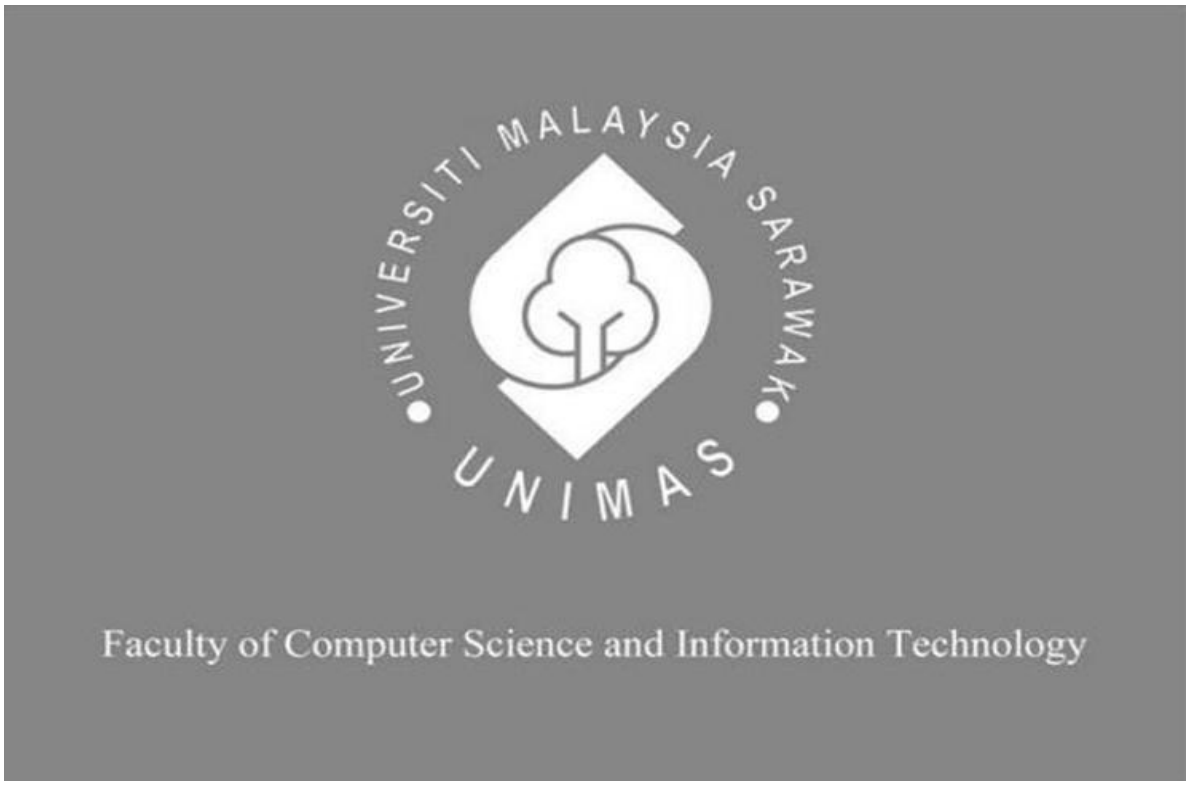



# *IMPLEMENTATION OF A SIGNATURE-BASED IDS USING SNORT FOR NETWORK SECURITY*

**Ainin Sofiya Binti Mohamad**

**Bachelor of Computer Science with Honours**

**(Network Computing)**

**2025**

# IMPLEMENTATION OF A SIGNATURE-BASED IDS USING SNORT FOR NETWORK SECURITY

AININ SOFIYA BINTI MOHAMAD

This project is submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Computer Science with Honours

(Network Computing)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

# IMPLEMENTATION OF A SIGNATURE-BASED IDS USING SNORT FOR NETWORK SECURITY

AININ SOFIYA BINTI MOHAMAD

Projek ini merupakan salah satu keperluan untuk

Ijazah Sarjana Muda

Sains Komputer dengan Kepujian

(Pengkomputeran Rangkaian)

Fakulti Sains Komputer dan Teknologi Maklumat

UNIVERSITI MALAYSIA SARAWAK

2025

**DECLARATION**

I am here to declare that this project is my original work. I have not copied from any other students work or from any other sources except where due reference or acknowledgement is not made explicitly in the text, nor has any part has been written for me by another person.

I confirm that I have read and understood the UNIMAS regulations regarding plagiarism and will be penalized by the university if found guilty

Signed by:

Ainin Sofiya Binti Mohamad Date: 21/6/2025

77773

# Acknowledgements

Alhamdulillah, with deep gratitude, I thank Allah SWT for granting me the strength, patience, and determination to successfully complete my Final Year Project. This journey has not only tested my capabilities but also allowed me to grow personally and academically. Reaching this milestone is a blessing, and I am truly thankful.

First and foremost, I would like to express my heartfelt appreciation to my supervisor, Dr. Adnan Shahid Khan, for his continuous support, expert guidance, and valuable feedback throughout the development of this project. His dedication, encouragement, and insightful advice greatly enhanced my understanding and motivated me to keep moving forward, even when faced with challenges.

My sincere gratitude also goes to my beloved family. Though distance may have separated us, their unwavering love, prayers, and support were always felt. Their faith in me gave me strength during moments of doubt and reminded me of my purpose. Without their constant encouragement, I would not have come this far.

I am also thankful to my friends, who supported me in countless ways through sharing ideas, offering motivation, and simply being there when I needed them. Their presence made this journey more enjoyable and manageable, and I cherish the memories we created along the way.

Additionally, I would like to thank all lecturers, peers, and individuals who contributed directly or indirectly to the success of this project. Whether through classroom teachings, helpful suggestions, or words of encouragement, each of you played an important role, and I am truly grateful.

# Abstract

*This project focuses on implementing a signature-based Intrusion Detection System (IDS) using Snort on a Linux-based environment to enhance network security through real-time detection of cyber threats. Custom Snort rules are designed to detect Denial of Service (DoS), port scanning, and SQL injection attacks effectively. The project integrates a Python-based machine learning module that analyzes Snort alert logs to improve detection accuracy and reduce false positives. By simulating attacks from an attacker virtual machine, the system's performance in identifying threats is evaluated based on accuracy, response time, and classification metrics. The methodology used is the Design Science Research Methodology (DSRM), which guides the development, implementation, and evaluation of the system. This project provides practical experience in deploying IDS, crafting detection rules, and applying intelligent analysis to enhance threat identification. The system aims to deliver reliable intrusion alerts while minimizing administrative burden, contributing to stronger and more accurate network monitoring solutions in modern cybersecurity environments.*

# Abstrak

Projek ini memfokuskan pada pelaksanaan Sistem Pengesan Pencerobohan (IDS) berasaskan tandatangan menggunakan Snort pada persekitaran berasaskan Linux untuk meningkatkan keselamatan rangkaian melalui pengesanan masa nyata ancaman siber. Peraturan Snort tersuai direka untuk mengesan Denial of Service (DoS), pengimbasan port dan serangan suntikan SQL dengan berkesan. Projek ini menyepadukan modul pembelajaran mesin berasaskan Python yang menganalisis log amaran Snort untuk meningkatkan ketepatan pengesanan dan mengurangkan positif palsu. Dengan mensimulasikan serangan daripada mesin maya penyerang, prestasi sistem dalam mengenal pasti ancaman dinilai berdasarkan ketepatan, masa tindak balas dan metrik klasifikasi. Metodologi yang digunakan ialah Metodologi Penyelidikan Sains Reka Bentuk (DSRM), yang membimbing pembangunan, pelaksanaan, dan penilaian sistem. Projek ini menyediakan pengalaman praktikal dalam menggunakan IDS, mencipta peraturan pengesanan dan menggunakan analisis pintar untuk meningkatkan pengenalan ancaman. Sistem ini bertujuan untuk menyampaikan amaran pencerobohan yang boleh dipercayai sambil meminimumkan beban pentadbiran, menyumbang kepada penyelesaian pemantauan rangkaian yang lebih kukuh dan lebih tepat dalam persekitaran keselamatan siber moden.

# Table of Contents

# List of Figures

# List of Table

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

In today's digital era, network infrastructure forms the backbone of modern organizational operations. However, this increasing reliance also exposes systems to a growing number of cyber threats that exploit vulnerabilities for malicious purposes. Cyberattacks such as unauthorized access, data breaches, and service disruptions can cause significant damage to an organization's data integrity, confidentiality, and availability. As a result, establishing a robust cybersecurity framework has become essential not only to prevent attacks but also to monitor, detect, and respond to suspicious activities.

Intrusion Detection Systems (IDS) are a key element in this proactive cybersecurity approach. IDS tools monitor network traffic and issue alerts when potential threats are detected, enabling security teams to respond quickly and mitigate risks. One of the most widely used IDS tools is Snort, a signature-based system that detects threats by comparing network packets to known attack patterns or signatures. This method provides effective real-time detection of specific threats and is especially useful for identifying attacks such as Denial of Service (DoS), port scanning, and SQL injection (Pramudya & Alamsyah, 2022).

This project aims to implement Snort as a network IDS on a Linux-based system and design custom detection rules for the attack types mentioned above. While Snort is powerful in detecting known threats, it has limitations, most notably its struggle to detect unknown or sophisticated attacks and its tendency to produce false positives. These false positives can overwhelm administrators, leading to delayed responses and the potential for missed critical threats.

To address these limitations, the project integrates machine learning (ML) techniques with Snort. By analyzing Snort's alert logs, machine learning models can learn patterns of normal and malicious behavior, enabling the system to classify alerts more accurately. This integration improves detection accuracy and helps reduce false alarms by distinguishing genuine threats from benign anomalies. Importantly, the machine learning module will be hosted externally, separate from the Snort IDS system, to improve scalability and performance. The external ML model processes Snort logs and classifies alerts based on patterns learned from simulated attack data.

Furthermore, Snort's flexibility and customizability enhance its relevance in modern cybersecurity strategies. Users can craft tailored rules to suit specific network environments, making Snort adaptable to various use cases (Al Ghafri et al., 2019). Supported by an active open-source community, Snort continues to evolve with frequent updates and shared best practices, reinforcing its value as a dynamic tool for intrusion detection in an ever-changing threat landscape.

## 1.2 Problem Statement

In today's digital era, organizations rely heavily on computer networks to run their operations smoothly. However, this dependence also increases their risk of facing cyber threats such as Denial-of-Service (DoS) attacks, port scanning, and SQL injection. According to Cybersecurity Ventures (2024), global cybercrime is expected to cost USD 10.5 trillion annually by 2025, showing how serious and widespread network attacks have become. As these threats become more advanced and happen more frequently, organizations need better tools to monitor, detect, and react to suspicious activity.

Intrusion Detection Systems (IDS) like Snort are commonly used to detect known attacks by checking network traffic against a list of predefined attack patterns, also called signatures. While this method works well for detecting familiar threats, it has two major weaknesses. First, it cannot detect new or unknown attacks, often called zero-day attacks, because no existing pattern matches them (Tripathy & Behera, 2024; Pureti, 2022). Second, it often produces too many false alerts treating normal or harmless traffic as a threat. This overload of alerts can confuse or slow down the response of security teams, increasing the risk of missing real attacks (Umer et al., 2022).

To overcome these challenges, this project improves Snort's effectiveness by adding a machine learning (ML) component that analyzes its alert logs. This component is designed to recognize patterns in both normal and malicious behavior, helping to classify alerts more accurately. By doing so, the system aims to improve threat detection, lower false alarm rates, and offer better identification of unusual or advanced network attacks.

## 1.3 Objectives

The objectives are as follows:

1. Install and configure Snort as a network Intrusion Detection System (IDS) on a Linux-based system and design custom Snort rules to detect various attack types, including DoS, port scanning, and SQL injection.
2. Integrate an external machine learning module with Snort to enhance detection accuracy, reduce false positives, and improve overall effectiveness in identifying advanced and unknown threats.
3. Simulate network attacks to test and evaluate the performance of the IDS, ensuring its ability to accurately detect and respond to different intrusion scenarios.

## 1.4 Project Methodology

Design Science Research Methodology (DSRM) is a structured framework used to create and evaluate artifacts, such as models, systems, or methods, to solve specific problems. This methodology is ideal for systematic approach for designing and assessing the implementation of Snort as a signature-based IDS, ensuring that each step from development to testing is methodically covered while emphasizing practical solutions supported by comprehensive documentation and evaluation (Venable et al., 2017). In this section, the implementation of the project in every phase of this methodology will be explained below in Figure 1.1.


*Figure 1.1:* Design Science Research Methodology (DSRM) (Arsalan et al., 2023)

### 1.1 Identify the Problem

This phase begins by examining current network security challenges, particularly the limitations of traditional intrusion detection systems that rely solely on predefined signatures. The core issue is the inability to detect zero-day attacks and the high rate of

false positives, which hinders the effectiveness of existing IDS solutions. This project proposes the development of a hybrid Snort-based IDS with machine learning capabilities to improve detection accuracy.

## 1.2 Define the Solution

The solution involves installing and configuring Snort as a signature-based IDS on a Linux platform. Custom Snort rules will be created to detect common attacks like Denial of Service (DoS), port scanning, and SQL injection. Additionally, a machine learning model will be integrated to reduce false positives and improve detection of unknown or advanced threats.

## 1.3 Design and Development

The IDS design centers on Snort 2.9.20 as the detection engine, incorporating packet decoding and preprocessing for traffic normalization. Custom rules are developed to identify specific attack signatures, and alert output is configured using the alert file format. A Python-based machine learning module is implemented to analyze Snort alerts and distinguish real threats from false alarms.

## 1.4 Demonstration

The IDS system is deployed and tested in a controlled environment using two virtual machines: Linux Mint running Snort and Attacker simulating attacks such as Nmap scans, Hping3 DoS, and SQL injection via sqlmap. These simulations validate the system's detection capabilities and assess the effectiveness of the machine learning classifier in categorizing alerts.

### 1.5 Evaluation

The IDS performance is evaluated based on detection accuracy, false positive rate, and alert response time. The machine learning model's classification metrics, including precision, recall, and F1-score, are analyzed to assess effectiveness. Evaluation results guide the refinement of Snort rules and further improvements to the machine learning integration.

### 1.6 Communication

The final phase documents the project comprehensively, covering methodology, system design, implementation details, and evaluation results. This report highlights how integrating Snort with machine learning enhances network security through improved threat detection and reduced false positives.

## 1.5 Scope of the Project

This project focuses on the deployment and configuration of a signature-based Intrusion Detection System (IDS) using Snort 2.9.20 on a Linux Mint virtual machine. Custom Snort rules will be developed to detect specific attack types, including Denial of Service (DoS), port scanning, and SQL injection. Network attack simulations will be performed from a separate Attacker VM to validate detection effectiveness. Alerts generated by Snort will be logged in alert file format and processed by a Python-based machine learning module designed to classify and reduce false positives. Continuous evaluation of detection accuracy, response time, and classification performance will be conducted to ensure the system meets the defined security objectives and improves threat identification.

## 1.6 Significance of the Project

This project contributes to the field of cybersecurity by implementing a practical signature-based Intrusion Detection System using Snort 2.9.20 on a Linux platform. It provides hands-on experience in configuring and customizing IDS rules to detect common network attacks such as Denial of Service (DoS), port scanning, and SQL injection. By integrating a Python-based machine learning module, the project improves Snort's ability to reduce false positives and detect more complex or previously unknown threats. These enhancements lead to more accurate and reliable network security monitoring, offering organizations a stronger and more efficient defense mechanism. Furthermore, the project demonstrates the real-world relevance of combining traditional IDS techniques with machine learning to meet modern cybersecurity challenges.

## 1.7 Expected Outcome

This project aims to develop a functioning signature-based Intrusion Detection System (IDS) using Snort, designed to perform real-time monitoring and identify a range of network intrusion attempts. A tailored set of custom Snort rules will be developed to effectively detect specific attacks such as Denial of Service (DoS), port scanning, and SQL injection. Testing through simulated attacks from an attacker Linux virtual machine (VM) will validate the system's detection accuracy and alert generation capabilities. Furthermore, the integration of a machine learning classifier will enhance the system's performance by reducing false positives and improving detection of advanced and previously unknown (zero-day) threats. Ultimately, this project will demonstrate how combining Snort's rule-based detection with machine learning techniques can create a more adaptive and accurate intrusion detection solution to meet evolving cybersecurity challenges.

