated for flat-file logging as a method, identifying notable memory and CPU gains compared to traditional database-backed systems. Their results indicate that flat-file logging is suitable for small-scale IDS systems that do not need the full overhead of SQL-based storage but must have strong event tracking.

These all provide the foundation for a large-scale, real-world practical IDS that can be of use to users outside the normal enterprise space. Most are, however, confined to test platforms or prototypes, and none have been field-deployed on low-end hardware in production environments.

**2.4 Identified Gaps in Related Work**

Despite clear progress in the development of lightweight and user-focused IDS technologies, significant gaps persist. One of the most critical is the lack of integration between efficiency and usability. Most existing systems focus either on resource constraints or user interaction but rarely both. This separation limits their effectiveness in real-world small-scale networks, where devices like Raspberry Pi must deliver security to users without advanced IT skills.

Another enormous gap is their true deployment in the real world. While most have been tested in a controlled test lab, very few have actually been field-tested on real Raspberry Pi hardware or in live home or small business network environments. This lack of real-world data impedes the understanding of how these systems run under different circumstances or with actual threat data..

Furthermore, the threshold mechanisms used in many models remain static and inflexible. As Laszka et al. (2014) point out, without adaptive tuning, such systems are prone to underperforming in dynamic environments. The need for smart thresholds that can respond to changing network conditions is an important area for further exploration.

Lastly, although CSV-based flat-file logging provides a good balance of simplicity and efficiency, it introduces its own limitations particularly regarding long-term event analysis and correlation across sessions or devices. As of now, very few systems attempt to bridge the gap between flat-file simplicity and more advanced analytics.

The NET-SENTINEL project is designed to close these gaps by developing a lightweight, deployable, and user-friendly IDS tailored specifically for Raspberry Pi environments. It leverages threshold-based detection tuned for home/small office behaviour patterns, uses lightweight CSV logging to conserve resources, and presents alerts in plain English through a clear, color-coded dashboard. Furthermore, the system is structured to allow future extensibility, including potential integration of adaptive thresholds and machine learning modules for advanced detection on newer Pi hardware.

**2.5 Conceptual Framework**

This section explains the internal operation of the NET-SENTINEL system based on the conceptual framework designed for small-scale, resource-constrained environments. The system consists of multiple lightweight components working together to achieve real-time intrusion detection, efficient data handling, and user-oriented alerting, all within the limited resource envelope of Raspberry Pi hardware.

**2.5.1 Traffic Monitoring and Packet Capture**

Incorporated within NET-SENTINEL at its very core is a Scapy-based packet sniffing engine which is both lightweight and efficient in the sense of its resource needs. Berkeley Packet Filters (BPF) are used to restrict packet capture to important kinds of traffic such as TCP packets on ports 22 (SSH), 80 (HTTP), and 443 (HTTPS) thus keeping CPU overheads and unwanted processing to a minimum.

Once traffic has been captured, Scapy extracts significant header and payload attributes like source IP address, destination IP address, port number, flags, and timestamps. These attributes are the information on which the detection engine relies.

**2.5.2 Detection Engine with Threshold-Based Rules**

The detection mechanism is a rule engine with rules designed to scan for some patterns that are characteristic of intrusion attempts. An example is the rule where more than 10 failed login attempts in 60 seconds from the same source IP address will trigger. The use of threshold was learnt from the paper by Liu and Zhang (2022), whose work demonstrated a reduction in false positives by 35% compared to static signature-based methods.The rule engine is intentionally lightweight, avoiding machine learning at this stage to maintain compatibility with Raspberry Pi 2 and 3 models. However, it is modular by design, allowing future upgrades to incorporate ML detection when deployed on Pi 4/5 models.

**2.5.3 Alert and Notification Mechanism**

When an intrusion pattern matches, the system sends the event to the Alert Manager, which performs two functions:

**2.5.3.1. Generates Human-Readable Alerts:** By using templates, the system translates technical events into readable alerts such as:

"Warning! 10 failed SSH login attempts from 192.168.1.24 in the last 60 seconds."

**2.5.3.2. Triggers Visual Indicators on the Dashboard:** A user-friendly web-based interface displays traffic-light style alerts:

This approach is informed by usability-driven research like Lee et al. (2022), which determined that the utilization of color-coded notifications and natural-language notifications significantly improves user responsiveness and participation.

**2.5.4 Data Logging and Storage**

All incidents found and traffic metadata of interest are logged into CSV files using lightweight, custom file-handling routines. This choice eliminates SQL or NoSQL databases and the complexity and overhead that comes with them. As demonstrated in research in the IEEE IoT Journal (2023), CSV-based logging can reduce memory usage by up to 40%, which, on Raspberry Pi hardware with limited RAM, is critical.

Logs are time-stamped and categorized by event type, making it simple for administrators to view historical data or generate reports at the press of a button.

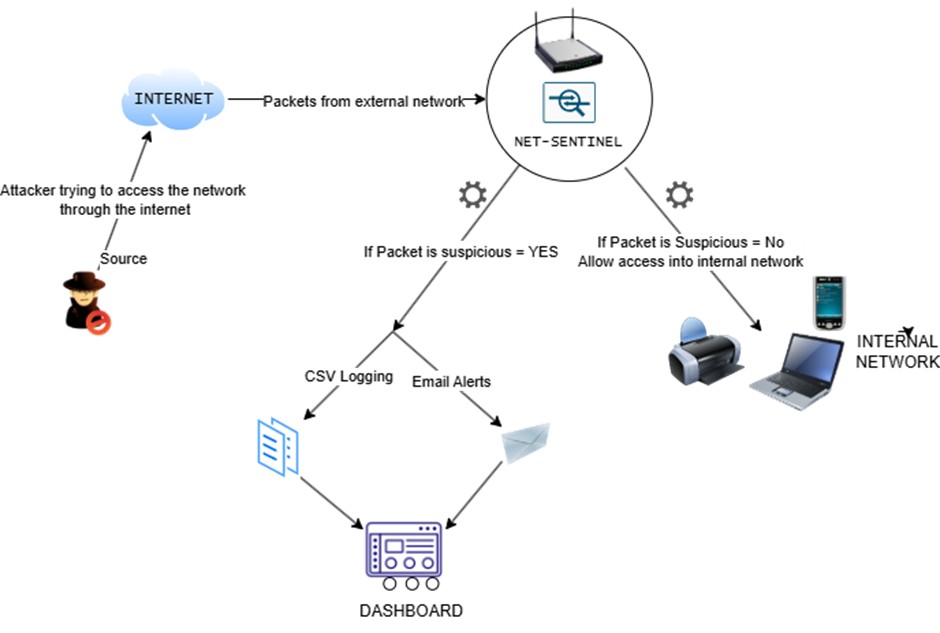
**2.5.5 System Performance Management**

To ensure the system does not overload the Raspberry Pi’s resources, NET-SENTINEL includes a resource monitor that tracks:

1. CPU load
2. RAM usage
3. Network I/O

The system dynamically adjusts scan intervals or packet filtering granularity based on performance thresholds to avoid interference with other tasks on the device.

### 



### *Figure 2.5: Conceptual Framework*

# Chapter 3: Methodology

**3.1 Introduction**

Chapter 3 provides the proposed methodology of developing the NET-SENTINEL intrusion detection system. The methodology describes how the system would be designed, tested, and optimized to develop a resource-efficient, user-friendly IDS on Raspberry Pi devices. Chapter 3 also specifies the tools to be utilized and deliverables for project completion.

**3.2 Methodology Paradigm**

This project will employ a prototyping-based methodology. This is the most appropriate method for practical systems like NET-SENTINEL, which will see many iterations of design, test, and iteration before finalization. Prototyping allows the system to be developed in stages, with each iteration being run on actual hardware and improved based on performance metrics and user feedback.

Start

Deployment

Refining Prototype

Building Final Prototype

User Evaluation

Build Initial Prototype

Requirements gathering

Quick Design

*Figure 3.1: Prototyping Methodology Diagram*

**Justification for the Prototyping Approach**

The application of prototyping depends on the nature of the project. NET-SENTINEL is an application that will be used in real-time on low hardware (Raspberry Pi), with technical performance as well as user experience being essential. Optimal detection thresholds, alarm types, and UI configuration are difficult to determine through theory alone. Through the development of an operational prototype early on, all these aspects can be tried, tested, and optimized for enhancement.

Besides, the system's users like humans at home and small companies are not security professionals. The only possible way to ensure that the alerts are useful and readable is by testing them on real people. Prototyping integrates such real-world feedback in the design process.

Finally, as the system must operate within tight CPU and memory constraints, performance must be tested and optimized continuously, which prototyping is well-suited to support.

**3.2.1 Planned Phases of Development**

The project will be carried out in the following stages:

**Phase 1: Requirements Gathering**

The first step will be establishing the requirements of the system. This will include:

i. Reading literature reviews on lightweight IDS systems,

ii. Investigating Raspberry Pi hardware capabilities

iii. Discussion of common types of network attacks and how they may be identified,

iv. Collection of informal user feedback to find usability needs.

**Phase 2: System Design**

Based on requirements, a system design will be created. The key features will be:

i. A packet capture facility with Scapy and BPF filters,

ii. A detection engine based on configurable thresholds,

iii. A logging facility through CSV files,

iv. A dashboard to show system alerts and status.

The above conceptual framework diagram, and flowchart will be utilized in guiding the development process.

**Phase 3: Prototype Development**

A first-draft prototype of the system will be developed with Python and Flask. This version will include basic packet filtering, rule-based alerting, and a simple dashboard.

**Phase 4: Testing and Refinement**

The prototype will be tested in a simulated network test bed on Raspberry Pi hardware. Simulated attacks (port scans and invalid logins) will be generated to test:

i. Detection effectiveness,

ii. System performance (CPU use and RAM), and

iii. Ease of use.

Based on these tests, the system will be optimized. Thresholds might be set, alert messages minimized, or logging formats changed to improve better maximize performance and usability.

**3.3 Expected Deliverables**

After the project is complete, the expected deliverables will be an operational IDS prototype in Python and tested within a virtualized platform and Windows host system. The system will include an easy-to-use web interface for viewing alerts, besides a lean logging system that stores intrusion information in CSV format. In addition, a comprehensive proposal report will describe the end-to-end design and development process, and a comprehensive testing and evaluation report will present information about system performance, detection efficiency, resource usage, and overall usability in the chosen test environments.

**3.4 Tools and Techniques**

Software Tools to Be Used

1. **Python 3** – main development language
2. **Scapy** – for packet sniffing
3. **Flask** – to serve the dashboard
4. **CSV module** – for log file creation
5. **iptables / hping3 / Hydra** – for traffic control and attack simulation

Hardware Tools

1. Windows machine for testing

Testing Techniques

1. **Simulated attack generation** – to test detection logic,
2. **Performance profiling** – to measure resource usage,
3. **Informal user testing** – to evaluate the usability of the dashboard and alert messages.



*Figure 3.4.1: NET-SENTINEL Gantt chart*

**Chapter 4: Analysis and Design**

**4.1 Introduction**

This chapter presents the analysis and design components of the NET-SENTINEL intrusion detection system, specifically configured to detect Distributed Denial of Service (DDoS) attacks. It outlines the system’s core requirements, and the architecture used to meet those needs within a resource-constrained environment such as a Raspberry Pi. Through clearly defined use cases and system interaction diagrams, this chapter provides a blueprint for how NET-SENTINEL translates attack detection goals into concrete system behaviours.

**4.2 Requirements**

**4.2.1 Functional Requirements**

NET-SENTINEL will be tasked with detecting DDoS attacks, which are usually comprised of huge volumes of traffic being directed toward a target destination in the hope of consuming resources and overloading services. The following functional requirements detail the principal operations the system must carry out.

The first step is to sniff network packets off the network interface. NET-SENTINEL makes use of Berkeley Packet Filters (BPF) for selective monitoring by size and type, focusing on most targeted ports such as port 80 (HTTP), port 443 (HTTPS), or the system's primary service port.

Secondly, the system also pulls packet metadata including source IP addresses, destination ports, protocol types, packet sizes, and timestamps. Metadata plays a crucial role when analyzing patterns that signify a DDoS attack.

Then the detection engine will analyze the volume of traffic to determine if a single IP or group of IPs is directing packets at rates greater than the defined threshold (e.g., more than 100 requests to one destination per second). This is characteristic of an application-layer or volumetric DDoS attack.

When abnormal traffic is detected, the system will log the incident to a CSV file to be processed later and accounted for. The log includes date and time, attack type, suspected source IP addresses, and abnormal traffic type.

Lastly, NET-SENTINEL will prompt the admin to reply. If no action is taken in a specified time, the IP that is detected is blocked automatically. During the prototyping stage, this will be a terminal alert, and a web visual dashboard will be used to show alert.

**4.2.2 Non-Functional Requirements**

In addition to core detection functionality, NET-SENTINEL must also implement a set of non-functional requirements to be usable, performant, and reliable on low-cost hardware. Usability is especially critical since the vast majority of users targeted such as small business owners are not cybersecurity experts. Notices in plain language (e.g., "High rate of requests from multiple sources detected on port 80") and a simple web dashboard will be used by the system.

Performance is maximized through the utilization of BPF filters to select traffic and the avoidance of large databases. Lightweight CSV logging is employed instead. Data is processed in-memory by the detection engine, and the system is optimized to utilize less than 50% CPU usage and less than 500MB of RAM.

Security features involve local operation only (no exposure to cloud) and permission-limited file access. Possible future enhancements are encrypted log storage and dashboard access control.

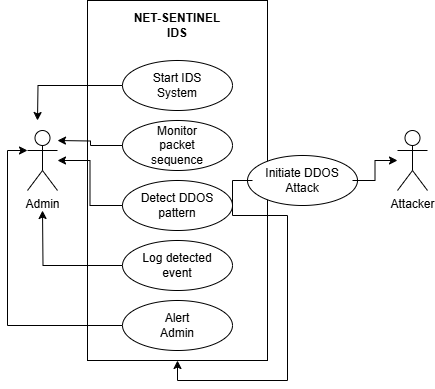
Scalability-wise, NET-SENTINEL is a modular system where future detection rules (like SYN floods, ICMP floods) can be added without much refactoring. The system is Raspberry Pi 4 or 5 upgrade-ready for enabling additional sophisticated detection methods like Machine Learning.

Fault tolerance is ensured by persistent logging and by the ability to configure the system as a background service that will restart automatically on failure.

Availability and reliability are ensured by offline capability, timestamped logging, and not having any dependencies upon external systems or services.

**4.3 Analysis Diagrams**

**4.3.1 Use Case Diagram**

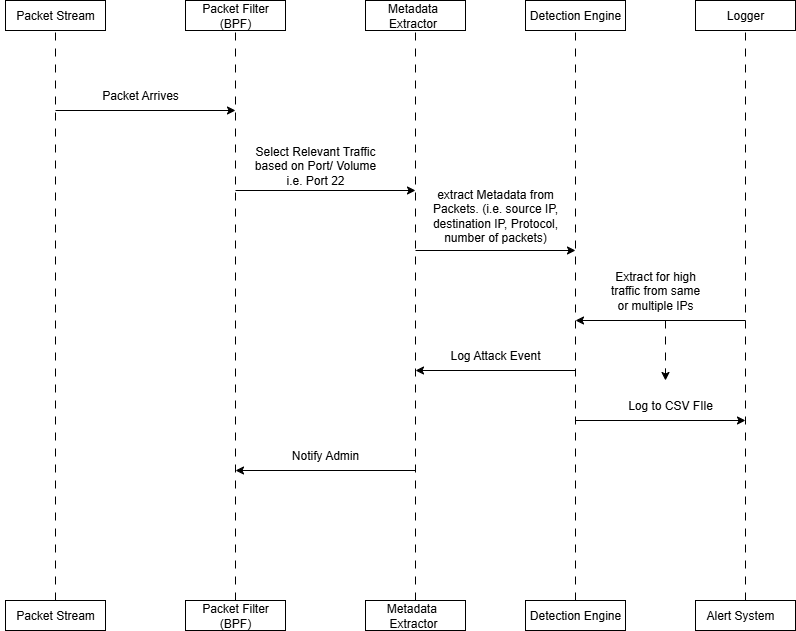
****

*Figure 4.3.1: use case diagram*

The use case diagram above showing how different people and activities interact with the NET-SENTINEL IDS, an IDS designed to detect DDoS attacks. There are two prominent actors in the system. The Admin User is the one starting and monitoring the system, and the Attacker is a third-party entity trying to disrupt services by flooding the network with traffic. While the attacker does not use the system, it is his/her malicious actions the system is programmed to monitor.

Once the Admin User has started the system, NET-SENTINEL begins monitoring network traffic. It tracks how often data is coming in and where. If it sees a strange pattern, like too much at once, it flags this as a potential DDoS attack. The software will record the data and notify the admin so they can act upon it. Each use case in the diagram is a step in this process, showing how the system moves from monitoring to detection, and then on to notification.