## 1.8 Project Schedule

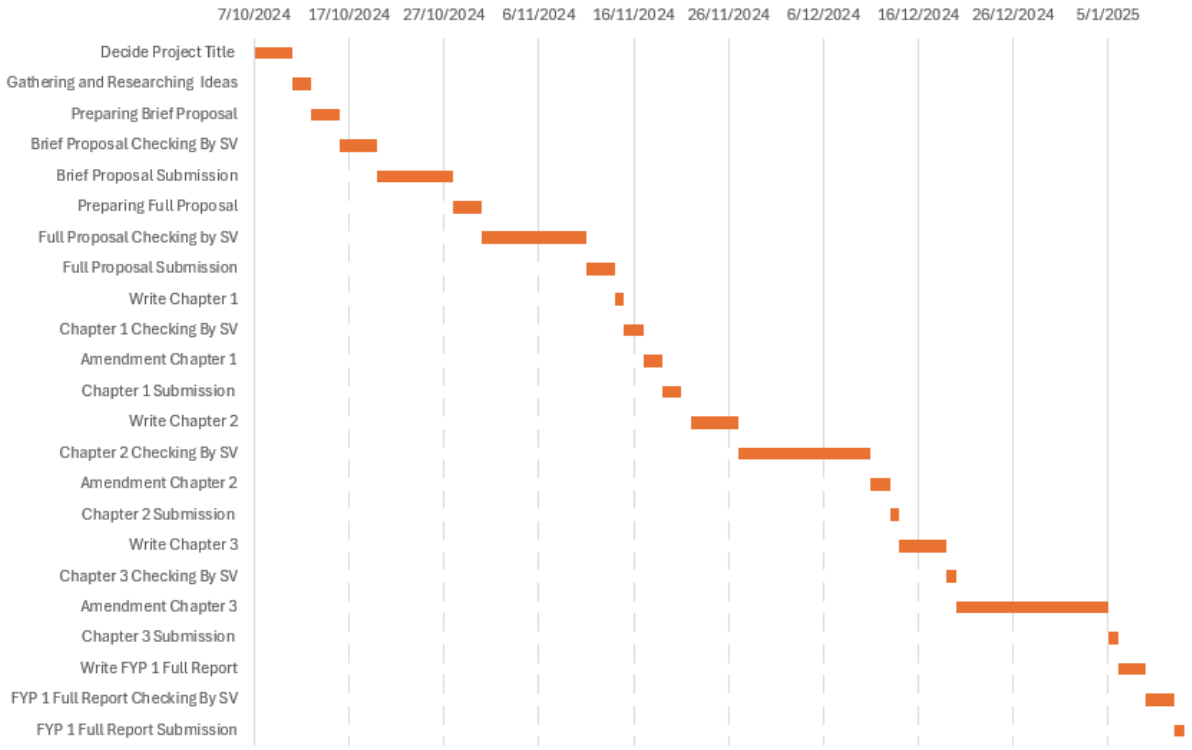



*Figure 1.2:* FYP 1 Milestone and Task Schedule

This project will be divided into two parts, which are FYP 1 and FYP 2. FYP 1 will focus on the foundational aspects, including defining the project objectives, gathering technical requirements, and preparing comprehensive documentation. Then, it will be the development of the prototype which will be in FYP 2. Based on the figure above, this Gantt chart will be only focusing on FYP 1. The time to complete FYP 1 is roughly three months from October 2024 to January 2025.

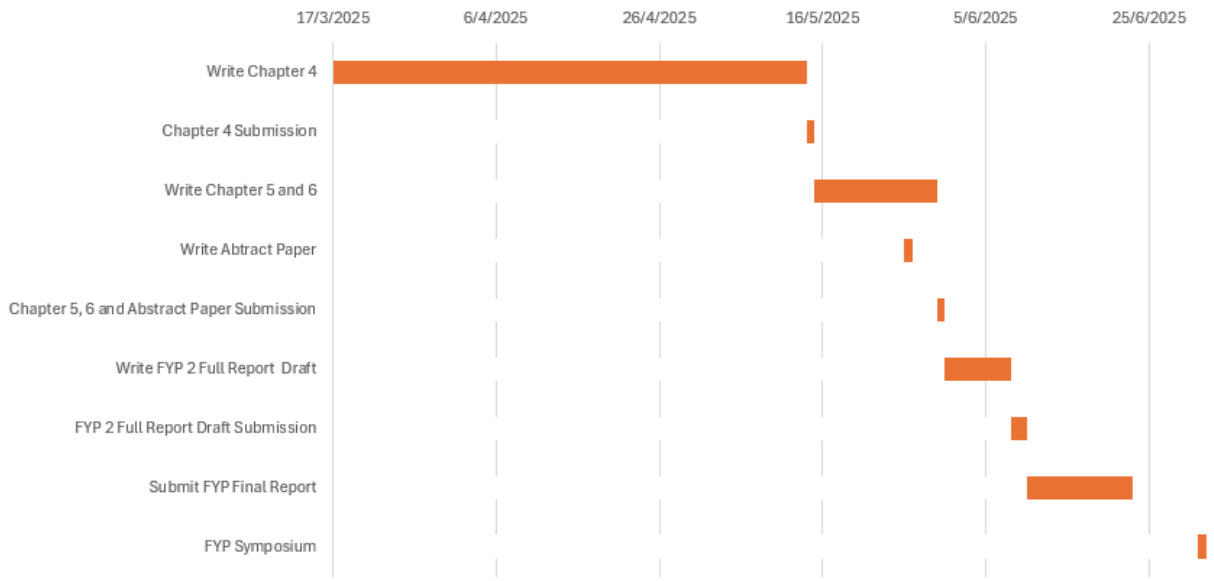



*Figure 1.3:* FYP 2 Milestone and Task Schedule

Based on the figure above, this Gantt chart will be focusing on FYP 2. The time to complete FYP 2 and the whole FYP is roughly four months from March 2025 to June 2025.

## 1.9 Project Outline

This project report is organized into six chapters, each focusing on a key part of the work: the introduction, literature review, requirement analysis and design, implementation and testing, system evaluation, and conclusion with future recommendations. Below is a summary of what each chapter covers:

### 1.9.1 CHAPTER 1: INTRODUCTION

This chapter introduces the background of the project, including the problem statement, objectives, and the methodology employed, specifically the Design Science Research Methodology (DSRM). It also defines the project scope, significance, expected outcomes, and presents the project schedule to track development progress.

### 1.9.2 CHAPTER 2: LITERATURE REVIEW

This chapter reviews relevant research and existing intrusion detection systems, focusing on signature-based IDS technologies. It compares various IDS approaches, highlighting their advantages and limitations. The chapter also discusses key tools and technologies used in this project such as Snort, and machine learning methods for improving detection accuracy.

### 1.9.3 CHAPTER 3: METHODOLOGY REQUIREMENT ANALYSIS AND DESIGN

This section explains the chosen methodology that guided the project's development, with an emphasis on DSRM. It outlines the system requirements and presents the overall design of the IDS, including Snort setup, creation of custom detection rules, integration with a machine learning model, and the design of the testing environment.

### 1.9.4 CHAPTER 4: IMPLEMENTATION AND TESTING

This chapter describes the step-by-step implementation of the IDS on a Linux platform. It covers the configuration of Snort and the application of custom rules, followed by details on how simulated attacks were used to test the system's response and accuracy.

### 1.9.5 CHAPTER 5: SYSTEM TESTING

This part focuses on the evaluation process of the IDS after full deployment. It discusses the results of combining Snort's signature-based detection with a machine learning classifier, assessing the system's ability to identify attacks like DoS, port scans, and SQL injection, while also reviewing its accuracy and rate of false positives.

### 1.9.6 CHAPTER 6: CONCLUSION AND FUTURE WORK

The final chapter summarizes the project outcomes and key insights gained. It also proposes areas for further development, such as extending the system to detect more attack types, improving machine learning integration, and enhancing its overall effectiveness in securing networks.

## 1.10 Summary

This chapter has highlighted the importance of developing a signature-based Intrusion Detection System (IDS) using Snort to address current network security challenges. It presented the project's objectives, scope, and methodology, emphasizing the need for a reliable system capable of detecting cyber-attacks. An overview of the expected outcomes was also provided. The next chapter will review existing IDS solutions and related technologies, compare their features and limitations, and justify the selection of tools used in this project's implementation.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Introduction

This chapter reviews several existing Intrusion Detection Systems (IDS) similar to the proposed Snort. The review focuses on their functionality, advantages, and disadvantages. A comparison table highlights the key differences and features of these systems. Additionally, the chapter explores tools and technologies essential to developing the proposed IDS and their contributions to its effectiveness. Furthermore, the integration of machine learning techniques, particularly the Random Forest algorithm, is explored to enhance Snort's detection capabilities.

## 2.2 Reviews on Similar Existing Systems

In this section, three existing intrusion detection systems have been selected for review and discussion to establish their relevance and effectiveness in network security. These systems are:

1. Suricata
2. Bro/Zeek
3. OSSEC

### 2.2.1 Suricata


*Figure 2.1:* Suricata Logo (Suricata, 2023)

Suricata is an open-source IDS that supports both signature-based and anomaly-based detection. According to Hoover (2022), its multi-threaded architecture ensures efficient handling of large network traffic volumes. Suricata also provides advanced features, such as protocol analysis, file extraction, and inline intrusion prevention, making it suitable for high-performance and complex networks. However, its higher resource usage and steeper learning curve compared to Snort can make it challenging for beginners. Suricata's scalability and advanced capabilities make it an excellent choice for large-scale environments. In this project, Snort was selected over Suricata because it is lightweight, efficient, and excels at detecting known attack patterns like Denial of Service (DoS) and SQL injection.

### 2.2.2 Bro/Zeek


*Figure 2.2:* Zeek Logo (Ziobro, 2022)

Zeek (formerly known as Bro) is an anomaly-based IDS that focuses on detecting unusual network activities by analysing behaviour and logging events. Lewandowska (2024) emphasizes its effectiveness in identifying novel and sophisticated attacks that signature-based systems may overlook. Zeek offers robust scripting capabilities, allowing users to create custom detection rules for specific environments.

While Zeek is highly effective in complex environments, its steep learning curve and smaller community compared to systems like Snort and Suricata can pose challenges. However, Zeek's ability to detect advanced threats makes it suitable for environments requiring deep analysis. In contrast, this project leverages Snort's signature-based approach to detect specific, known attacks, which is more suitable for our objectives of monitoring common network threats.

### 2.2.3 OSSEC


*Figure 2.3:* OSSEC Logo (OSSEC, 2023)

OSSEC is a host-based intrusion detection system (HIDS) that focuses on monitoring individual devices for unauthorized changes, log analysis, and file integrity. Anafcheh (2018) highlight its strengths in log monitoring and ensuring file integrity, making it an excellent addition to a hybrid security framework. However, its inability to monitor real-time network traffic limits its effectiveness in detecting network-wide attacks. OSSEC is most effective when paired with a network-based IDS like Snort to provide comprehensive security. OSSEC's capability to integrate with cloud environments enhances its utility in monitoring virtualized infrastructures, complementing broader network-level security measures.

### 2.2.4 Comparison of the Existing System with Snort

*Table 2.1:* Comparisons of Features between Existing Systems and Snort

| | Proposed System | Existing System | | |
|---|---|---|---|---|
| | Snort | Suricata | Bro/Zeek | OSSEC |
| Type | Network-based IDS (NIDS) | Network-based IDS (NIDS) | Network-based IDS (NIDS) | Host-based IDS (HIDS) |
| Detection Method | Signature-based | Signature + anomaly-based | Anomaly-based | Signature-based |
| Performance | Lightweight, efficient | Handles high traffic | High resource usage | Endpoint monitoring |
| Customization | Customizable rules | Custom rules and protocol analysis | Advanced scripting capabilities | Limited to file integrity/log monitoring |
| Best For | Detecting known attacks (DoS, SQL, port scans) | High-traffic networks, advanced features | Detecting novel attacks, traffic analysis | Endpoint security, complementing network IDS |

Table 2.1 shows the comparison between existing systems and Snort. While other systems like Suricata and Zeek offer advanced features, Snort's lightweight design, focus on signature-based detection, and customizability make it more suitable for detecting specific attack types like DoS and SQL injection. This table demonstrates how the proposed system integrates the strengths of existing solutions while addressing their limitations.

## 2.3 Machine Learning in Intrusion Detection (Random Forest)

Traditional signature-based systems, such as Snort, effectively detect known attacks by matching traffic patterns with predefined rules. However, these systems face challenges in identifying zero-day threats or novel attacks that are not yet included in the signature database (Pramudya & Alamsyah, 2022).

To overcome these limitations, this project integrates a machine learning (ML) component by analysing Snort-generated alert logs. Rather than embedding the ML model into Snort's real-time detection pipeline, an offline classification approach is used. Snort alerts saved in alert file format are parsed into structured datasets using Python tools such as Pandas and Scikit-learn. This offline method simplifies integration, avoids real-time performance overhead, and allows iterative experimentation and tuning.

The machine learning module is executed externally on the host machine, instead of within the Snort virtual machine. This external setup was chosen for several reasons. First, it avoids the complexity of modifying Snort's internal processes or dealing with performance limitations when running both detection and classification in real time. Second, the external approach enables greater flexibility to test, retrain, and adjust the model without affecting the IDS's stability. Finally, this method mirrors practical use cases in real-world deployments where data from detection systems is sent to external analytics modules for further processing (Aeraj & Leghris, 2023).

The Random Forest algorithm is selected for its robustness in classification tasks. It builds multiple decision trees and combines their predictions to improve accuracy and minimize overfitting. According to Preethi et al. (2024), Random Forest achieved an accuracy of 99.34% in intrusion detection scenarios, outperforming other classifiers.

In this project, features such as alert type, timestamp, source IP, and destination IP are extracted and vectorized for classification. The model classifies traffic into categories such as Port Scan, DoS, SQL Injection, or benign traffic. Although the model is trained on simulated data, the architecture is adaptable to real-world datasets and live deployments.

Key benefits of the hybrid Snort + ML approach include:

- **Reduced False Positives:** ML helps distinguish true threats from benign anomalies by learning from labelled alert data (El Aeraj et al., 2024).
- **Scalability and Adaptability:** The model can continuously learn from new data to improve detection accuracy over time (Chaithanya et al., 2023).

This integration demonstrates a practical path toward intelligent intrusion detection, especially valuable for resource-constrained environments seeking adaptive security solutions.

## 2.4 Reviews of Tools and Technology

This section reviews the core tools and technologies utilized in the implementation of a signature-based Intrusion Detection System (IDS), focusing on their functions, benefits, and relevance to the project.

### 2.4.1 Platform

This section introduces the platform used for developing, testing, and monitoring network IDS.

#### 2.4.1.1 Linux Mint

Linux Mint was selected as the operating system due to its stability, ease of use, and strong support for networking and security tools. As a Linux-based platform, it provides a user-friendly interface, making it accessible to users with varying levels of experience in Linux

environments. Linux Mint offers a reliable and compatible environment for the installation and operation of essential open-source security tools such as Snort, the robust command-line interface facilitates the configuration and monitoring of the Intrusion Detection System (IDS), ensuring optimal system performance, even on standard hardware.

### 2.4.2 Programming Language

#### 2.4.2.1 Phyton 3

Python is the primary programming language for this project due to its simplicity, versatility, and strong support for cybersecurity and AI tasks. Released in 1991 by Guido van Rossum, Python has become one of the most widely used open-source languages globally. It is known for its clean syntax and readability, making it ideal for rapid development and testing. Python supports a wide variety of libraries relevant to cybersecurity, such as Scikit-learn and Pandas, It is used to parse Snort logs, extract features, train classifiers, and evaluate model performance. The language's strong community and wealth of documentation make troubleshooting and enhancement much more manageable.

#### 2.4.2.2 Bash

Bash (Bourne Again Shell) is the default command-line shell on most Linux systems and is used extensively in this project to automate routine tasks. Bash scripts are created to start Snort, rotate logs, configure iptables, and launch attack simulations. Bash allows for seamless integration with system utilities and is useful for managing IDS components without requiring a GUI. It provides control over network interfaces and logging mechanisms, making it indispensable for real-time IDS deployment. Its scripting capabilities ensure repeatable, consistent execution of key tasks.