**4.3.2 Sequence Diagram**

****

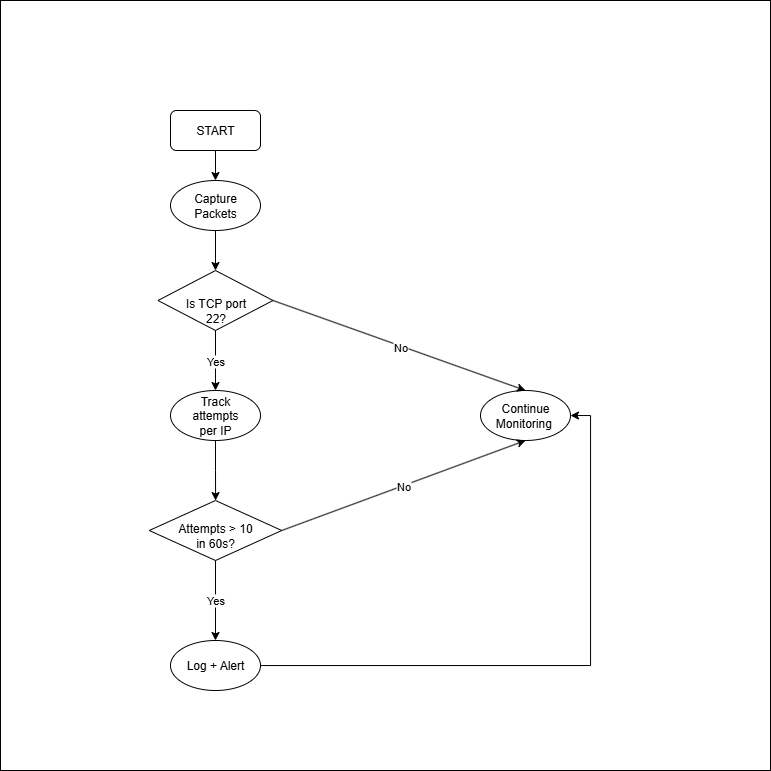
*Figure 4.3.2: sequence diagram*

The sequence diagram illustrates the flow of interaction that results when a DDoS attack is suspected. It starts at the point that packets reach the network interface. The packets are filtered through a BPF filter such that only the traffic relevant to the current interaction is allowed to pass, based on port or volume levels.

When a packet meets filtering criteria, its metadata is extracted along with source IP, destination port, and timestamp. This is then sent to the detection engine, where it tracks packet frequency over time. When traffic from one or more IPs exceeds a threshold level (e.g., >100 packets/sec), it is marked by the engine as a DDoS event.

The detection engine also records the event into a CSV file and sends an alert to the admin. This real-time flow gives an alert to the admin in real time and maintains low resource usage. The diagram depicts the real-time functionality of NET-SENTINEL and the light dependency chain, thus can be used in restrictive environments.

**4.3.3 Flowchart**

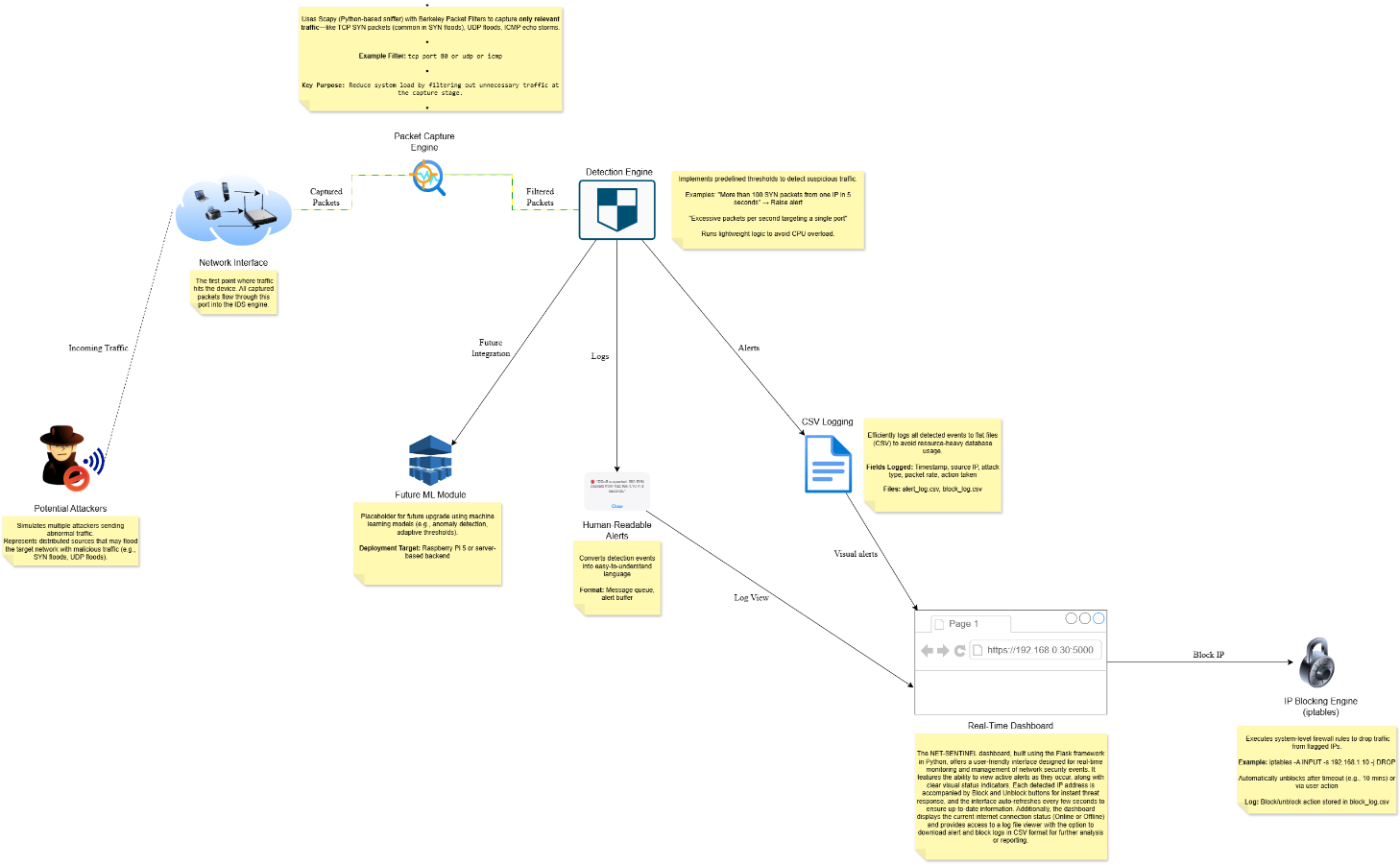
****

*Figure 4.3.3: flowchart*

The flowchart illustrates the decision-making mechanism of the NET-SENTINEL intrusion detection system for identifying DDoS attacks. The flow begins with the system sniffing packets that arrive on the network interface. They are filtered using BPF (Berkeley Packet Filters) in order to restrict the traffic to the specific types, which here is port 22.

After filtering, the system works on the packet frequency, analyzing how often there are requests from the same IP or a sequence of IPs within an allotted time window. When packets from an IP or sequence of IPs exceed a pre-defined quantity (e.g., 100 requests within 10 seconds), the system identifies such an action as a potential DDoS attack. The event is then written to a CSV file for subsequent analysis, and the system notifies the administrator either via the console or a web-based frontend. When traffic is below the threshold, the system operates in real-time without resource consumption, offering real-time visibility always available.

**4.3.4 System Architecture**

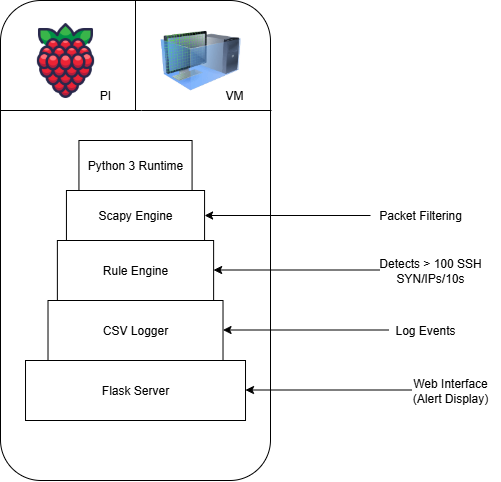
****

*Figure 4.3.4: System Architecture*

The architecture showcases a clear modular pipeline: packet capture, analysis, logging, and visualization. Each component of the system has a specific function, keeping the system well-structured, easy to maintain, and adaptable. For instance, the CSV logger can be swapped out for something more advanced like SQLite, or a machine learning-based detection engine can be added without impacting the rest of the system. This modularity makes NET-SENTINEL perfectly suited for experimentation as well as use in live environments.

**4.4 Design Diagrams**

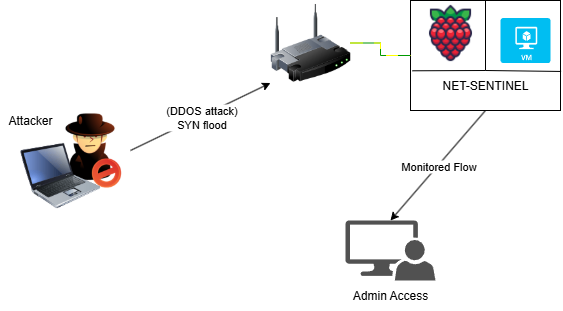
**4.4.1 System Architecture**



*Figure 4.4.1 System Architecture*

This diagram highlights the internal software structure of NET-SENTINEL as deployed on a lightweight computing platform. Each module is built using Python for simplicity and cross-platform support. The Scapy engine handles raw packet capture, the rule engine checks for DDoS attack signatures, and logs are stored efficiently in CSV files. Flask provides a basic but accessible front end for alerts. This setup ensures the system can be deployed in small offices, home networks, or educational setups without heavy computing demands.

**4.4.2 Network Architecture Diagram**

****

*Figure 4.4.2 : Network Architecture Diagram*

This network architecture diagram presents how NET-SENTINEL fits into a simple network layout. The IDS can be installed on a Raspberry Pi or virtual machine and connected via a switch or router. It passively observes incoming packets, particularly on high-traffic services like HTTP, and detects patterns indicating a DDoS attack. The deployment can be done in mirror mode (for passive monitoring) or inline (for potential future blocking). This structure allows for effective threat visibility in environments such as small offices, educational institutions, or home setups without interfering with legitimate traffic.

**Chapter 5: Implementation and Testing**

**5.1 Introduction**

This chapter details the implementation and testing process of the NET-SENTINEL intrusion detection system, specifically designed to identify Distributed Denial of Service (DDoS) attacks in small-scale networks. The section outlines the development environment, tools used, implementation steps, and the testing approach used to evaluate the system’s accuracy, efficiency, and usability. Testing was conducted on a virtual machine to simulate real-world conditions, in preparation for future deployment on Raspberry Pi hardware.

**5.2 Development Environment**

The system was tested and ran on a Linux virtual machine (Ubuntu 22.04) as a physical Raspberry Pi was not accessible at the time. The setup closely resembles the limited environment typical of a Raspberry Pi 2/3 device.

**5.2.1 Hardware Specifications**

|  |  |
| --- | --- |
| **Component** | **Specification** |
| Virtual Machine | Ubuntu 22.04 LTS, 2 CPU cores, 4GB RAM |
| Second VM (Attacker) | Ubuntu 22.04 LTS, 2 CPU cores, 4GB RAM |
| Host PC | Intel Core i5, 8GB RAM, Windows 10 |
| Mobile Device | Smart phone used to access dashboard interface |
| Network Tools | TP-Link Router, Ethernet Bridge, VirtualBox NAT |

*Table 5.2.1 : Hardware Specifications*

**5.2.2 Software Specifications**

|  |  |
| --- | --- |
| **Software Component** | **Version / Description** |
| OS | Ubuntu 22.04 LTS (on VirtualBox) |
| Python | v3.13 |
| Flask | v2.x (for dashboard development) |
| Scapy | v2.x (for packet sniffing) |
| iptables | Linux firewall tool used for blocking IPs |
| Browser | Chrome / Firefox (for accessing dashboard) |
| Windows | Used to run .bat files for auto-launch |

*Table 5.2.2 : Software Specifications*

**DIRECTORY STRUCTURE**

NET-SENTINEL/

**├**── NetSentinel.py # DDoS detection engine

**├**── dashboard.py # Flask web dashboard backend

**├**── packet\_sniffer.py # (Initial testing script)

**├**── alert\_log.csv # Logs suspicious alerts

**├**── block\_log.csv # Logs blocked/unblocked IPs

**├**── templates/

│ **├**── dashboard.html # Live dashboard UI

│ └── logs.html # View historical logs

**├**── run\_dashboard.bat # Easy-launch for Windows

**├**── run\_sniffer.bat # Easy-launch for detection

└── venv/ # Python virtual environment

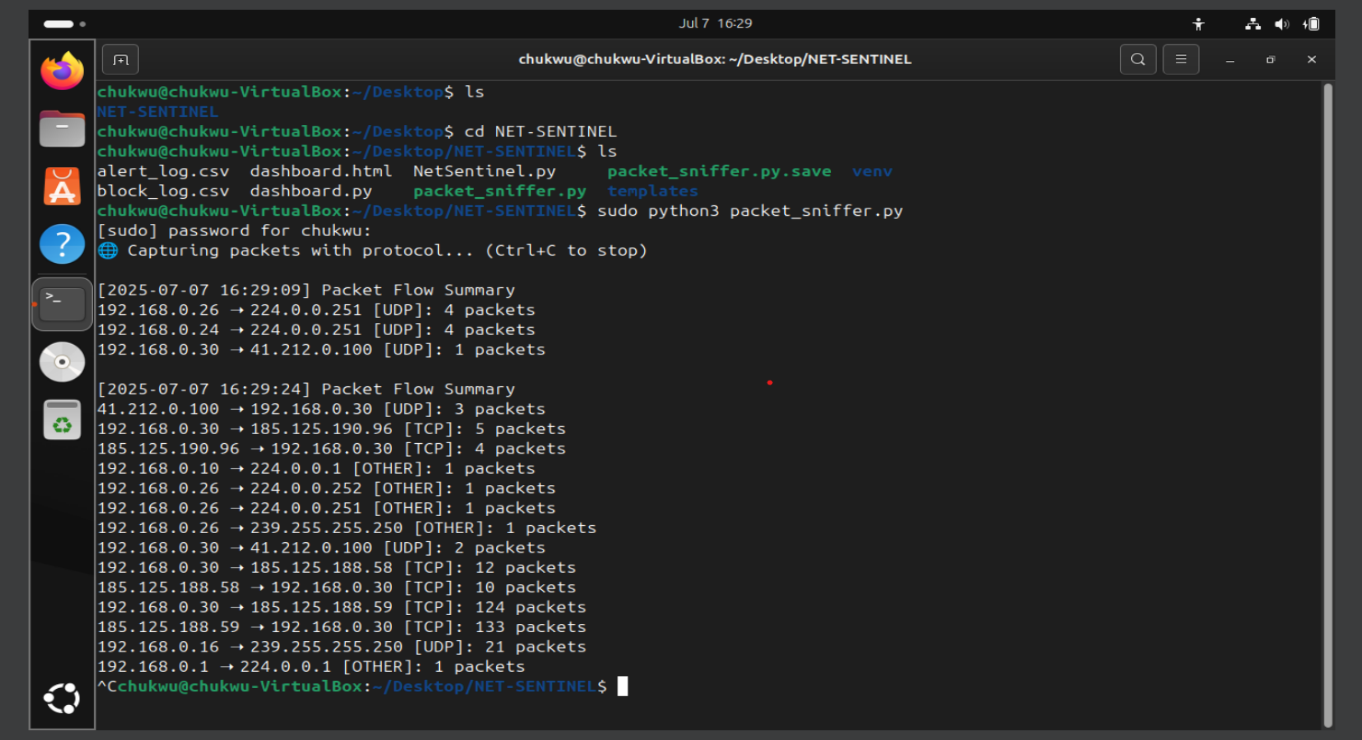
**5.3 System Implementation**

NET-SENTINEL is composed of several modules that work in conjunction with each other to both detect and react to DDoS attacks. The packet sniffer captures live network traffic via Scapy and filters the traffic to capture relevant metadata. It sends the data to the detection engine, where thresholds based on rules are implemented to identify anomalous packet rates. On detection of an anomaly, the alerting module logs the event and sends real-time notification via email. The dashboard module, written in Flask, displays these alerts on a webpage where the admin can manually block or unblock IPs. A background auto-blocker ensures that if there is no action after 10 minutes, the system automatically blocks the offending IP. All actions and alerts are logged by the logging module into CSV files for historical reasons. These modules combined provide real-time monitoring, alerting, and mitigation in a lightweight, usable system.