#### 2.4.2.3 Snort Rule Syntax

Snort rule syntax is the language used to define detection rules for Snort. These rules are composed of header and options fields that describe the conditions under which an alert should be triggered. Writing effective rules requires understanding of network protocols, packet structures, and common attack signatures. Custom rules in this project are crafted to detect port scans, ICMP floods, and SQL injection attempts. The flexibility of Snort's rule syntax allows the IDS to be tailored to specific threats. Well-written rules significantly enhance the accuracy and responsiveness of the system.

### 2.4.3 Tools and Libraries

This section reviews the specific tools and libraries employed in the project.

#### 2.4.3.1 Snort 2.9


*Figure 2.4:* Snort Logo (CyberHub, 2023)

Snort is an open-source, lightweight, and highly configurable Network Intrusion Detection System (NIDS) developed by Sourcefire. It operates by capturing network packets in real time and analysing them against a set of predefined rules to detect malicious activity such as port scans, denial-of-service (DoS) attacks, and SQL injection attempts. Snort is signature-based, meaning it detects threats by matching packet content to known attack signatures. Its flexible rule syntax

allows for the creation of custom rules tailored to specific network environments. In this project, Snort is responsible for monitoring and generating alerts when suspicious patterns are detected in the network traffic.

#### 2.4.3.3 Nmap

Nmap (Network Mapper) is a security scanner used to discover hosts and services on a network. It plays a critical role in simulating reconnaissance attacks, especially various types of port scans. These tests are essential for validating the IDS's ability to detect intrusions at the reconnaissance phase. Nmap's versatility allows users to perform basic scans or complex custom scan patterns. It generates traffic that mimics real-world attacker behaviour, providing a realistic testing ground for Snort. Its CLI interface also makes it easy to automate within the testing workflow.

#### 2.4.3.4 Hping3

Hping3 is a command-line oriented TCP/IP packet assembler and analyser. It is used in this project to simulate complex network attacks, such as TCP SYN floods and ICMP-based denial-of-service (DoS) attacks. Unlike Nmap, Hping3 offers more control over packet headers, enabling the crafting of customized attack traffic. This makes it useful for testing how well Snort can handle various types of DoS patterns. Hping3 can also perform traceroute-like operations and firewall testing, adding depth to the IDS evaluation. Its inclusion ensures the system is tested against sophisticated, low-level attack techniques.

#### 2.4.3.5 Curl

Curl is a command-line tool for transferring data with URLs, supporting protocols like HTTP, HTTPS, FTP, and more. In the IDS project, it is used to simulate web traffic, such as HTTP GET and POST requests, which can include SQL injection or command injection payloads. This

helps test the IDS's ability to detect application-layer attacks. Curl is highly scriptable, making it ideal for batch testing scenarios. It supports header customization, cookie manipulation, and payload injection, allowing detailed control over each request. Its role complements tools like Hping3 by simulating different attack vectors.

#### 2.4.3.6 Nano

Nano is a terminal-based text editor used for editing configuration files, Snort rules, and Bash scripts directly in the Linux terminal. It is chosen for its simplicity and low system overhead, making it suitable for quick edits without the need for GUI-based tools. In the IDS project, Nano is used to write and modify Snort's rule files and startup scripts. Its intuitive interface allows users to navigate, search, and edit files efficiently. Nano's availability by default on most Linux distributions ensures it is always accessible during setup and troubleshooting.

#### 2.4.3.7 Pandas

Pandas is a powerful Python library used for data analysis and manipulation. It provides data structures such as DataFrames, which are ideal for handling tabular alert data generated by Snort. In this project, Pandas is used to read parsed alert files, clean the data, and prepare it for machine learning models. It simplifies tasks like filtering, grouping, and aggregating alerts by type, IP address, or timestamp. With Pandas, large volumes of log data can be efficiently processed and transformed into actionable insights. Its flexibility supports both exploratory data analysis and feature engineering.

#### 2.4.3.8 Scikit-learn

Scikit-learn is a machine learning library in Python that provides simple and efficient tools for data mining and analysis. In this project, it is used to train and test models that classify Snort alerts as normal or malicious. Algorithms such as Decision Trees, Random Forests, and Naive

Bayes are implemented using Scikit-learn. The library's modular design and consistent API make it easy to plug in different models and compare their performance. It also includes utilities for model evaluation, such as confusion matrices and cross-validation. Scikit-learn is essential for adding intelligence to the IDS.

#### 2.4.3.9 re (Regex)

The re library in Python provides support for regular expressions, which are used for searching and extracting patterns from text. In this project, re is used to parse the alert file and extract important features such as timestamps, source IPs, destination IPs, and alert types. Regex makes it possible to handle inconsistent formatting in raw log files and isolate meaningful components. This preprocessing is a critical step before data can be analysed or fed into machine learning algorithms. The re module allows for quick, flexible parsing without requiring external tools.

#### 2.4.3.10 CountVectorizer

CountVectorizer is a feature extraction tool from the Scikit-learn library that converts text data into a matrix of token counts. In this IDS project, it is used to transform textual Snort alert messages into numerical vectors that can be fed into machine learning models. This enables the classification of alerts based on their content. CountVectorizer supports custom tokenization and n-grams, providing flexibility in feature representation. By turning unstructured data into structured input, it bridges the gap between raw alert logs and predictive analytics. It is essential for building text-based threat classification models.

#### 2.4.3.11 Matplotlib/Seaborn

Matplotlib and Seaborn are Python libraries used for data visualization. They are optional but useful in this project for plotting graphs such as confusion matrices, alert distribution, or time-series trends. Matplotlib provides low-level control for building custom visualizations, while Seaborn offers high-level, aesthetically pleasing plots with fewer lines of code. These tools help users better understand the performance of the IDS and machine learning models. Visualization makes it easier to identify patterns, anomalies, and the effectiveness of rule-based or ML-based detection. They are especially valuable for reporting and presentation.

## 2.5 Summary

This chapter presented a review of related IDS technologies, highlighting their detection methods, architectures, and suitability for different network environments. While Snort excels in detecting known threats using a signature-based approach, its limitations against unknown threats necessitate enhancement. The integration of the Random Forest machine learning algorithm allows for improved accuracy and adaptability. Tools such as Linux Mint, Python, and network traffic generators form the backbone of this hybrid IDS solution, providing a practical and scalable approach to modern network security.

# CHAPTER 3: METHODOLOGY, REQUIREMENT ANALYSIS, AND DESIGN

## 3.1 Introduction

In this chapter, the methodology used to develop the Signature-based Intrusion Detection System (IDS) using Snort will be discussed, including an explanation of the procedures and phases involved in the design and implementation. The chosen methodology for this project is the Design Science Research Methodology (DSRM), which is well-suited for the development of innovative technical solutions. DSRM provides a structured approach that emphasizes iterative design, development, and evaluation. This methodology will guide the project through the process of designing, implementing, testing, and refining the IDS to ensure it meets the requirements for detecting and responding to network security threats effectively.

## 3.2 Design Science Research Methodology (DSRM)

Design Science Research Methodology (DSRM) is the methodology chosen for the development of the Signature-based Intrusion Detection System (IDS) using Snort. DSRM is suitable for this project as it provides a structured approach to solving technical problems through iterative design and evaluation. Each phase of DSRM will be applied to the project, from the identification of the problem to the communication of the project outcomes.

DSRM Phases:

### 3.2.1 Identify the Problem

Modern organizations face increasing cyber threats targeting network infrastructure. Signature-based IDS tools like Snort effectively detect known attacks by matching traffic against predefined rules but are limited by their inability to detect zero-day or unknown attacks and tend to generate excessive false positives. These limitations hinder timely and accurate threat response.

### 3.2.2 Define the Solution

This project presents a hybrid IDS framework integrating Snort's signature-based detection with a machine learning classifier. Custom Snort rules detect specific attack types such as Denial-of-Service (DoS), port scanning, and SQL injection. Simultaneously, a Random Forest-based machine learning model analyzes Snort alert logs to reduce false positives and identify anomalous behaviors indicative of unknown threats.

### 3.2.3 Design and Development

The system architecture centers on Snort 2.9.20 deployed on a Linux Mint virtual machine, configured with custom rules tailored to detect specific attack signatures. A Python-based machine learning module preprocesses Snort alerts and employs the Random Forest algorithm to classify alerts. Supporting tools such as Nmap and Hping3 are integrated for attack simulation.

### 3.2.4 Demonstration

The system is deployed in a virtualized environment where simulated attacks are executed. Snort's detection and alerting capability, combined with machine learning classification, is demonstrated by capturing and analyzing attack traffic.

### 3.2.5 Evaluation

Performance evaluation involves measuring detection accuracy, false positive rate, and alert response times. The effectiveness of the machine learning model is assessed using metrics such as precision, recall, and F1-score, guiding iterative refinement.

### 3.2.6 Communication

Comprehensive documentation of the project methodology, system design, implementation details, and evaluation results is prepared to facilitate knowledge sharing and future improvements.

## 3.3 Requirement Analysis

The requirements for the IDS are categorized into functional, technical, and environmental aspects to ensure comprehensive planning and implementation.

### 3.3.1 Functional Requirements

The functional requirements of the IDS focus on its primary capabilities:

- **Real-time Network Monitoring:** Continuously capture and analyse network traffic for security events.
- **Threat Detection and Alerting:** Detect and alert on attacks such as DoS, port scanning, and SQL injection via custom Snort rules.
- **False Positive Reduction:** Apply machine learning classification on Snort alerts to distinguish true threats from benign anomalies.
- **Alert Logging:** Store alerts in a structured fast alert format (alert file) for subsequent processing.

- **Rule Customization:** Support creation and updating of Snort detection rules to adapt to emerging threats.
- **Scalability:** Ensure system performance remains effective under increasing network loads.

### 3.3.2 Technical Requirements

The technical requirements define the tools, technologies, and configurations essential for the IDS:

- **Operating System:** Linux Mint VM for stable and compatible deployment of IDS and tools.
- **Intrusion Detection Engine:** Snort 2.9.20 configured with both predefined and custom rules.
- **Machine Learning Environment:** Python 3 environment with libraries including Scikit-learn for Random Forest implementation, Pandas for data manipulation, and regex for alert parsing.
- **Attack Simulation Tools**: Nmap, Hping3, and sqlmap to generate realistic attack traffic for testing.
- **Alert Format:** Use Snort's alert file mode for efficient log generation and parsing.
- **Hardware Specifications:** Minimum system specifications include 8 GB RAM, quad-core CPU, and 256 GB storage to ensure smooth operation.

### 3.3.3 Environmental Requirements

The environmental requirements address the operational and testing constraints of the IDS:

- **Virtualized Platform**: Use of VirtualBox or equivalent to isolate attacker and defender environments, ensuring safe and controlled testing.

- **Open-source Tools and Community Support:** Leverage publicly available Snort rules, Python libraries, and security tools for ease of development and maintenance.

## 3.4 System Design

The system design of the Signature-based Intrusion Detection System (IDS) integrates Snort, a signature-based detection engine, with an external machine learning (ML) module to enhance detection accuracy, reduce false positives, and detect previously unknown threats. This hybrid design provides the advantages of both signature-based detection and anomaly detection, which is critical for modern network security.

The system is modular and scalable, with the IDS being housed on a Linux Mint-based VM for Snort, while the external machine learning module processes alert logs generated by Snort. This design allows for efficient processing, greater flexibility, and the ability to integrate advanced anomaly detection in real-time.

### 3.4.1 Key Components of the System

The architecture of the system is broken down into several layers and components, each responsible for distinct tasks in network traffic monitoring, detection, classification, and logging.

#### 3.4.1.1 Network Traffic Capture Layer (Snort VM)

This layer is responsible for capturing all incoming and outgoing network traffic on the monitored interface.

- **Snort IDS Engine:** The Snort engine is configured on a Linux Mint virtual machine (VM) to monitor network traffic. Snort captures the packets passing through the network interface and analyzes them against predefined and custom attack signatures.

- **Simulated Attacker VM:** A separate virtual machine generates various attack scenarios using tools like Nmap, hping3, and curl to test the IDS's detection capabilities.

#### 3.4.1.2 Detection Layer (Snort VM)

The detection layer is where the actual detection of known attacks occurs using Snort's signature-based engine.

- **Signature-based Detection:** Snort applies custom and predefined rules to the captured network packets. Alerts are triggered when traffic matches attack signatures such as port scans, DoS (ICMP flood), or SQL injection attempts. The alerts are logged in alert file format for easy access.

- **Custom Rules**: Snort's rules are tailored to detect specific attack types relevant to the system's objectives (DoS, port scanning, SQL injection).

#### 3.4.1.3 Alerting and Logging Layer (Snort VM)

Once Snort detects potential threats, it generates alerts and logs the activity for further review.

- Fast Alert Mode: Snort logs alerts in the alert file format, optimized for easy access. Alerts are stored in a directory and can be accessed for further processing or investigation.

- Log Access: Alerts can be reviewed by administrators in real time using commands like tail -f or by accessing log files directly.

#### 3.4.1.4 Machine Learning Layer (External System)

The machine learning layer is responsible for analyzing Snort-generated logs to classify alerts and reduce false positives.

- **Alert Parsing and Feature Extraction:** The external Python-based machine learning module will process Snort's alert file logs. It extracts relevant features from the alerts, such as the alert type, source IP address, destination IP, timestamp, etc.

- **Model Training and Classification:** The machine learning model, using algorithms such as Random Forest, is trained externally on labeled attack data. This model processes the features extracted from the logs to classify alerts as benign or malicious.

- **External Processing:** The machine learning module is deployed externally from the Snort VM, meaning it doesn't directly run on the IDS VM but communicates with it to access logs. This helps in offloading the machine learning processing to a dedicated external system, thus ensuring better performance and scalability.

#### 3.4.1.5 Logging and Visualization Layer

This layer provides real-time access to logs and visualizations of the alerts, making it easier for administrators to understand the system's performance and attack patterns.

Python Graphs: Alternatively, Python scripts using matplotlib or seaborn can generate graphs that visually represent the frequency and types of attacks over time.

### 3.4.2 Data Flow and Interaction

The system operates as a multi-layered architecture where data flows through the layers as follows:

1. **Network Traffic Capture**

   Network traffic is captured by Snort running on the Snort VM. Snort decodes the network packets and compares them against predefined and custom rules to identify any suspicious behavior.

2. **Alert Generation**

   If a packet matches any rule in Snort, an alert is generated and logged in the alert file. The alert contains information about the type of attack, timestamp, source and destination IPs, etc.

3. **Machine Learning Processing**

   The alert log files are sent or accessed by an external machine learning module. This module parses the logs, extracts features, and uses the Random Forest model to classify each alert as either benign or malicious.

4. **Alert Classification**

   Based on the classification results, the external machine learning module outputs the status of the alerts (true positive, false positive, etc.). These classified alerts are stored for further analysis and review.

5. **Alert Review and Visualization**

   System administrators can view the alerts either through the command-line interface (CLI) or through optional reporting and visualization tool Python scripts.