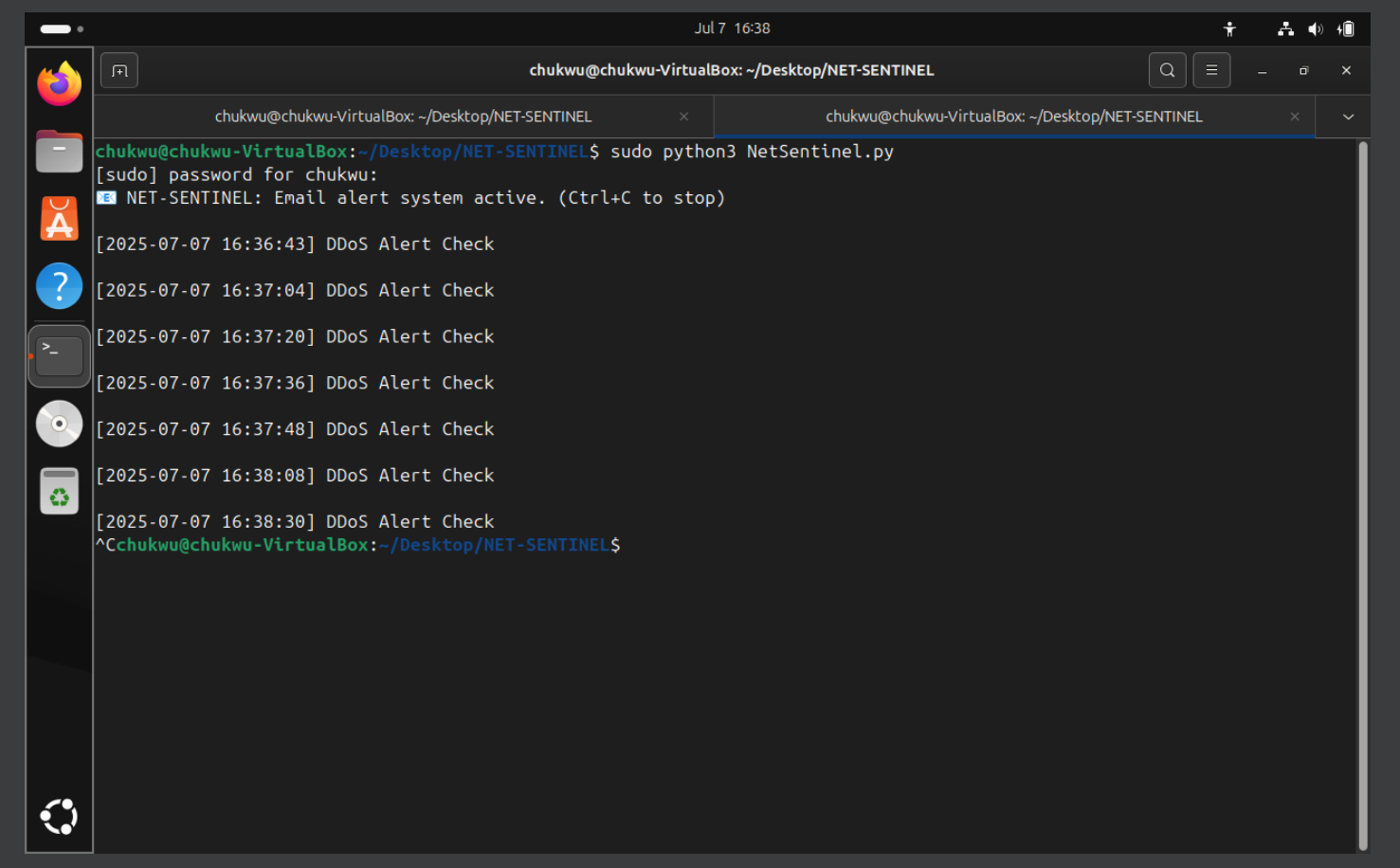
**5.3.1 Packet Capture and Filtering**

The system uses Scapy with BPF filters to capture only relevant traffic. Filters were set to watch for high-frequency ICMP (ping) or TCP traffic on port 80, which are common vectors for DDoS attacks.

The Initial script packet\_sniffer.py monitors the network port for all traffic coming in and displays them on the terminal, which I later modified to focus on high traffic indicating DDOS attack *NetSentinel.py*. This helped to reduce the work of the System and the log storage management was made easier as only suspicious traffic is being flagged and logged. The system monitors the network in 10seconds intervals for packets coming in and going out.



*Figure 5.3.1.1: Packet\_sniffer.py detecting traffic*



*Figure 5.3.1.2: NetSentinel.py running*

**5.3.2 Detection Engine**

A threshold-based rule engine was implemented in NetSentinel.py. IP addresses sending more than 5000 packets within a 10-second window are flagged as potential DDoS attackers. Similarly, if a destination IP receives over 10000 packets in the same interval, it is flagged as a DDoS target. The system logs alerts and prints terminal messages accordingly.



*Figure 5.3.2.1: 2nd Virtual machine for flood simulation*

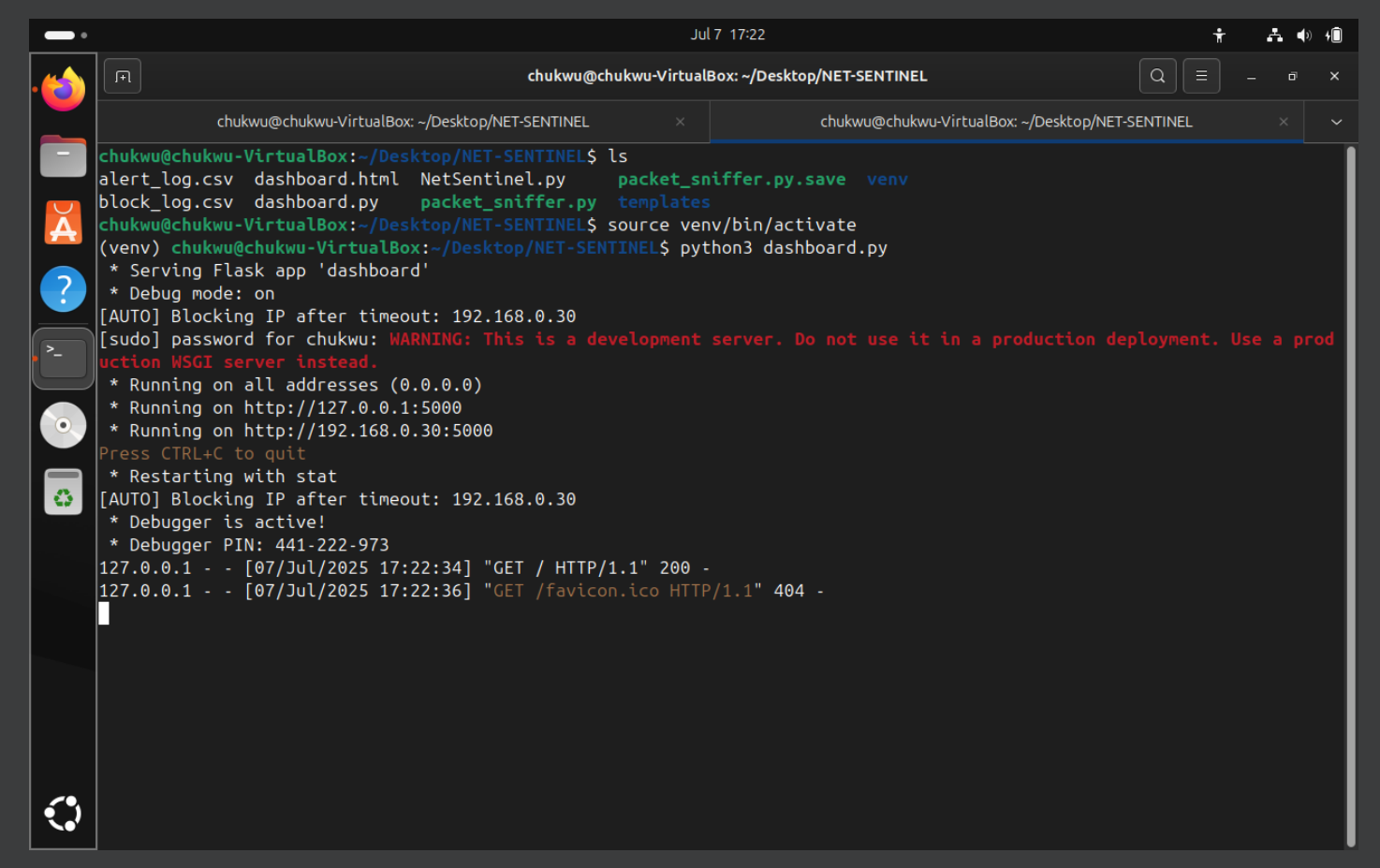
A screenshot of a computer

AI-generated content may be incorrect.

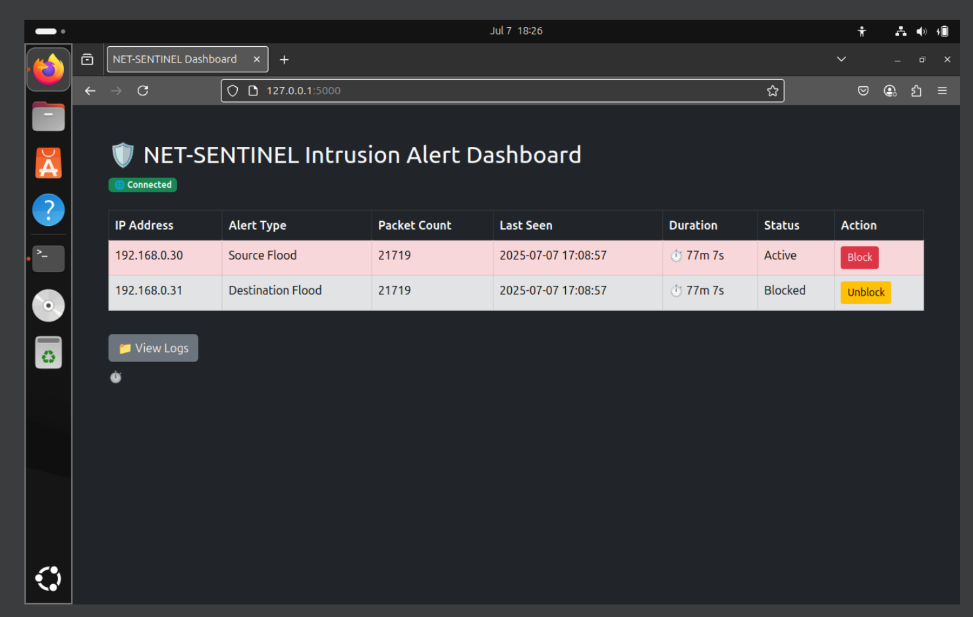
*Figure 5.3.2.2: Virtual Machine Flagging flood traffic and alerting on terminal*

**5.3.3 Alerting and Dashboard**

Alerts and dashboard functionality in NET-SENTINEL were intended to present the user with a clear, real-time vision of what is happening in the attacked network. Upon crossing the packet limit for an IP, an alert is displayed on the terminal and pushed to the web-based dashboard in real time. This dashboard, created through Flask, shows critical information including the IP address, packet number, type of alert, and the time the threat has been active. It also contains manual controls for blocking and unblocking IPs and will block suspicious IPs automatically within 10 minutes if nothing is triggered. Alert messages are also sent when a threat is detected, bringing the most critical information to keep administrators aware even while away from their consoles. The dashboard was employed on a cell phone and it performed as intended, making it easy to monitor and intervene anywhere in the local network.