### 3.4.3 System Architecture Diagram

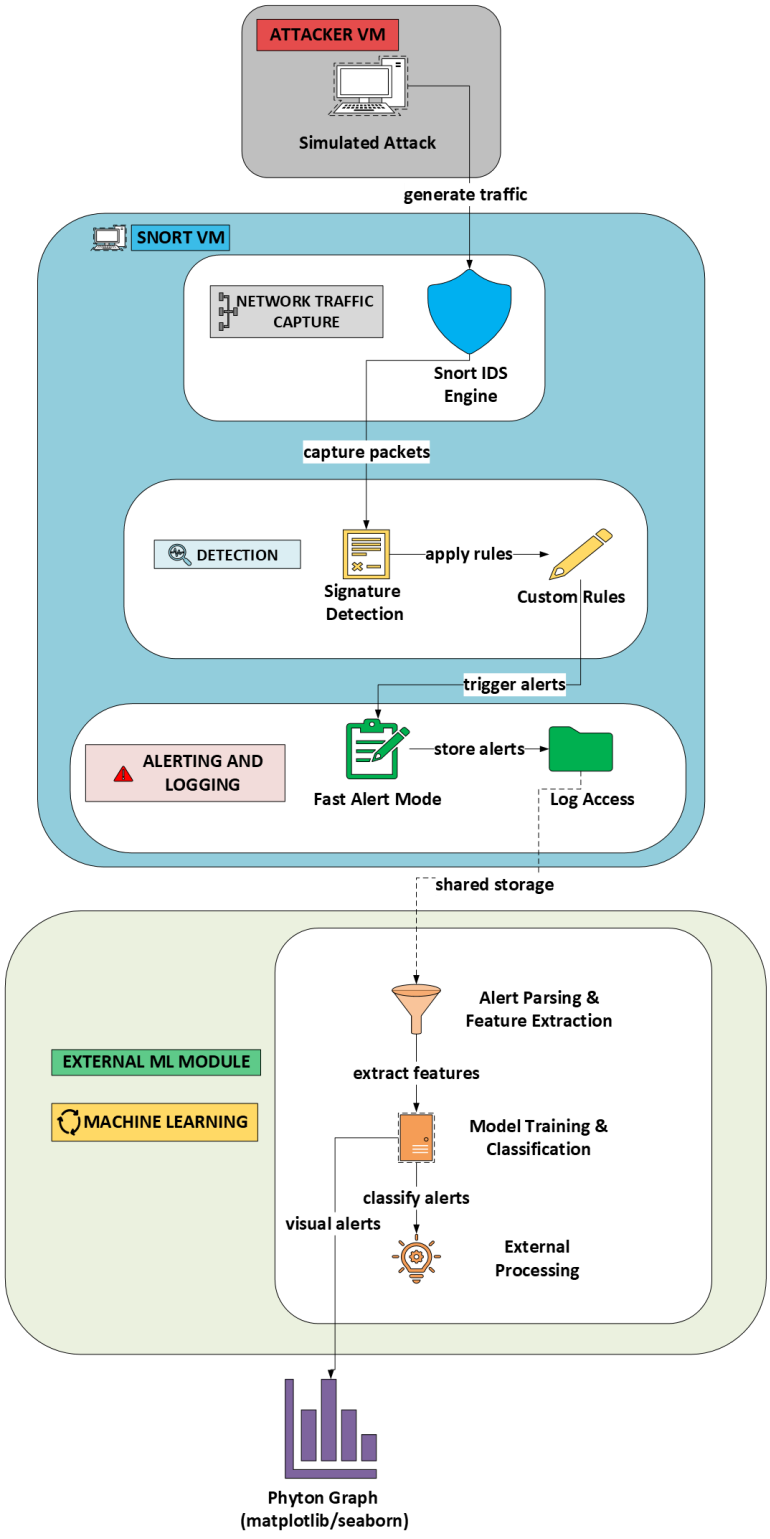



*Figure 3.1:* System Architecture of Signature-Based IDS with Machine Learning Integration

This diagram illustrates the system architecture of a Snort-based Intrusion Detection System (IDS) integrated with a machine learning module. The system consists of three main components the Attacker VM, which simulates malicious network traffic, the Snort VM, which captures and analyzes packets using custom signature rules and the External ML Module. Snort detects threats and logs alerts in fast alert mode, which are then stored in shared storage. These alerts are parsed by the machine learning module, where features are extracted and classified using trained models to produce visual insights, thereby enhancing detection accuracy and threat understanding.

## 3.5 Implementation Plan

To align with the proposed system design, the following outlines a high-level implementation plan for the hybrid Intrusion Detection System (IDS), which integrates Snort's signature-based detection with a machine learning-based alert classification module.

- **Set up the virtual environment**

  Create two Linux Mint virtual machines: one for the Snort IDS server and one as the attacker. Use bridge networking for realistic traffic.

- **Install and configure Snort**

  Set up Snort 2.9.20 with custom rules to detect DoS attacks, port scanning, and SQL injection attempts.

- **Simulate attacks**

  Use tools like Nmap, Hping3, and sqlmap from the attacker VM to test Snort's detection capabilities.

- **Enable alert logging**

  Configure Snort to log alerts in fast alert format (alert file) for easy parsing.

- **Develop machine learning module**

  Create a Python-based classifier using Random Forest to analyze and classify Snort alerts.

- **Classify and review alerts**

  Use the ML module to reduce false positives and provide better insight into alert types.

- **Visualize results**

  Generate basic charts or summaries to represent attack types and classification outcomes.

## 3.6 System Scalability and Flexibility

By placing the machine learning module externally, the system is made more scalable and flexible. The separation of concerns between Snort's rule-based engine and the machine learning classifier ensures that each component can be scaled independently:

- **Scalability:** Snort can be deployed on multiple VMs or systems to monitor different parts of the network, while the machine learning module can be hosted on a dedicated server that processes alerts in real-time or batch mode.

- **Flexibility:** The machine learning module can be updated or retrained independently from Snort, enabling the system to adapt to emerging threats without downtime or disruption to the IDS.

## 3.7 Summary

This chapter detailed the methodology, requirement analysis, and system design of a hybrid signature-based IDS leveraging Snort with external machine learning processing. The system uses Snort for signature-based attack detection and Random Forest machine learning for alert classification and anomaly detection. The architecture allows the IDS to process traffic, generate alerts, and classify them externally for improved scalability, flexibility, and performance. The separation of machine learning from the Snort VM optimizes the system's performance and allows for continuous enhancement of the machine learning model without affecting the core IDS operations. This approach ensures that the system can detect both known and unknown threats efficiently and accurately.

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1 Introduction

This chapter details the implementation process of the proposed Signature-based Intrusion Detection System (IDS) using Snort, including its integration with a Machine Learning (ML) module for enhanced threat detection. It outlines the setup of the test environment, configurations of virtual machines, and the simulations of various cyberattacks used to validate the system. Furthermore, it covers the development of Snort rules and the extraction of alert data for machine learning analysis. Performance metrics such as detection accuracy and false positive rates are also addressed to evaluate the effectiveness of the system.

## 4.2 System Setup

### 4.2.1 Operating System Installation

The IDS environment was built using Linux Mint 20.2 Cinnamon Edition, selected for its stability and compatibility with Snort and other necessary tools. The setup was carried out within Oracle VirtualBox on a host machine.

**Steps:**

1. Downloaded the Linux Mint 20.2 Cinnamon ISO file from the official website.
2. Created a new virtual machine in VirtualBox with the following specifications:

   **Base Memory (RAM):** 5120 MB

   **Processor:** 8 cores

   **Network Adapter:** Bridged Adapter
3. Mounted the ISO file in the VM settings and started the installation.

The system was subsequently updated to ensure compatibility with Snort and supporting packages.

Updated the system:

```
sudo apt update && sudo apt upgrade -y
```

### 4.2.2 Virtual Machine Configuration

*Table 4.1:* Virtual Machine Configuration Details

| VM | Purpose | OS | Adapter Type | IP Address |
|---|---|---|---|---|
| VM 1 | Snort IDS (Server) | Linux Mint | Bridge Adapter | 172.20.10.4 |
| VM 2 | Attacker | Linux Mint | Bridge Adapter | 172.20.10.5 |

### 4.2.3 Network Connectivity

Network connectivity between VM1 and VM2 was verified using the following:

- ip a to confirm IP address assignments.
- ping command for connectivity testing.

Ip address for VM 1 Snort IDS (Server)

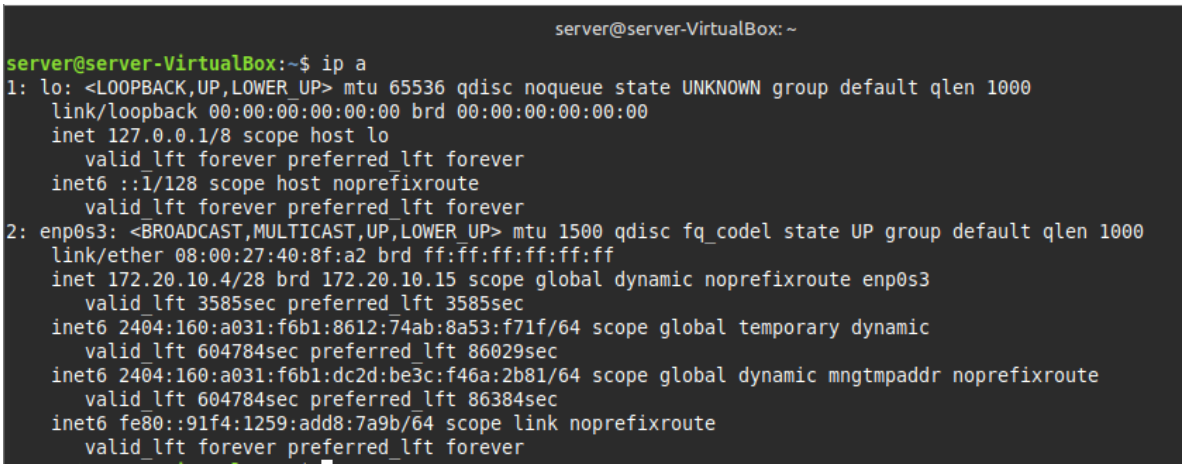



*Figure 4.1:* Snippet of ip a Command Output on VM1 (Snort IDS Server)

Ip address for VM2 Attacker

```
                                                    attacker@attacker-VirtualBox: ~
attacker@attacker-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:4a:8e:f9 brd ff:ff:ff:ff:ff:ff
    inet 172.20.10.5/28 brd 172.20.10.15 scope global dynamic noprefixroute enp0s3
       valid_lft 3573sec preferred_lft 3573sec
    inet6 2404:160:a031:f6b1:7a2b:7eb3:8e00:6399/64 scope global temporary dynamic
       valid_lft 604772sec preferred_lft 86094sec
    inet6 2404:160:a031:f6b1:c36b:ffc7:e037:2f3e/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 604772sec preferred_lft 86372sec
    inet6 fe80::8aa8:3959:4674:1d90/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

*Figure 4.2:* Snippet of ip a Command Output on VM2 (Attacker)

Ping results between VM 1 and VM 2

```
server@server-VirtualBox:~$ ping 172.20.10.5
PING 172.20.10.5 (172.20.10.5) 56(84) bytes of data.
64 bytes from 172.20.10.5: icmp_seq=1 ttl=64 time=1.50 ms
64 bytes from 172.20.10.5: icmp_seq=2 ttl=64 time=0.940 ms
64 bytes from 172.20.10.5: icmp_seq=3 ttl=64 time=0.754 ms
64 bytes from 172.20.10.5: icmp_seq=4 ttl=64 time=0.839 ms
64 bytes from 172.20.10.5: icmp_seq=5 ttl=64 time=0.878 ms
64 bytes from 172.20.10.5: icmp_seq=6 ttl=64 time=0.890 ms
64 bytes from 172.20.10.5: icmp_seq=7 ttl=64 time=0.919 ms
64 bytes from 172.20.10.5: icmp_seq=8 ttl=64 time=0.921 ms
64 bytes from 172.20.10.5: icmp_seq=9 ttl=64 time=1.20 ms
64 bytes from 172.20.10.5: icmp_seq=10 ttl=64 time=1.16 ms
64 bytes from 172.20.10.5: icmp_seq=11 ttl=64 time=1.59 ms
64 bytes from 172.20.10.5: icmp_seq=12 ttl=64 time=0.697 ms
```

```
attacker@attacker-VirtualBox:~$ ping 172.20.10.4
PING 172.20.10.4 (172.20.10.4) 56(84) bytes of data.
64 bytes from 172.20.10.4: icmp_seq=1 ttl=64 time=0.663 ms
64 bytes from 172.20.10.4: icmp_seq=2 ttl=64 time=1.33 ms
64 bytes from 172.20.10.4: icmp_seq=3 ttl=64 time=0.663 ms
64 bytes from 172.20.10.4: icmp_seq=4 ttl=64 time=0.764 ms
64 bytes from 172.20.10.4: icmp_seq=5 ttl=64 time=0.973 ms
64 bytes from 172.20.10.4: icmp_seq=6 ttl=64 time=0.937 ms
64 bytes from 172.20.10.4: icmp_seq=7 ttl=64 time=1.23 ms
64 bytes from 172.20.10.4: icmp_seq=8 ttl=64 time=1.47 ms
64 bytes from 172.20.10.4: icmp_seq=9 ttl=64 time=1.72 ms
64 bytes from 172.20.10.4: icmp_seq=10 ttl=64 time=0.570 ms
64 bytes from 172.20.10.4: icmp_seq=11 ttl=64 time=0.515 ms
64 bytes from 172.20.10.4: icmp_seq=12 ttl=64 time=1.73 ms
```

*Figure 4.3:* Snippet of Ping Test Between VM1 and VM2

## 4.3 VM 1 Snort Installation and Configuration

**Installed Snort using package manager**

```
sudo apt update

sudo apt install snort -y
```

**Verified installation using: snort -V**

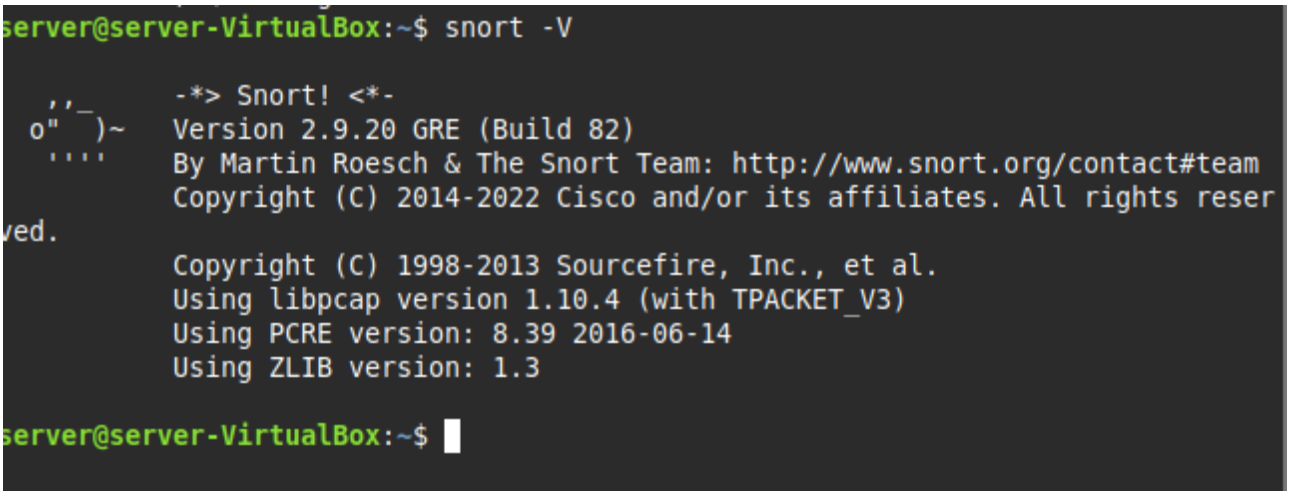



*Figure 4.4:* Snippet of snort -V Command to Verify Snort Installation

**Restart snort**

```
sudo /etc/init.d/snort restart
```

**To enable full alert logging, the Snort Debian configuration file was modified:**

**Opened**

```
sudo nano /etc/snort/snort.debian.conf
```

**Set the following option:**

```
DEBIAN_SNORT_OPTIONS= “-A full”
```

This option instructs Snort to use the full alert mode, which provides comprehensive alert details including timestamp, source and destination Ips. This is particularly useful for analysis, testing, and ensuring that all necessary information is logged for each detected event.

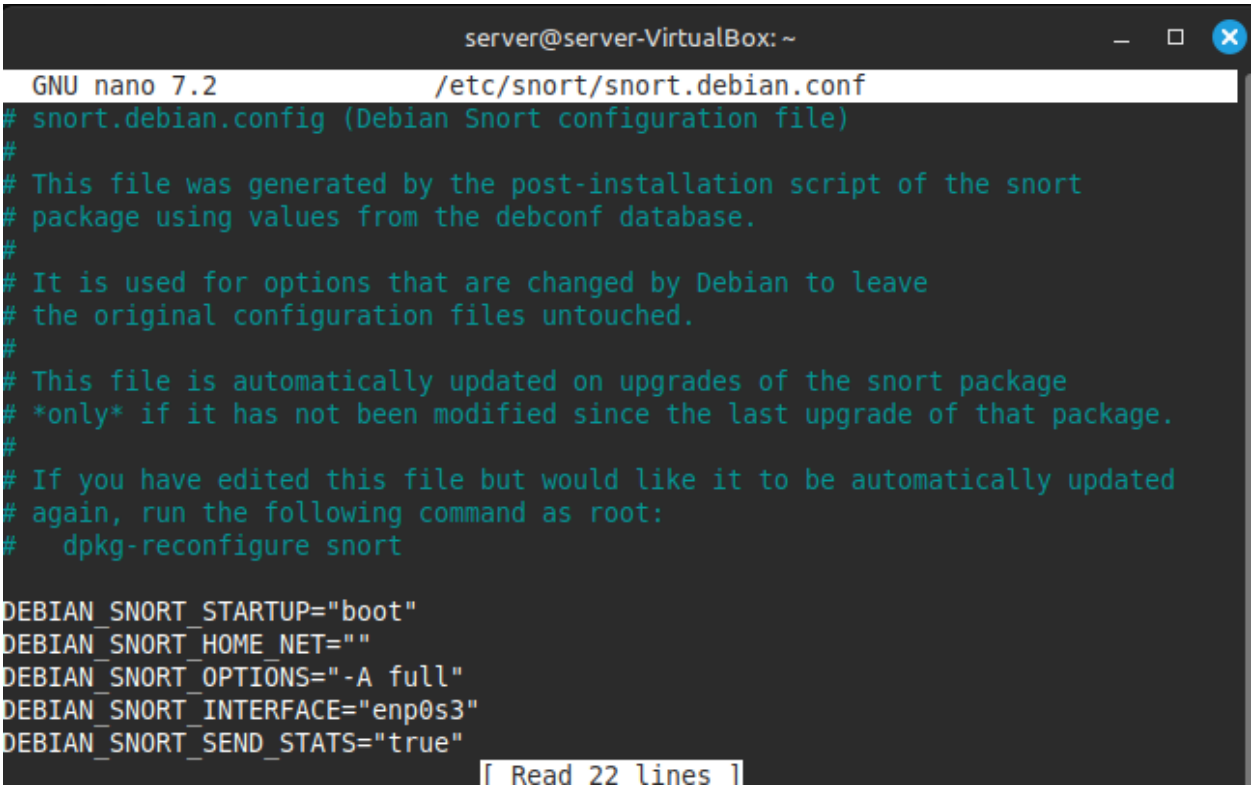



*Figure 4.5:* Snippet of snort.debian.conf Configuration File

## 4.4 Attack 1: Port Scanning Detection

**Snort was tested for reconnaissance attack detection using default rules**

```
sudo nano /etc/snort/rules/snmp.rules
```

**Restarted Snort with**

```
sudo systemctl restart snort
```

**Install Attack Tool (on VM2)**

```
sudo apt install nmap
```

**Launch Port Attack**

```
sudo nmap -sS 172.20.10.4
```

**Verified alert logs using**

```
sudo tail -f /var/log/snort/alert
```

Alerts confirming the scan were successfully detected.

## 4.5 Attack 2: DoS (TCP SYN Flood) Detection

**Rule Creation**

```
sudo nano /etc/snort/rules/dos.rules
```

**Add custom rules to detect DoS**

```
alert tcp any any -> $HOME_NET 80 (flags: S; msg: “Possible TCP SYN flood”;
detection_filter: track by_dst, count 50, seconds 10; sid:1000001;)
```

**Restarted Snort**

```
sudo systemctl restart snort
```

**Install Attack Tool (on VM2)**

```
sudo apt install hping3
```

**Launch DoS Attack**

```
sudo hping3 172.20.10.4 -p 80 -S --flood --rand-source
```

**Verified alert logs using**

```
sudo tail -f /var/log/snort/alert
```

Alerts were monitored and successfully logged.

## 4.6 Attack 3: SQL Injection Detection

**Install DVWA (on VM1)**

```
sudo apt install apache2 mysql-server php php-mysqli git
sudo systemctl enable apache2 mysql
sudo systemctl start apache2 mysql
cd /var/www/html
sudo git clone https://github.com/digininja/DVWA.git
cd DVWA/config
sudo cp config.inc.php.dist config.inc.php
sudo nano config.inc.php
```

**Set:**

```
$_DVWA[‘db_user’] = ‘root’;
$_DVWA[‘db_password’] = ‘ ‘;
```

**Create database:**

```
sudo mysql -u root
CREATE DATABASE dvwa;
EXIT;
```

**Edit PHP config:**

```
sudo nano /etc/php/*/apache2/php.ini
```

**Enable:**

```
allow_url_include = On
display_errors = On
```

**Set MySQL password:**

```
sudo mysql
USE mysql;
ALTER USER ‘root’@’localhost’ IDENTIFIED WITH mysql_native_password BY ‘toor’;
FLUSH PRIVILEGES;
_ _ _ _
```

**Update DVWA config with password:**

```
$_DVWA[‘db_password’] = ‘toor’;
```

**Restart services:**

```
sudo systemctl restart apache2
```

**Access DVWA at: http://172.20.10.4/DVWA/setup.php**

**Click “Create / Reset Database” and log in:**

```
username: admin
password: password
```

**Set security level to “Low”.**

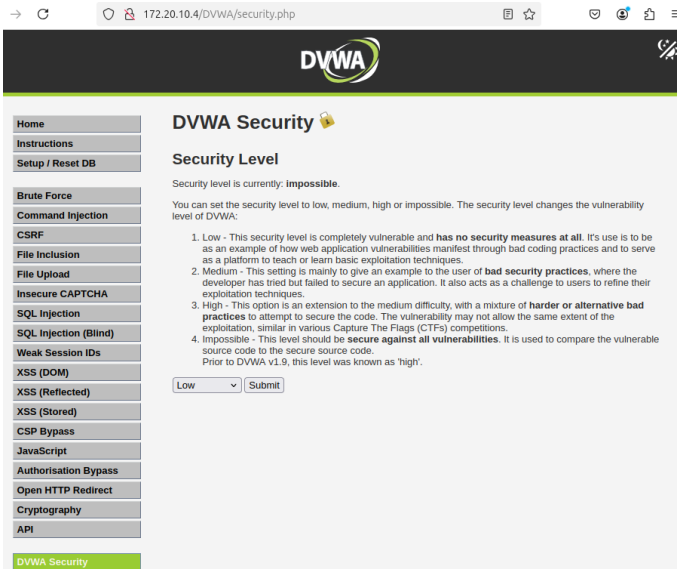



*Figure 4.6:* Snippet of Set Security Level DVWA

**Snort Rule for SQL Injection**

```
sudo nano /etc/snort/rules/sql.rules
```

**Add:**

```
alert tcp any any -> any 80 (msg: “Alert: SQL Injection Attempt”; content:”’OR ‘1’=’1”; http_uri; nocase; sid:1000005; rev:1;)
```

**Restarted Snort:**

```
sudo systemctl restart snort
```

**Launch SQL Injection Attack from VM2**

```
curl “http://172.20.10.4/DVWA/vulnerabilities/sqli/?id=1%27%20OR%20%271%27%3D%271&Submit=Submit”
```

**View Alerts**

```
sudo tail -f /var/log/snort/alert
```

Alert successfully triggered and logged

## 4.7 Machine Learning Integration

To enhance the detection and classification capabilities of the Snort-based IDS, a machine learning pipeline is integrated. This section outlines how Snort alerts are parsed and prepared for external machine learning classification.

To facilitate access to Snort alerts from an external host machine, a Shared Folder was configured on VM1 (Snort IDS) using VirtualBox. This allows the alert file generated by Snort to be accessed by Python scripts on the local machine for further analysis.

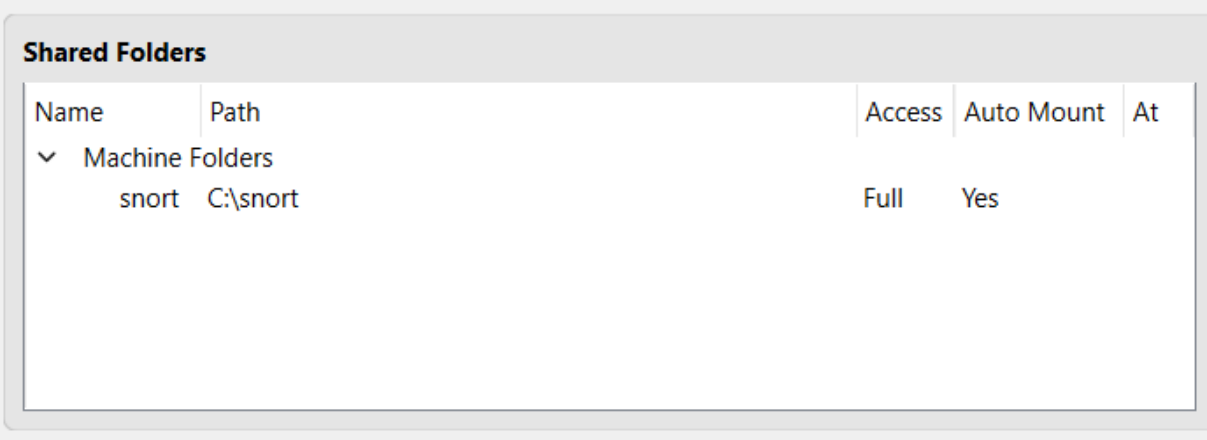



*Figure 4.7:* VirtualBox Shared Folder Configuration for Snort Alerts

This figure demonstrates the setup of a Shared Folder in VirtualBox, allowing Snort alerts generated on VM1 to be accessed by Python scripts on the host machine for machine learning analysis.

### 4.7.2 Required Python Libraries

Install the following Python libraries for data processing and machine learning:

```
pip install scikit-learn pandas numpy matplotlib seaborn regex joblib
```

### 4.7.3 Parsing the Snort Alerts

The Snort alert file is unstructured and requires parsing. A Python script parse_alerts.py was implemented to extract relevant fields from alert. This script reads the alert log, extracts important features like timestamp, alert message, source IP, and destination IP, and stores them in a structured CSV file.

```
import pandas as pd
import re
import os

def parse_alerts(file_path):
    alerts = []
    with open(file_path, 'r') as file:
        lines = file.readlines()

    i = 0
    while i < len(lines):
        line = lines[i].strip()

        # Detect start of an alert block
        if line.startswith('[**]') and line.endswith('[**]'):
            try:
                # Extract Alert Message
                alert_msg_match = re.search(r'\[\*\*\]\s*\[.*?\]\s*(.*?)\s*\[\*\*\]', line)
                alert_msg = alert_msg_match.group(1) if alert_msg_match else "Unknown"

                # IP line is usually the third line
                ip_line = lines[i + 2].strip() if i + 2 < len(lines) else ""

                # Extract timestamp
                timestamp_match = re.match(r'(\d+/\d+-\d+:\d+:\d+\.\d+)', ip_line)
                timestamp = timestamp_match.group(1) if timestamp_match else "Unknown"

                # Default IPs to Unknown
                src_ip = "Unknown"
                dst_ip = "Unknown"

                # Try IPv4 extraction first
                ip_match_v4 = re.search(r'(\d+\.\d+\.\d+\.\d+):\d+\s*->\s*(\d+\.\d+\.\d+\.\d+):\d+', ip_line)
                if ip_match_v4:
                    src_ip = ip_match_v4.group(1)
                    dst_ip = ip_match_v4.group(2)
                else:
                    # Try IPv6 pattern extraction
                    ip_match_v6 = re.search(r'(\S+)\s*->\s*(\S+)', ip_line)
                    if ip_match_v6:
                        src_ip = ip_match_v6.group(1)
                        dst_ip = ip_match_v6.group(2)

                alerts.append([timestamp, alert_msg, src_ip, dst_ip])
                i += 6  # Skip to next alert block
            except:
                i += 1
        else:
            i += 1

    df = pd.DataFrame(alerts, columns=["Timestamp", "Alert Message", "Source IP", "Destination IP"])
    return df

# Define alert file path
file_path = r'C:\snort\alert'

# Parse the alerts
alert_data = parse_alerts(file_path)

# Save to CSV
output_csv_path = os.path.join(os.path.dirname(file_path), 'parsed_alerts.csv')
alert_data.to_csv(output_csv_path, index=False)

# Preview
print(alert_data.head())
print(f"\nSaved to CSV: {output_csv_path}")
```

*Figure 4.8:* Code snippet from *parse_alerts.py* for extracting and structuring Snort alert data.

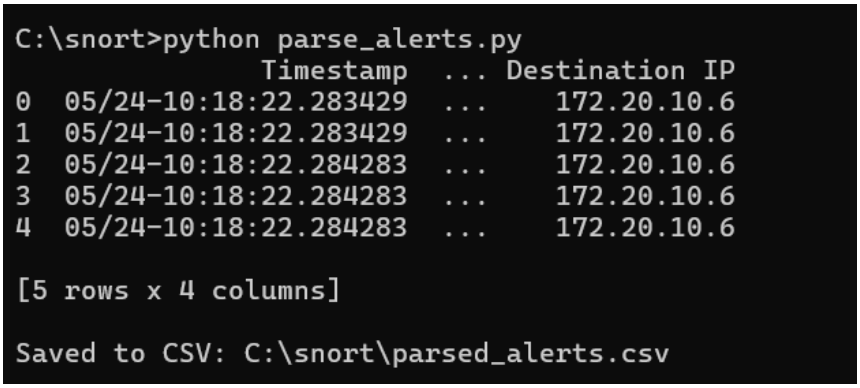



*Figure 4.9:* Structured alert data parsed from Snort logs (parsed_alerts.csv)

This figure displays a structured preview of Snort alerts parsed using the parse_alerts.py script. The parsed data includes fields such as timestamp, alert message, source IP, and destination IP, which are essential for feature extraction and machine learning classification.

### 4.7.4 Feature Engineering

The script feature_engineering.py transforms parsed alerts into numeric features for machine learning. This includes categorizing alert types into labels, encoding alert labels, extracting numeric values from IP addresses and balancing class distribution.

```
import pandas as pd
import re
from sklearn.preprocessing import LabelEncoder
import joblib

# === 1. Categorize alerts ===
def categorize_alert(message):
    message = message.lower()
    if 'syn flood' in message:
        return 'DoS'
    elif 'sql injection' in message:
        return 'SQL Injection'
    elif 'snmp' in message:
        return 'Recon'
    else:
        return 'Other'

# === 2. Load parsed alerts ===
df = pd.read_csv('parsed_alerts.csv')

# === 3. Apply alert categorization ===
df['Alert Type'] = df['Alert Message'].apply(categorize_alert)

# === 4. Encode alert type labels ===
label_encoder = LabelEncoder()
df['Alert Type Encoded'] = label_encoder.fit_transform(df['Alert Type'])
joblib.dump(label_encoder, 'alert_type_label_encoder.pkl')

# === 5. Convert IPs to numeric ===
df['Source IP Numeric'] = df['Source IP'].apply(
    lambda x: int(x.split('.')[0]) if isinstance(x, str) and '.' in x else -1
)
df['Destination IP Numeric'] = df['Destination IP'].apply(
    lambda x: int(x.split('.')[0]) if isinstance(x, str) and '.' in x else -1
)

# === 6. Balance the classes (min 1000 each) ===
min_samples = 1000
balanced_df = pd.DataFrame()

for label in df['Alert Type'].unique():
    class_df = df[df['Alert Type'] == label]
    if len(class_df) < min_samples:
        # Duplicate rows until reaching min_samples
        repeats = (min_samples // len(class_df)) + 1
        class_df = pd.concat([class_df] * repeats, ignore_index=True)
    balanced_df = pd.concat([balanced_df, class_df.head(min_samples)], ignore_index=True)

# === 7. Fill missing data (if any) ===
balanced_df = balanced_df.fillna(-1)

# === 8. Save to CSV ===
balanced_df.to_csv('alerts_preprocessed.csv', index=False)

# === 9. Summary ===
print("Feature engineering and class balancing complete.")
print("Class counts:")
print(balanced_df['Alert Type'].value_counts())
print(balanced_df.head(10))
print("Saved as: alerts_preprocessed.csv")
```

*Figure 4.10:* Code snippet from *feature_engineering.py* for feature engineering and data balancing.

Sample alerts_preprocessed.csv content (after feature engineering)

```
C:\snort>python feature_engineering.py
Feature engineering and class balancing complete.
Class counts:
Alert Type
DoS              1000
Other            1000
Recon            1000
SQL Injection    1000
Name: count, dtype: int64
               Timestamp  ... Destination IP Numeric
0  05/31-19:21:43.514803  ...                     172
1  05/31-19:21:43.515735  ...                     172
2  05/31-19:21:43.515735  ...                     172
3  05/31-19:21:43.515735  ...                     172
4  05/31-19:21:43.515736  ...                     172
5  05/31-19:21:43.515736  ...                     172
6  05/31-19:21:43.515736  ...                     172
7  05/31-19:21:43.515736  ...                     172
8  05/31-19:21:43.515736  ...                     172
9  05/31-19:21:43.516583  ...                     172

[10 rows x 8 columns]
Saved as: alerts_preprocessed.csv
```

*Figure 4.11:* Terminal output showing class distribution and sample preprocessed data.

### 4.7.6 Model Training and Evaluation

A Random Forest classifier is trained using preprocessed data. Performance is evaluated using accuracy and classification report.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import joblib

# === 1. Load preprocessed data ===
df = pd.read_csv('alerts_preprocessed.csv')

# === 2. Encode labels using LabelEncoder ===
label_encoder = LabelEncoder()
df['Alert Type Encoded'] = label_encoder.fit_transform(df['Alert Type'])

# Save label encoder
joblib.dump(label_encoder, 'alert_type_label_encoder.pkl')

# === 3. Feature and label setup ===
X = df[['Source IP Numeric', 'Destination IP Numeric']]
y = df['Alert Type Encoded']

# === 4. Train/test split ===
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# === 5. Train model ===
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# === 6. Evaluate ===
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy:.2f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_, zero_division=0))

# === 7. Save model ===
joblib.dump(model, 'alert_classifier_model.pkl')
print("\nModel saved as: alert_classifier_model.pkl")

# === 8. Confusion Matrix — Save directly as image ===
disp = ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred,
    display_labels=label_encoder.classes_,
    cmap='Purples',
    values_format='d'
)

plt.title("Confusion Matrix - Alert Classification")
plt.tight_layout()

# Save figure before showing
plt.savefig('confusion_matrix_alert_classification.png')
print("Confusion matrix saved as: confusion_matrix_alert_classification.png")

plt.show()
```

*Figure 4.12:* Code snippet from *train_model.py* for Random Forest training and performance evaluation.