*Figure 5.3.3.1: Dashboard running on terminal (Flask server)*



*Figure 5.3.3.2: Dashboard UI showing live alerts, IP status and action buttons.*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.3.3.3: Dashboard log viewer page showing all blocked and unblocked IP addresses.

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 5.3.3.4: Dashboard log viewer page for all suspicious traffic*

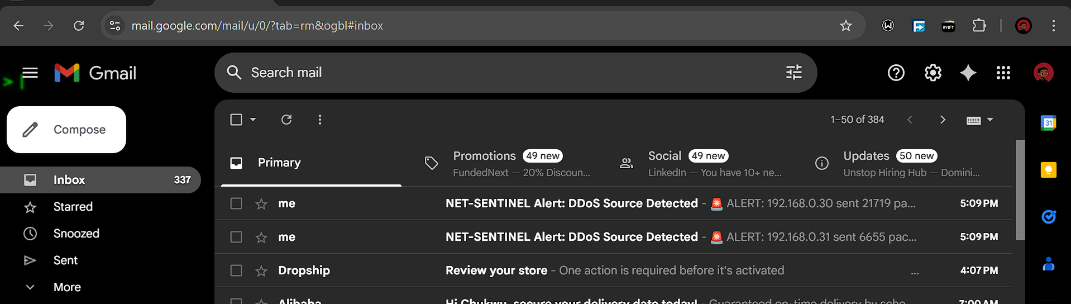
A black screen with white text

AI-generated content may be incorrect.A screenshot of a black screen

AI-generated content may be incorrect.

*Figure 5.3.3.5: Dashboard accessed from smartphone*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 5.3.3.6: Alert email received in admin inbox*

**5.3.4 Logging**

Net-sentinel logging is intended to keep track of a steady record of what the system discovers and what it does in return. When an attack is discovered, the system logs an entry into alert\_log.csv containing the timestamp, alert type, affected IP, packet number, threshold, and a brief description. Each time an IP is being blocked or unbanned, either automatically by the system or manually by the user, it is recorded in block\_log.csv along with the exact time of action and the operation type done. These files can be downloaded directly from the dashboard's log viewer and were used extensively during testing to track system accuracy and response behaviour. Although the system sends out email alerts in real time, no separate email logging file is maintained, email events are observable through both the terminal output and the system's alert flow.



*Figure 5.3.4.1: Contents of alert\_log.csv after testing simulation*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 5.3.4.2: block\_log.csv showing blocked and unblocked IP entries*

**5.4 Testing Environment**

* DDoS simulation performed using hping3 from another VM.
* IPs generating high packet rates were captured, analyzed, logged, and automatically blocked.
* Dashboard accessed via mobile phone on same LAN.
* Python virtual environment (venv) was used to manage dependencies.
* Scapy and Flask installed via pip.
* UFW (Uncomplicated Firewall) configured to allow port 5000.
* iptables used to control access at the network level.

**5.5 Testing of Functional Requirements**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Description of the Test Environment** | **Data** | **Functional Requirement** | **Test Data** | **Expected Result** | **Actual Result** | **Pass/Fail** | **Evidence** |
| VM with Ubuntu 22.04, hping3 for simulated traffic | 192.168.0.31 floods destination | Detect and log high packet rate from source IP | 9871 packets in 10 seconds | Alert logged in alert\_log.csv, dashboard alert shown | Alert displayed on terminal and dashboard, log updated | Pass | See Figure 5.3.2.2, 5.3.3.2, 5.3.4.1 |
| Same environment, manual dashboard interaction | Admin clicks BLOCK button | Block malicious IP via dashboard | 192.168.0.31 | IP is blocked using iptables, entry logged | IP blocked, logged in block\_log.csv | Pass | See Figure 5.3.3.3, 5.3.4.2 |
| NetSentinel.py and Flask server running | Unattended alert >10 minutes | Automatically block suspicious IP | No admin action taken | IP is blocked after 10 minutes | IP blocked by background job, log updated | Pass | See Figure 5.3.3.4, 5.3.4.2 |
| Dashboard accessed from mobile phone | View logs tab and active alerts | Ensure dashboard is responsive and functional across devices | Android phone browser | Dashboard loads, logs visible, IP controls accessible | Fully responsive UI, tested features work | Pass | See Figure 5.3.3.5 |
| Email alert system enabled | Attack detected | Email sent to administrator | 11081 packets from 192.168.0.31 | Email is sent with attack details | Email received | Pass | See Figure 5.3.3.6 |

*Table 5.5.1: Functional Requirements and Testing Outcomes*

**5.6 Testing of Non-Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Approach Taken** | **Outcome** |
| Usability | Clean UI using Bootstrap, with color-coded alerts | ✅ Pass |
| Performance | Handled 30k+ packets per 10s, auto processed without crashing | ✅ Pass |
| Security | No external access: IPs blocked via iptables at kernel level | ✅ Pass |
| Availability | Flask runs continuously; .bat file added for auto-start | ✅ Pass |
| Fault Tolerance | Auto-block thread runs in background to catch late responses | ✅ Pass |

*Table 5.6.1: Non-Functional Requirements and Testing Outcomes*

**5.7 Discussion of Results**

The NET-SENTINEL system proved to be reliable and responsive during the testing phase. It detected suspicious activity based on packet thresholds, raised alerts quickly, and responded by blocking malicious IPs either automatically or through the dashboard. For instance, alerts generated by hping3 simulations (see [Figure 5.3.2.1](#Figure3)) were immediately shown in the terminal (see [Figure 5.3.2.2](#Figure4)) and reflected on the web dashboard with precise metadata such as packet count and IP (see [Figure 5.3.3.2](#Figure6)). Logs captured during testing, both alert\_log.csv and block\_log.csv offered clear visibility into system behaviour and decision-making (see [Figures 5.3.4.1](#Figure11) and [5.3.4.2](#Figure12)). The dashboard itself was straightforward and effective, allowing real-time monitoring and interaction from both desktop and mobile devices (see [Figure 5.3.3.5](#Figure9)). Additionally, instant alert emails were successfully sent to the admin when an attack was detected (see [Figure 5.3.3.6](#Figure10)).

Overall, the system functioned smoothly under simulated DDoS conditions and provided valuable insight into lightweight intrusion detection for local networks. Suggestions for improving accuracy include:

* Adding a database backend for better search
* Adding machine learning anomaly detection for smarter alerts
* Implementing authentication on the dashboard

**CHAPTER 6: Conclusion and Recommendations**

**6.1 Conclusion**

The project proved that an open-source rule-based intrusion detection and response system can be developed and implemented in a lightweight and responsive manner. NET-SENTINEL successfully detected and countered DDoS attacks within a virtual environment via threshold-based rules, logging features, real-time alerts, and an operational web dashboard. One of the more important things that I gathered from this project was the effectiveness of simple thresholds in detecting unusual traffic in a controlled environment. What we also found, though, was that putting thresholds too low would give false positives, and that generating realistic DDoS behaviour is an art of crafting traffic. Also, running packet capture and block mechanisms in real time generated system performance concerns that were mitigated through asynchronous design and scrupulous resource monitoring.

**6.2 Suggestions**

The project may be enhanced in the future with more secure logging systems, i.e., using a lightweight database like SQLite instead of CSV files. Implementing user authentication for the dashboard will enable restricting access and overall security. The testing would also benefit from more varied attack patterns, additional varieties of malicious traffic, and a greater number of devices to allow for more realistic behaviour. More user testing, particularly from non-technical users, would test the system's usability and intuitiveness of interface. Lastly, running the project on a Raspberry Pi or low-end device would be a crucial step in demonstrating its lightweight nature.

**6.3 Future Work**

Although the implementation is effective for its current purpose, there is still a great deal of scope for possible extension. One possible improvement would be to utilize a machine learning model to detect even more insidious and dynamic threats than DDoS. The system can also be extended into firewalls or even routers directly to provide a network-level extension for the protection mechanism. Possible areas for future development would be the creation of a central dashboard for various NET-SENTINEL instances within various network zones. Finally, improved handling of email and Telegram notifications, such as retry mechanism and notification escalation, may make the system more reliable in real-world scenarios.