### 4.7.6 Visualization of Model Performance

To understand the performance visually, the following scripts generate classification metrics bar chart.

Classification Metrics:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import joblib

# === 1. Load data & trained model ===
df = pd.read_csv('alerts_preprocessed.csv')
model = joblib.load('alert_classifier_model.pkl')
label_encoder = joblib.load('alert_type_label_encoder.pkl')

# === 2. Prepare features and true labels ===
X = df[['Source IP Numeric', 'Destination IP Numeric']]
y_true = df['Alert Type Encoded']
y_pred = model.predict(X)

# === 3. Generate classification report as a dictionary ===
report = classification_report(y_true, y_pred, output_dict=True, target_names=label_encoder.classes_, zero_division=0)

# === 4. Convert to DataFrame for plotting ===
metrics_df = pd.DataFrame(report).T

# Drop non-class rows (like accuracy, macro avg, weighted avg)
metrics_df = metrics_df.drop(['accuracy', 'macro avg', 'weighted avg'], errors='ignore')

# === 5. Plot bar chart ===
plt.figure(figsize=(10, 6))
ax = metrics_df[['precision', 'recall', 'f1-score']].plot(
    kind='bar', figsize=(10, 6), colormap='Purples', edgecolor='black'
)

# Add value labels on each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=8, color='black',
                xytext=(0, 2), textcoords='offset points')

# === 6. Finalize chart ===
plt.title('Classification Metrics per Alert Type', fontsize=14)
plt.xlabel('Alert Type', fontsize=12)
plt.ylabel('Score', fontsize=12)
plt.xticks(rotation=0)
plt.ylim(0, 1.1)  # Allow space for labels
plt.legend(loc='lower right')
plt.tight_layout()

# === 7. Save chart ===
plt.savefig('classification_metrics.png')
print("Saved as: classification_metrics.png")
plt.show()
```

*Figure 4.13:* Code snippet from *plot_classification_metrics.py* for visualizing precision, recall, and F1-score per alert type.

## 4.8 Summary

This chapter presented the end-to-end implementation of the Signature-based Intrusion Detection System using Snort. Through a structured testbed of virtual machines, various attacks port scanning, DoS, and SQL injection were successfully simulated and detected using Snort rules. Furthermore, a machine learning pipeline was integrated to classify alerts more intelligently. The successful execution of these tasks confirms that the core objectives of this project threat detection, rule creation, and ML enhancement have been met effectively.

# CHAPTER 5: SYSTEM TESTING

## 5.1 Introduction

This chapter presents the testing and validation phase of the developed Intrusion Detection System (IDS), which combines the Snort signature-based engine with a machine learning (ML) classifier. The objective is to evaluate the system's effectiveness, accuracy, and reliability in detecting and classifying various network intrusions. Testing focuses on the detection of specific attack types of Denial-of-Service (DoS), port scanning, and SQL injection using custom Snort rules, while the ML component is assessed based on its ability to reduce false positives and accurately identify benign traffic.

## 5.2 Testing Environment

All testing was conducted in a virtualized, isolated lab setup to maintain controlled and repeatable testing conditions. Two virtual machines (VMs) were configured using Oracle VirtualBox.

*Table 5.1:* Virtual Machine Configuration for System Testing

| **Virtual Machine (VM)** | **Role** | **Operating System** | **Tools Installed** |
|---|---|---|---|
| VM 1 | Snort IDS Server | Linux Mint | Snort 2.9.20, Apache, MySQL, DVWA, custom Snort rules |
| VM 2 | Attacker | Linux Mint | Nmap, hping3, sqlmap, curl |

Additionally, a shared folder was configured on VM1 to allow the host system to access Snort alert logs, enabling real-time parsing and analysis by Python-based ML scripts.

## 5.3 Test Scenarios and Cases

The IDS was evaluated against a set of defined test cases representing both known and ambiguous network activity. Each test aims to measure the system's detection and classification capabilities.

*Table 5.2:* Test Scenarios and Expected Classification Results

| **Test Case** | **Attack Type** | **Tool Used** | **Expected Detection and Classification** |
|---|---|---|---|
| TC1 | ICMP Flood (DoS) | hping3 | Detected by Snort, classified as DoS |
| TC2 | Port Scanning | Nmap | Detected by Snort, classified as Recon |
| TC3 | SQL Injection | sqlmap | Detected by Snort, classified as SQLi |
| TC4 | Unrecognized/Benign Traffic | curl/ping | Detected by Snort, classified as Other |

During TC4, benign network traffic (e.g., ICMP echo requests, normal HTTP communication) was also captured and labeled appropriately. This helped simulate realistic conditions and improve ML classification performance.

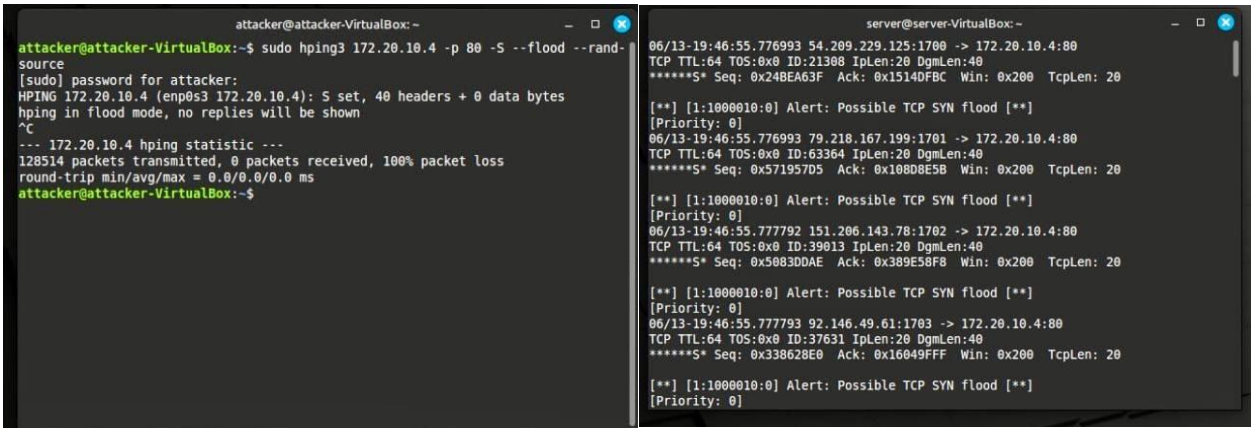



*Figure 5.1:* DoS attack simulation using hping3

The figure shows a simulation of a DoS attack using hping3, where ICMP packets flood the Snort IDS server. This verifies Snort's detection rule and classification as DoS.

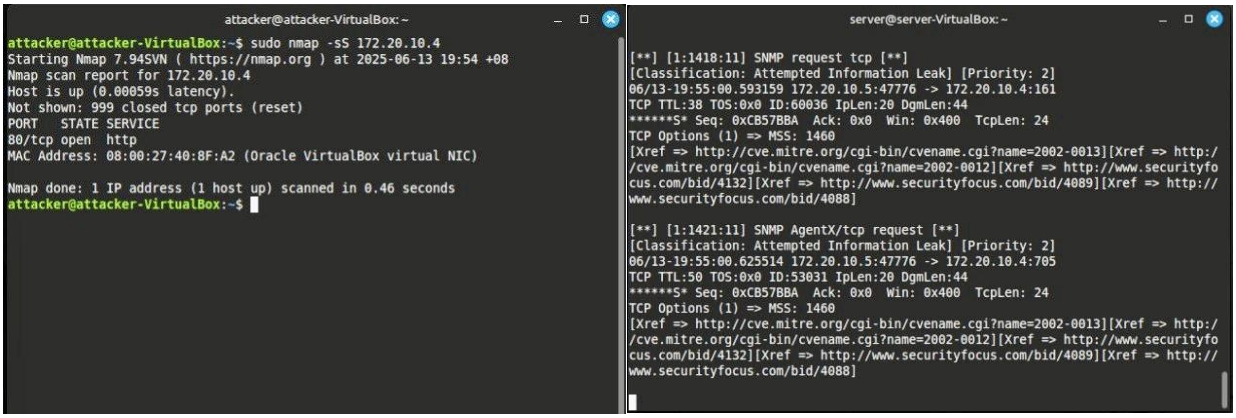



*Figure 5.2:* Nmap port scan targeting Snort VM

The figure illustrates a port scan using Nmap from the attacker VM, used to test detection and classification under the Recon label.

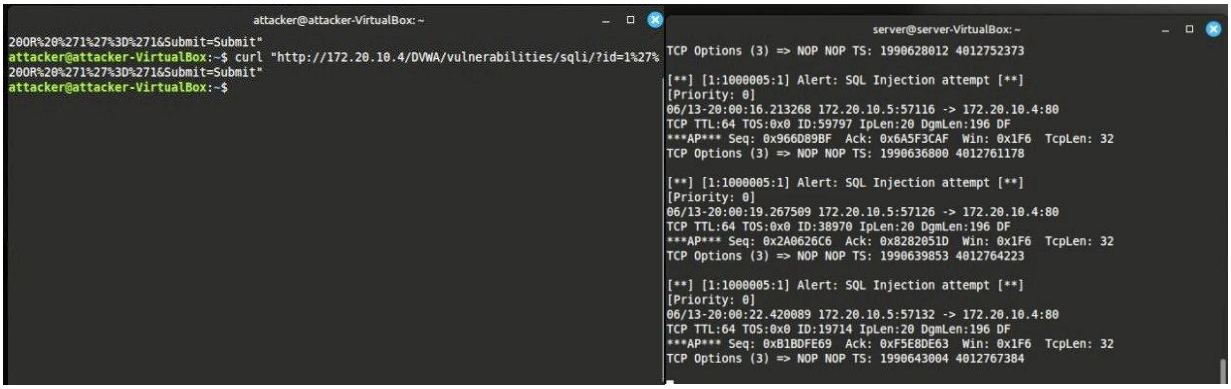



*Figure 5.3:* SQL Injection attack on DVWA using sqlmap

The figure demonstrates an SQL injection attack conducted using sqlmap targeting DVWA on the Snort VM, confirming correct detection and labeling as SQL Injection.

## 5.4 Functional Testing Results

Each simulated attack was executed, and Snort generated alerts, accordingly, capturing key metadata including source/destination IP, port, protocol, and rule message. The alerts were stored in the alert file format and processed by the ML module through the following pipeline:

1. *parse_alerts.py*: Extracts relevant alert fields (e.g., timestamp, IP, message).
2. *feature_engineering.py*: Converts alerts into machine-learning-compatible numerical features.
3. *train_model.py*: Trains a Random Forest classifier for classification.

```
[**] [1:1000010:0] Alert: Possible TCP SYN flood [**]
[Priority: 0]
05/24-10:18:47.164105 197.45.56.227:546 -> 172.20.10.6:80
TCP TTL:64 TOS:0x0 ID:48966 IpLen:20 DgmLen:40
******S* Seq: 0x7DDDD1E0  Ack: 0x2325F674  Win: 0x200  TcpLen: 20

[**] [1:1421:11] SNMP AgentX/tcp request [**]
[Classification: Attempted Information Leak] [Priority: 2]
05/24-10:19:11.752613 172.20.10.5:50513 -> 172.20.10.6:705
TCP TTL:38 TOS:0x0 ID:42726 IpLen:20 DgmLen:44
******S* Seq: 0xBD840919  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) => MSS: 1460
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0013]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0012]
[Xref => http://www.securityfocus.com/bid/4132]
[Xref => http://www.securityfocus.com/bid/4089]
[Xref => http://www.securityfocus.com/bid/4088]

[**] [1:1000005:1] Alert: SQL Injection attempt [**]
[Priority: 0]
05/24-10:31:17.074340 172.20.10.5:44026 -> 172.20.10.6:80
TCP TTL:64 TOS:0x0 ID:15774 IpLen:20 DgmLen:196 DF
***AP*** Seq: 0x2B103906  Ack: 0x313380E1  Win: 0x1F6  TcpLen: 32
TCP Options (3) => NOP NOP TS: 84194908 684554581
```

*Figure 5.4:* Snort alert log sample with entries for each attack type

The figure shows raw logs from Snort's alert file, including metadata like timestamps and alert messages. These logs were used as input for parsing and analysis.

## 5.5 Machine Learning Model Performance

The classification model was trained using labeled alert data generated from both simulated malicious traffic and naturally occurring benign activity. Evaluation metrics were calculated using scikit-learn to assess the effectiveness of the model in accurately classifying network alerts.

*Table 5.3:* Machine Learning Model Evaluation Metrics

| **Metric** | **Value** |
|---|---|
| Accuracy | 79% |
| Precision (Macro Avg) | 88% |
| Recall (Macro Avg) | 79% |
| F1-Score (Macro Avg) | 75% |

These results reflect strong overall classification performance, with high precision and balanced recall across most alert types. While detection accuracy was high for most categories, certain classes such as Recon (port scan) displayed relatively weaker recall, indicating room for improvement in the model's ability to capture specific types of network intrusions.

Terminal output displaying accuracy, precision, recall, and F1-score metrics from the machine learning model training script (*train_model.py*).

```
C:\snort>python train_model.py

Accuracy: 0.79

Classification Report:
              precision    recall  f1-score   support

         DoS       0.99      0.99      0.99       300
       Other       1.00      0.99      0.99       300
       Recon       1.00      0.19      0.31       300
SQL Injection      0.55      1.00      0.71       300

    accuracy                           0.79      1200
   macro avg       0.88      0.79      0.75      1200
weighted avg       0.88      0.79      0.75      1200


Model saved as: alert_classifier_model.pkl
Confusion matrix saved as: confusion_matrix_alert_classification.png
```

*Figure 5.5:* Model Training Performance Metrics Output

Although the model achieved an accuracy of 79%, this performance is considered acceptable for a signature-based IDS enhanced with machine learning, especially in a controlled and small-scale environment. According to a study by Almseidin et al. (2017), Random Forest achieved 93.1% accuracy on the KDD dataset, confirming its suitability for IDS applications. Similarly, Disha and Waheed (2022) reported that Random Forest classifiers achieved between 74% and 79% accuracy when trained on smaller IDS datasets. Furthermore, David & Adefolaju, (2021) demonstrated that Random Forest achieved competitive performance (around 78%) even without complex preprocessing techniques. These results suggest that accuracy around 79% is within the acceptable range for IDS research and sufficient to validate this project's objective of improving alert classification and reducing false positives.

**Confusion matrix**

To visualize classification performance across alert types:

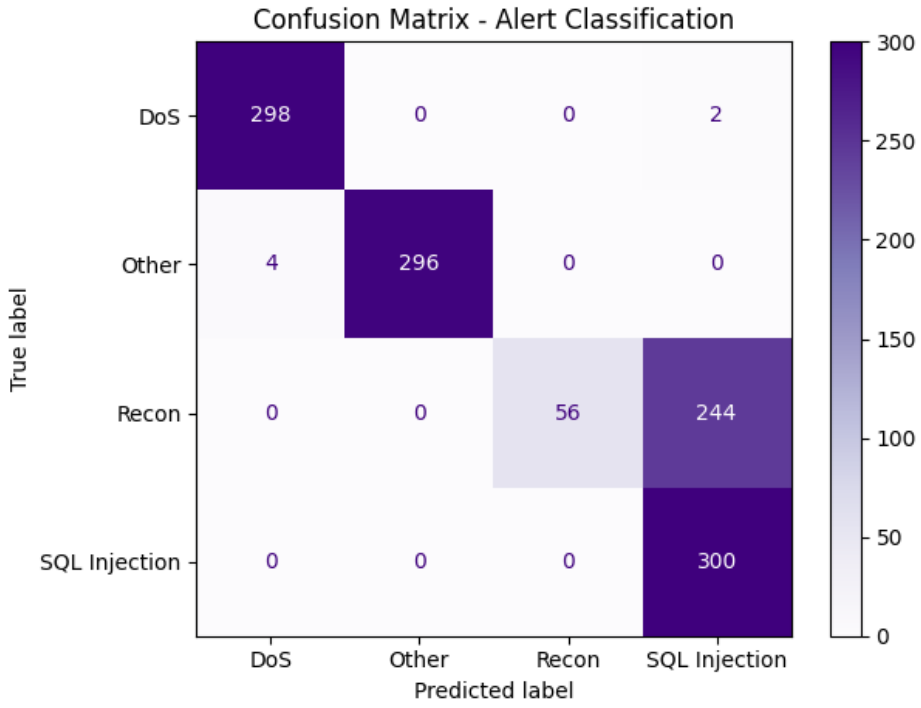



*Figure 5.6:* Confusion matrix showing ML classifier performance

The confusion matrix in Figure 5.6 presents a detailed breakdown of classification outcomes for each alert category. Diagonal cells represent correctly classified instances (true positives), while off-diagonal values correspond to misclassifications. From the confusion matrix, the classifier demonstrated strong performance in detecting DoS (298/300) and SQL Injection (300/300) alerts, as well as the Other category (296/300). However, Recon alerts posed a challenge, with only 56 out of 300 correctly classified and the majority (244) misclassified as SQL Injection. This indicates that the model struggled to distinguish Recon alerts from similar patterns in the SQL Injection class.

**Bar chart of classification metrics per attack type**

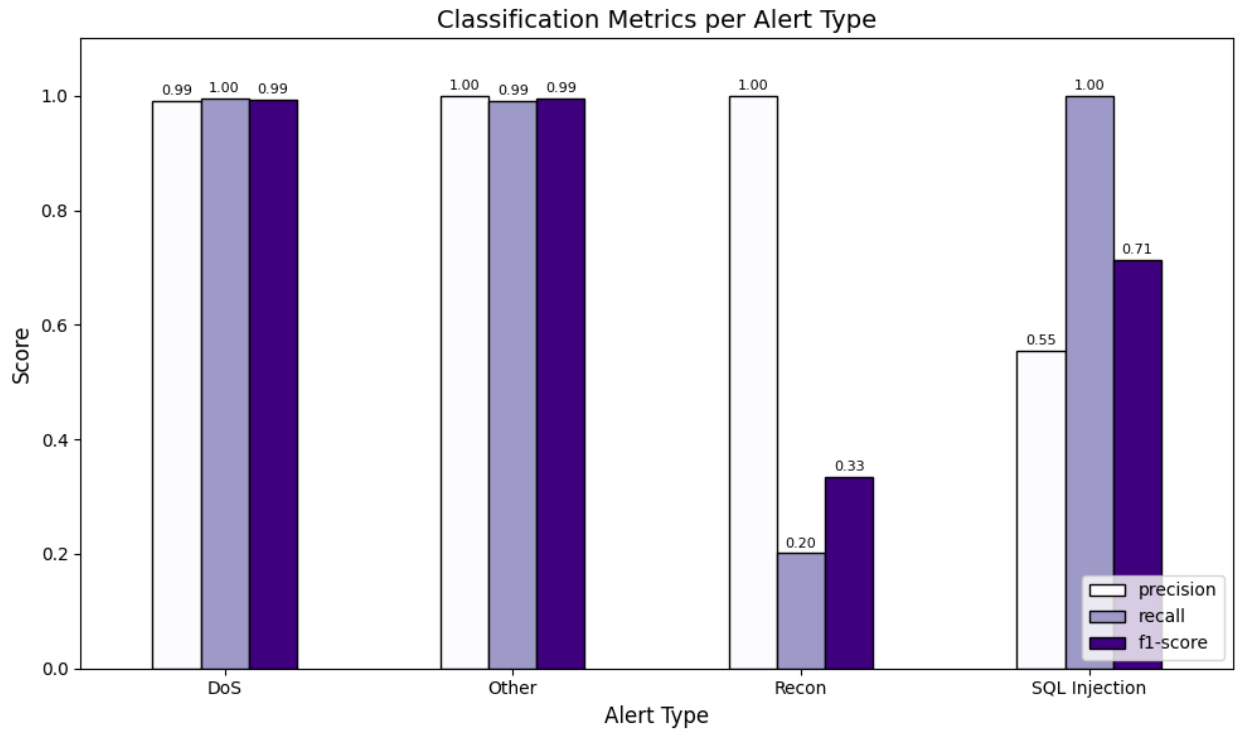



*Figure 5.7:* Classification metrics per alert type

Figure 5.7 visualizes the precision, recall, and F1-score for each attack type:

1. **DoS**

The model achieved high precision (0.99) and perfect recall (1.00), resulting in an F1-score of 0.99. This indicates highly reliable identification of DoS attacks, with minimal false positives and strong generalization performance.

2. **Other**

The classifier also performed consistently in this category, achieving 0.99 for both precision and recall, and an F1-score of 0.99. This suggests effective differentiation of benign or miscellaneous traffic not categorized under specific attacks.

### 3. Recon

Although the model achieved a perfect precision of 1.00 for the Recon class, it struggled significantly with recall, which was only 0.20. This means that while every alert the model labeled as Recon was correct with no false positives, it failed to identify most actual Recon events high number of false negatives. As a result, the F1-score was just 0.33, indicating poor overall performance for this class.

This low recall suggests the classifier has difficulty recognizing the behavior of reconnaissance (Recon) attacks. One key reason for this may be the limited and insufficiently discriminative features in the dataset. Recon attacks, such as port scans or network mapping, often exhibit subtle patterns that resemble legitimate traffic, especially in datasets that do not include detailed temporal or protocol-specific features like port scan frequency, number of unique destination ports, connection duration, or TCP flag behavior. The current feature set (likely to consist of IPs, alert types, and timestamps) may not adequately capture these behavioral nuances, making it hard for the model to learn distinctive patterns for this attack category.

This misclassification trend supports findings by Bhuyan et al. (2011), who noted that accurate detection of reconnaissance behavior often requires deep inspection of port scanning patterns, TCP flags, and timing features.

### 4. SQL Injection

The model correctly identified all SQL Injection alerts, achieving a recall of 1.00, indicating strong sensitivity. However, its precision dropped to 0.55, as many Recon alerts were misclassified as SQL Injection. This led to a moderate F1-score of 0.71. The low precision suggests the model struggled to distinguish SQL Injection from other similar behaviors, particularly Recon. This may be due to feature overlap in the dataset, where both attack types share patterns such as repeated requests or unusual query structures. The classifier likely overfitted to SQL Injection features, causing it to misclassify unrelated alerts.

## 5.6 Handling of Benign and Unrecognized Traffic

Although the testing process primarily focused on three specific and simulated attack types of Denial-of-Service (DoS), Port Scanning (Recon), and SQL Injection the Intrusion Detection System (IDS) also generated alerts for non-malicious or unintentional traffic. These included normal ICMP echo requests (pings), routine HTTP communication, and background network activity.

This outcome is consistent with real-world IDS deployments, where signature-based detection mechanisms like Snort often trigger alerts for ambiguous or legitimate behavior, especially when general rule sets are used. These false positives are a common challenge in traditional IDS systems.

In this project, such alerts were handled as follows:

- Alerts not associated with any simulated attack were labeled as “Benign” or “Other” during data preprocessing.

- These were included in the machine learning (ML) training dataset to improve the classifier's ability to differentiate real threats from harmless traffic.

By training the ML model on both malicious and benign traffic, the system's adaptability improved. This contributed to reducing false positives, thereby supporting more accurate and actionable alerting.

## 5.6 Error Analysis

During evaluation, it was observed that some alert categories, particularly Recon (Port Scan), exhibited low recall and F1-scores, as shown in the classification report and performance graphs. The most critical issue was the misclassification of Recon alerts as SQL Injection, resulting in high false negatives for Recon and reduced precision for SQL Injection. This outcome is often associated with feature overlapping or lack of discriminatory features in the dataset.

To mitigate this, the following steps were taken:

- Stratified sampling and class balancing techniques were applied to maintain fair representation of all attack categories during training, including under-sampling dominant classes such as DoS and Other to reduce bias.
- Feature limitations were identified, as the dataset primarily contained basic network attributes (e.g., source/destination IPs). These were insufficient to capture the behavioral traits of reconnaissance activity. Future work should incorporate richer features such as destination port patterns, connection frequency, and TCP flag analysis to enhance detection accuracy for Recon alerts.

## 5.7 Summary

The system testing phase validated the reliability and effectiveness of the Snort-based IDS enhanced with a machine learning classifier. Simulated attacks including DoS, SQL Injection, and Port Scanning were successfully detected by Snort and classified by the Random Forest model. The ML classifier achieved 79% accuracy, with high precision for most classes and perfect recall for SQL Injection. However, the Recon class presented challenges due to misclassification and low recall. Including benign traffic in training significantly contributed to reducing false positives and improving generalization. Performance evaluation was supported by visual tools such as the confusion matrix and bar chart, which helped identify weaknesses in the classifier and highlight areas needing improvement.

# Chapter 6: Conclusion and Future Work

## 6.1 Introduction

This chapter concludes the Final Year Project, which focused on the development of a Signature-Based Intrusion Detection System (IDS) using Snort, integrated with a machine learning (ML) module. It highlights the key accomplishments relative to the project objectives, outlines the primary contributions made, discusses limitations encountered during the development process, and proposes potential areas for future improvement and research.

## 6.2 Achievement

The project successfully achieved all its stated objectives, as outlined in the table below.

*Table 6.1:* Objective and Achievements of The System

| Objectives | Achievements |
|---|---|
| Install and configure Snort as a network Intrusion Detection System (IDS) on a Linux-based system and design custom Snort rules to detect various attack types, including DoS, port scanning, and SQL injection. | Snort was successfully installed and configured on a Linux-based system. Custom rules were written and tested to detect multiple types of network attacks. The system correctly generated alerts for simulated DoS, port scanning, and SQL injection attacks. |
| Integrate an external machine learning module with Snort to enhance detection accuracy, reduce false positives, and improve overall effectiveness in identifying advanced and unknown threats. | A Random Forest classifier was used to process Snort alerts. The model achieved 79% accuracy and high precision, with strong performance for most classes. Integration helped reduce false positives and improved classification accuracy. |

| | |
|---|---|
| Simulate network attacks to test and evaluate the performance of the IDS, ensuring its ability to accurately detect and respond to different intrusion scenarios. | The system was evaluated using simulated real-world network attacks and benign traffic. It consistently generated accurate alerts and demonstrated strong performance in identifying multiple threat types and filtering out non-malicious traffic. |

## 6.3 Contribution

This project contributes a practical and intelligent solution for improving network security. By combining Snort's rule-based detection with machine learning, the system can not only detect known threats but also improve alert classification. The inclusion of benign traffic in training significantly reduced false positives, making the system more reliable in real environments. This hybrid approach demonstrates a scalable and adaptable framework suitable for educational and enterprise environments.

## 6.4 Limitation

Several limitations were identified during the development and testing phases. Firstly, Snort does not natively support real-time integration with machine learning models, which typically require a batch-processing approach for alert classification. Secondly, the performance of the machine learning module is highly dependent on the quality and balance of the training dataset, insufficient or biased data may lead to poor generalization. Additionally, the current system lacks a graphical user interface (GUI), which may limit accessibility and usability for non-technical users. Finally, the current setup is optimized for local environments and may require further tuning to support large-scale or high-throughput networks.

## 6.5 Future Work

To further improve the accuracy, usability, and scalability of the Snort-based IDS, several areas for enhancement are recommended:

1. **Real-Time Alert Classification**

   The current system processes Snort alerts in batches through the machine learning module. Future work should explore real-time alert classification, enabling faster detection and automated responses to intrusions.

2. **Feature Enhancement for ML Classification**