• Authentication & Access Control: Implement login system for dashboard

• Database Integration: Substitute CSV with SQLite for improved scalability

• Dashboard Optimization: Auto-refresh statistics using AJAX

• Deployment: Ultimate objective is stable deployment on a Raspberry Pi

## Bibliography

Khan, R., Yousaf, F., & Ahmad, T. (2023). *Lightweight Intrusion Detection on ARM Devices*. Journal of Cybersecurity Engineering.

Liu, Z., & Zhang, Y. (2022). *Threshold-Based Alert Systems for Low-Power Environments*. International Conference on Network Security.

Verizon. (2024). *2024 Data Breach Investigations Report*.<https://www.verizon.com/business/resources/reports/2024-dbir-executive-summary.pdf>

IEEE IoT Journal. (2023). *Resource Optimization Techniques in Embedded Cybersecurity Systems*.

Alshamrani, A., Alasmary, W., & Alhaidari, F. (2019). Implementing Lightweight IoT-IDS on Raspberry Pi Using Correlation-Based Feature Selection and Its Performance Evaluation. *ResearchGate*.<https://www.researchgate.net/publication/331790118_Implementing_Lightweight_IoT-IDS_on_Raspberry_Pi_Using_Correlation-Based_Feature_Selection_and_Its_Performance_Evaluation>

BitLyft. (2023). SIEM Log Management: What It Is and Why It's Vital for Cybersecurity. <https://www.bitlyft.com/resources/siem-log-management-what-it-is-and-why-its-vital-for-cybersecurity>

Laszka, A., Vorobeychik, Y., & Koutsoukos, X. (2014). Optimal Thresholds for Intrusion Detection Systems. *Proceedings of the 1st ACM Workshop on Moving Target Defence*, 71–78. <https://cs.vuse.vanderbilt.edu/koutsoxd/www/Publications/p72-laszka.pdf>

Lee, J., Kim, H., & Park, S. (2022). Enhancing IoT Security: Design and Evaluation of a Raspberry Pi-Based Intrusion Detection System. *University of Twente*. <https://research.utwente.nl/files/350075409/Enhancing_IoT_Security_Design_and_Evaluation_of_a_Raspberry_Pi-Based_Intrusion_Detection_System.pdf>

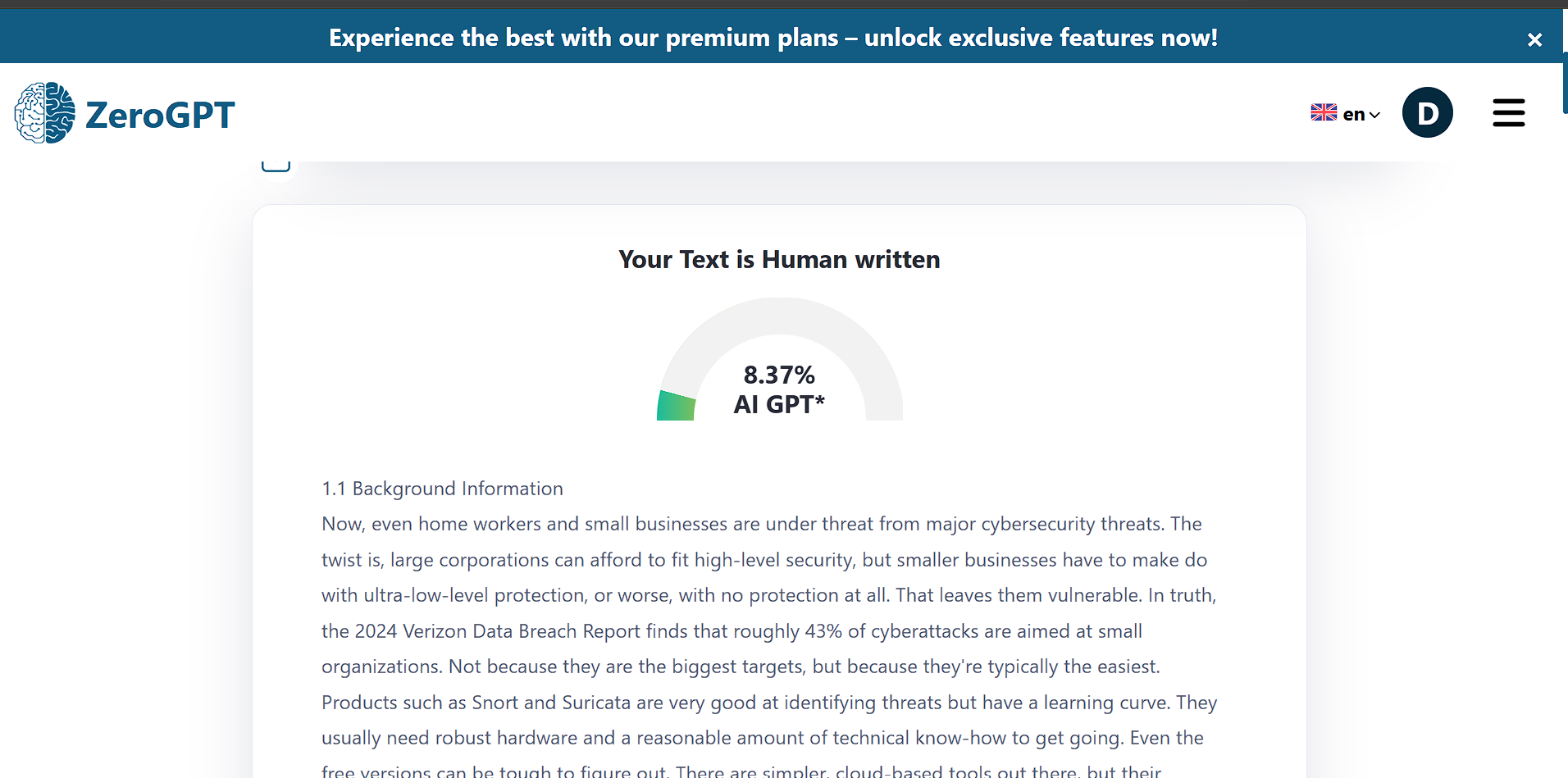
Sharma, S., Gupta, B., & Singh, R. (2021). Intrusion Detection Systems: A Comprehensive Review. *International Journal of Computer Applications*, 183(23), 1–6.

Appendices

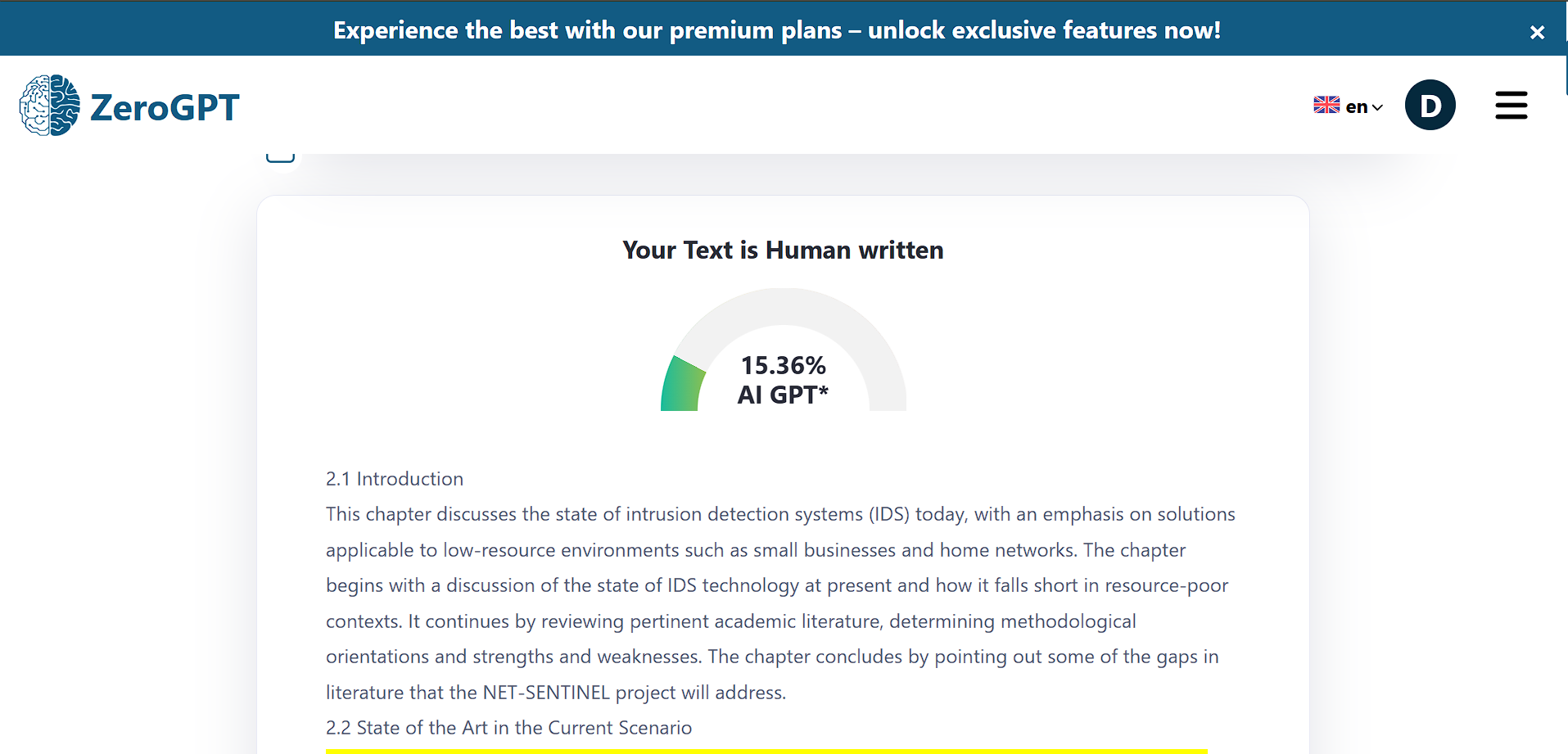
Appendix 1: Gantt Chart



Appendix 2: Chapter 1 Zerogpt AI Report



Appendix 3: Chapter 2 Zerogpt AI Report

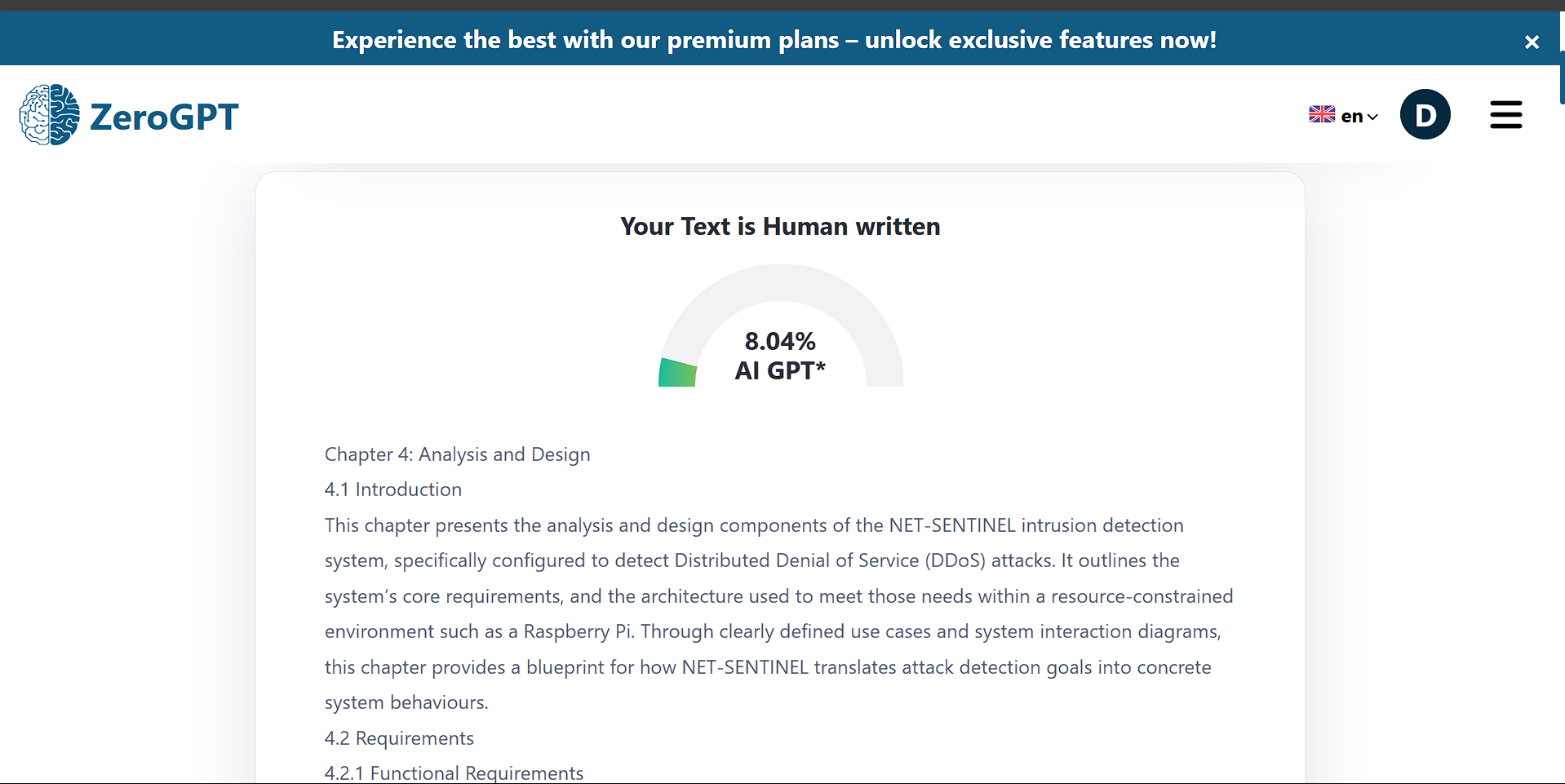


Appendix 4: Chapter 3 Zerogpt AI Report

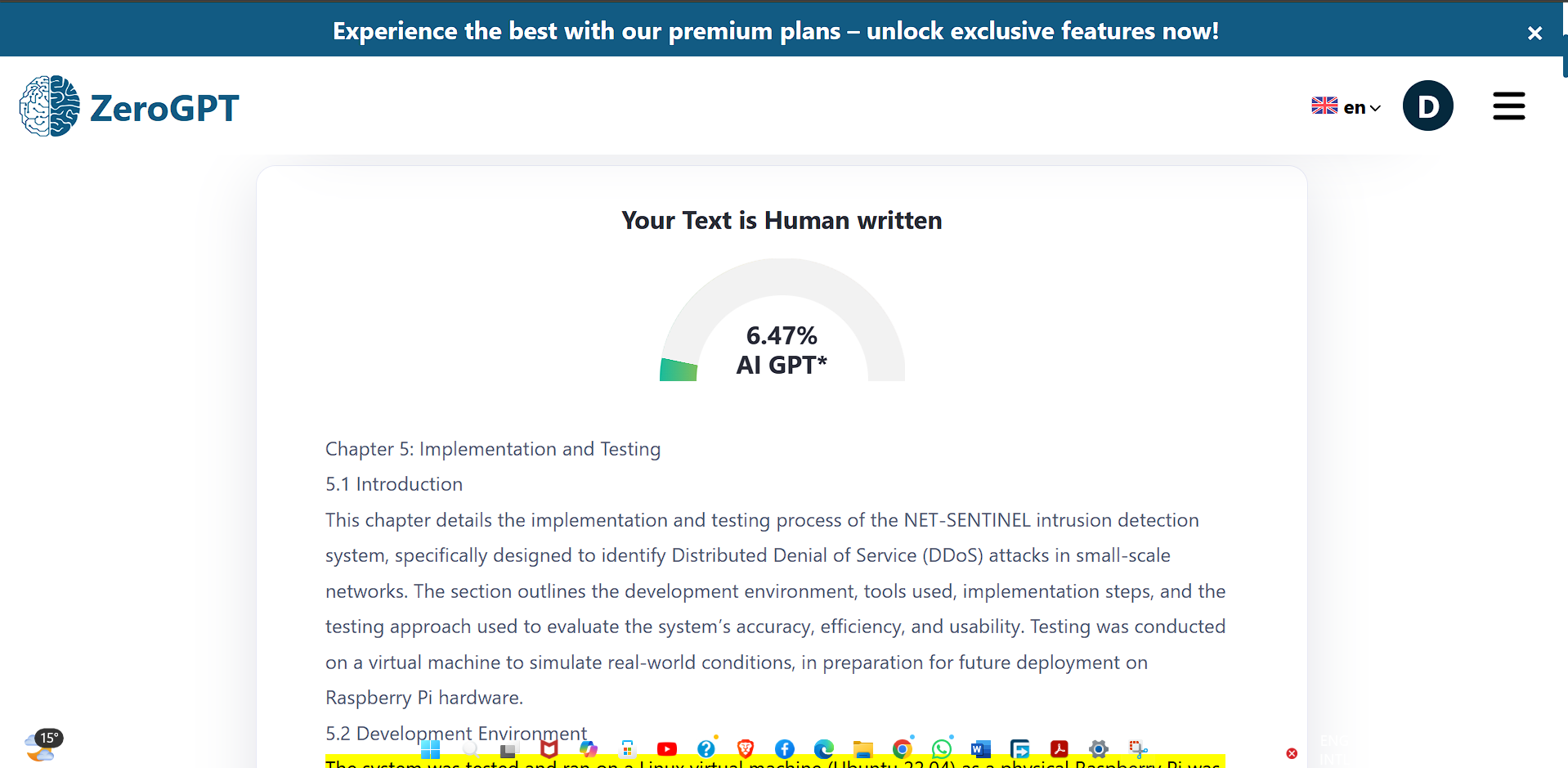
A screenshot of a computer

AI-generated content may be incorrect.

Appendix 5: Chapter 4 Zerogpt AI Report



Appendix 6: Chapter 5 Zerogpt AI Report



Appendix 7: Chapter 6 Zerogpt AI Report