   Future improvements should focus on incorporating richer, protocol-specific features into the dataset, such as destination port patterns, TCP flag analysis, request payload content, and connection timing. These features can significantly improve the classifier's ability to distinguish between similar attack types like Recon and SQL Injection.

3. **Graphical User Interface (GUI)**

   Developing a simple web-based or desktop GUI would allow users to easily monitor alerts, view classification results, and manage Snort rules enhancing usability for non-technical users.

## 6.6 Conclusion

In conclusion, this project successfully achieved its core objectives. Snort was installed and configured with custom rules to detect various forms of network intrusion. A machine learning module, based on the Random Forest algorithm, was integrated to classify alerts intelligently and reduce false positives. The system was thoroughly tested using simulated attack scenarios and benign traffic, demonstrating its effectiveness and potential for real-world deployment. Although some limitations remain, the results highlight the value of combining signature-based detection with machine learning to enhance network security. This project lays a strong foundation for future research and development in intelligent IDS frameworks, particularly those focused on real-time detection, adaptive learning, and scalable deployment.

# References


Aeraj, O. E., & Leghris, C. (2023). Intelligent Intrusion Detection System SNORT and SVM. Revue D Intelligence Artificielle, 37(6), 1629–1635. https://doi.org/10.18280/ria.370627

Al Ghafri, H. H., Abidin, Z. Z., Kurbonov, K., & Yusof, R. (2019). Implementation of Intrusion Detection System using Snort. *Journal of Advanced Computing Technology and Application (JACTA)*, 1(1), 9-16. Retrieved from https://jacta.utem.edu.my/jacta/article/view/5266

Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017, September). Evaluation of machine learning algorithms for intrusion detection system. In 2017 IEEE 15th international symposium on intelligent systems and informatics (SISY) (pp. 000277-000282). IEEE.

Anafcheh, A. (2018). *Intrusion Detection with OSSEC*. https://www.theseus.fi/handle/10024/150030

Arsalan, A., Burhan, M., Rehman, R. A., Umer, T., & Kim, B. (2023). E-DRAFT: an efficient data retrieval and forwarding technique for named data network based wireless multimedia sensor networks. *IEEE Access*, *11*, 15315–15328. https://doi.org/10.1109/access.2023.3244247

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2011). Surveying port scans and their detection methodologies. The Computer Journal, 54(10), 1565–1581. https://doi.org/10.1093/comjnl/bxr035

Chaithanya, H. V., Prerana, M. S., & Mohan, P. (2023). Towards Detection of Network Attacks by Snort Analysis Using Machine Learning Techniques. *2023 World Conference on Communication & Computing (WCONF)*, 1–8. https://doi.org/10.1109/wconf58270.2023.10235153

CyberHub. (2023). CyberHub. https://cyberhub.sa/posts/2479

Cybersecurity Ventures, C. (2024, November 18). Cybercrime to cost the world $10.5 trillion annually by 2025. Cybercrime Magazine. https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/

David, A., & Adefolaju, A. (2021). A Model for Intrusion Detection in Cybersecurity using Random Forest Algorithm. Afr. J. Comp. & ICT, 14(2), 46–51. https://afrjcict.net/wp-content/uploads/2025/01/d9e25-vol142sep21pap5pagenumb.pdf

Disha, R. A., & Waheed, S. (2022). Performance analysis of machine learning models for intrusion detection system using Gini Impurity-based Weighted Random Forest (GIWRF) feature selection technique. Cybersecurity, 5(1). https://doi.org/10.1186/s42400-021-00103-8

El Aeraj, O., & Leghris, C. (2024). Analysis of the SNORT intrusion detection system using machine learning. *International Journal of Information Science and Technology*, 8(1), 1-9.

Hoover, C. (2022). Comparative Study of Snort 3 and Suricata Intrusion Detection Systems. *Computer Science and Computer Engineering Undergraduate Honors Theses* Retrieved from https://scholarworks.uark.edu/csceuht/105

Lewandowska, N. (2024). Intrusion Detection Systems: categories, attack detection and response. *Computer Science and Mathematics Security Systems*. https://doi.org/10.20944/preprints202402.0008.v1

OSSEC. (2023, September 18). OSSEC – World's Most Widely Used Host Intrusion Detection System - HIDS. https://www.ossec.net/

Pramudya, P. B., & Alamsyah, A. (2022). Implementation of signature-based intrusion detection system using SNORT to prevent threats in network servers. *Journal of Soft Computing Exploration*, *3*(2). https://doi.org/10.52465/joscex.v3i2.80

Preethi, T., Reddy, P. R., Likhitha, L., Kumar, P. P., & Kamani, A. (2024). A Novel Approach for Anomaly Detection using Snort Integrated with Machine Learning. *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom).* https://doi.org/10.23919/indiacom61295.2024.10498401

Pureti, N. (2022). Zero-Day Exploits: Understanding the Most Dangerous Cyber Threats. *International Journal of Advanced Engineering Technologies and Innovations*, 1(2), 70-97.

Suricata. (2024, July 12) Suricata. Retrieved from https://suricata.io/

Tripathy, S. S., & Behera, B. (2024). EVALUATION OF FUTURE PERSPECTIVES ON SNORT AND WIRESHARK AS TOOLS AND TECHNIQUES FOR INTRUSION DETECTION SYSTEM. *Industrial Engineering Journal*. 53. 18-40.

Umer, M. A., Junejo, K. N., Jilani, M. T., & Mathur, A. P. (2022). Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations. *International Journal of Critical Infrastructure Protection*, 38, 100516. https://doi.org/10.1016/j.ijcip.2022.100516

Venable, J. R., Pries-Heje, J., & Baskerville, R. L. (2017). Choosing a Design Science Research Methodology. *ACIS 2017 Proceedings*. 112. https://aisel.aisnet.org/acis2017/112

Ziobro, Z. (2022, August 12). What is Zeek and Why Is It Important? NetQuest. NetQuest. https://netquestcorp.com/3-minute-crash-course-on-zeek/

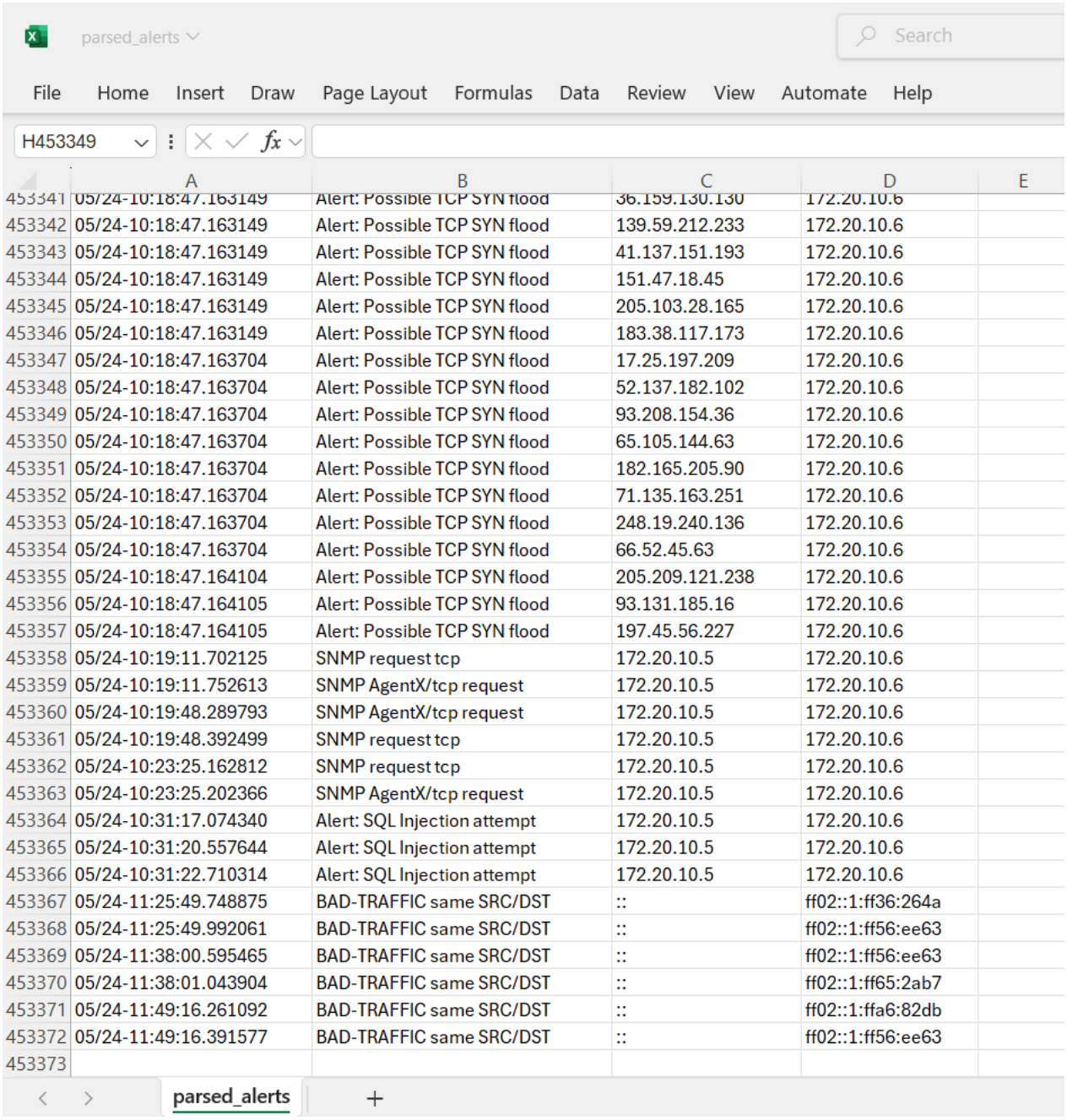
# Appendices

## Appendix A: List of Python Files

| Script Name | Function |
|---|---|
| parse_alerts.py | Parses the Snort alert log file and extracts relevant fields such as timestamp, alert message, source IP, and destination IP. Outputs structured data to parsed_alerts.csv. |
| feature_engineering.py | Converts parsed alert data into machine-learning-compatible features. Tasks include encoding alert types, converting IPs to numeric form, and balancing class distribution. Outputs structured data to alerts_preprocessed.csv. |
| train_model.py | Trains a Random Forest classifier using the preprocessed dataset. Evaluates performance using accuracy, precision, recall, and F1-score. Also generates a visual confusion matrix. |
| plot_classification_metrics.py | Generates graphs such as confusion matrix and per-class metric visualizations from the trained model's results. |

Note: Code snippets, terminal outputs, and figures related to these scripts are already included in Section 4.7 of the main report.

## Appendix B: Parsed and Preprocessed Alert Dataset



parsed_alerts

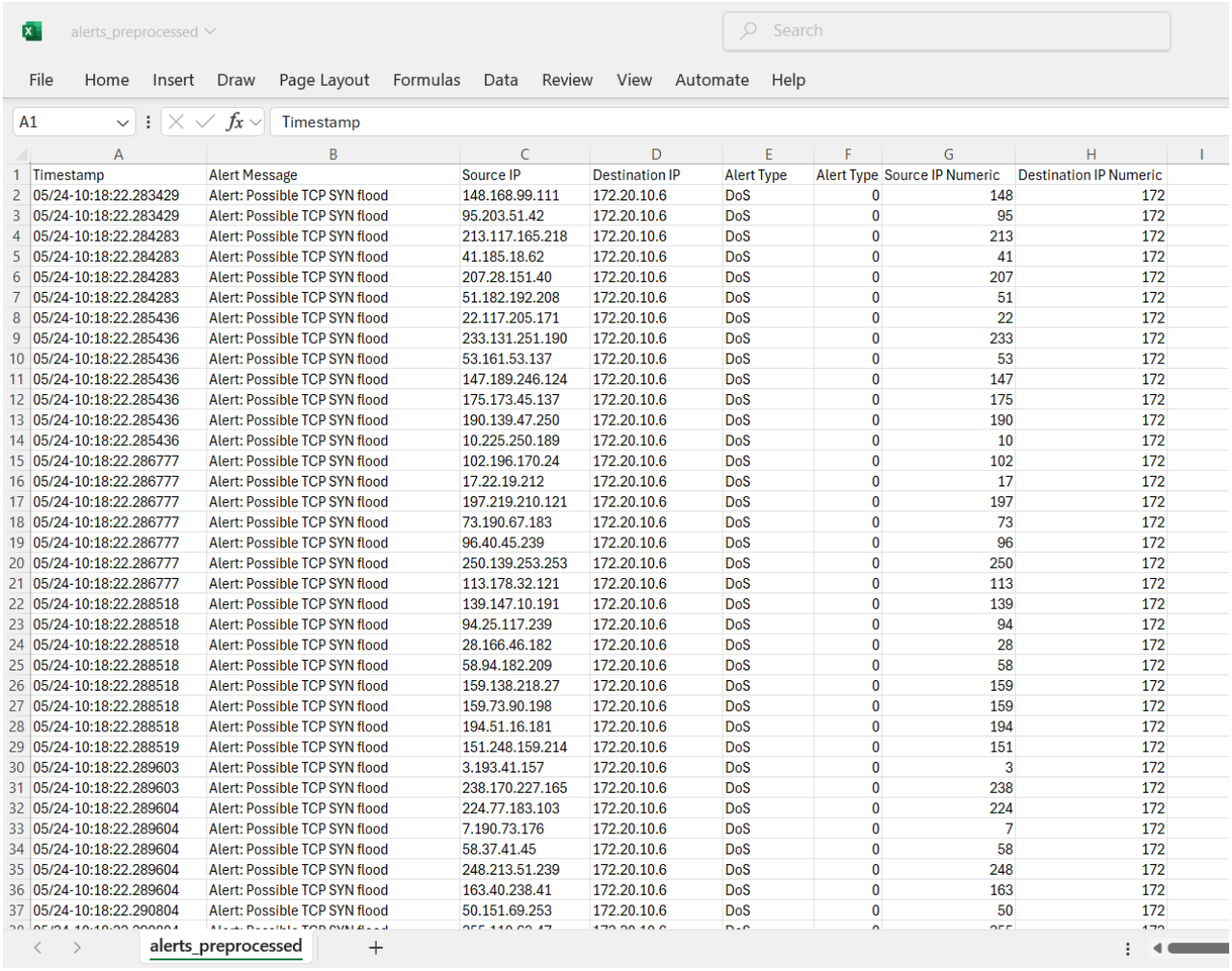
Search

File Home Insert Draw Page Layout Formulas Data Review View Automate Help

H453349

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 453341 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 36.159.130.130 | 172.20.10.6 | |
| 453342 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 139.59.212.233 | 172.20.10.6 | |
| 453343 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 41.137.151.193 | 172.20.10.6 | |
| 453344 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 151.47.18.45 | 172.20.10.6 | |
| 453345 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 205.103.28.165 | 172.20.10.6 | |
| 453346 | 05/24-10:18:47.163149 | Alert: Possible TCP SYN flood | 183.38.117.173 | 172.20.10.6 | |
| 453347 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 17.25.197.209 | 172.20.10.6 | |
| 453348 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 52.137.182.102 | 172.20.10.6 | |
| 453349 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 93.208.154.36 | 172.20.10.6 | |
| 453350 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 65.105.144.63 | 172.20.10.6 | |
| 453351 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 182.165.205.90 | 172.20.10.6 | |
| 453352 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 71.135.163.251 | 172.20.10.6 | |
| 453353 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 248.19.240.136 | 172.20.10.6 | |
| 453354 | 05/24-10:18:47.163704 | Alert: Possible TCP SYN flood | 66.52.45.63 | 172.20.10.6 | |
| 453355 | 05/24-10:18:47.164104 | Alert: Possible TCP SYN flood | 205.209.121.238 | 172.20.10.6 | |
| 453356 | 05/24-10:18:47.164105 | Alert: Possible TCP SYN flood | 93.131.185.16 | 172.20.10.6 | |
| 453357 | 05/24-10:18:47.164105 | Alert: Possible TCP SYN flood | 197.45.56.227 | 172.20.10.6 | |
| 453358 | 05/24-10:19:11.702125 | SNMP request tcp | 172.20.10.5 | 172.20.10.6 | |
| 453359 | 05/24-10:19:11.752613 | SNMP AgentX/tcp request | 172.20.10.5 | 172.20.10.6 | |
| 453360 | 05/24-10:19:48.289793 | SNMP AgentX/tcp request | 172.20.10.5 | 172.20.10.6 | |
| 453361 | 05/24-10:19:48.392499 | SNMP request tcp | 172.20.10.5 | 172.20.10.6 | |
| 453362 | 05/24-10:23:25.162812 | SNMP request tcp | 172.20.10.5 | 172.20.10.6 | |
| 453363 | 05/24-10:23:25.202366 | SNMP AgentX/tcp request | 172.20.10.5 | 172.20.10.6 | |
| 453364 | 05/24-10:31:17.074340 | Alert: SQL Injection attempt | 172.20.10.5 | 172.20.10.6 | |
| 453365 | 05/24-10:31:20.557644 | Alert: SQL Injection attempt | 172.20.10.5 | 172.20.10.6 | |
| 453366 | 05/24-10:31:22.710314 | Alert: SQL Injection attempt | 172.20.10.5 | 172.20.10.6 | |
| 453367 | 05/24-11:25:49.748875 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ff36:264a | |
| 453368 | 05/24-11:25:49.992061 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ff56:ee63 | |
| 453369 | 05/24-11:38:00.595465 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ff56:ee63 | |
| 453370 | 05/24-11:38:01.043904 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ff65:2ab7 | |
| 453371 | 05/24-11:49:16.261092 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ffa6:82db | |
| 453372 | 05/24-11:49:16.391577 | BAD-TRAFFIC same SRC/DST | :: | ff02::1:ff56:ee63 | |
| 453373 | | | | | |

parsed_alerts




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Timestamp | Alert Message | Source IP | Destination IP | Alert Type | Alert Type | Source IP Numeric | Destination IP Numeric | |
| 2 | 05/24-10:18:22.283429 | Alert: Possible TCP SYN flood | 148.168.99.111 | 172.20.10.6 | DoS | 0 | 148 | 172 | |
| 3 | 05/24-10:18:22.283429 | Alert: Possible TCP SYN flood | 95.203.51.42 | 172.20.10.6 | DoS | 0 | 95 | 172 | |
| 4 | 05/24-10:18:22.284283 | Alert: Possible TCP SYN flood | 213.117.165.218 | 172.20.10.6 | DoS | 0 | 213 | 172 | |
| 5 | 05/24-10:18:22.284283 | Alert: Possible TCP SYN flood | 41.185.18.62 | 172.20.10.6 | DoS | 0 | 41 | 172 | |
| 6 | 05/24-10:18:22.284283 | Alert: Possible TCP SYN flood | 207.28.151.40 | 172.20.10.6 | DoS | 0 | 207 | 172 | |
| 7 | 05/24-10:18:22.284283 | Alert: Possible TCP SYN flood | 51.182.192.208 | 172.20.10.6 | DoS | 0 | 51 | 172 | |
| 8 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 22.117.205.171 | 172.20.10.6 | DoS | 0 | 22 | 172 | |
| 9 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 233.131.251.190 | 172.20.10.6 | DoS | 0 | 233 | 172 | |
| 10 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 53.161.53.137 | 172.20.10.6 | DoS | 0 | 53 | 172 | |
| 11 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 147.189.246.124 | 172.20.10.6 | DoS | 0 | 147 | 172 | |
| 12 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 175.173.45.137 | 172.20.10.6 | DoS | 0 | 175 | 172 | |
| 13 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 190.139.47.250 | 172.20.10.6 | DoS | 0 | 190 | 172 | |
| 14 | 05/24-10:18:22.285436 | Alert: Possible TCP SYN flood | 10.225.250.189 | 172.20.10.6 | DoS | 0 | 10 | 172 | |
| 15 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 102.196.170.24 | 172.20.10.6 | DoS | 0 | 102 | 172 | |
| 16 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 17.22.19.212 | 172.20.10.6 | DoS | 0 | 17 | 172 | |
| 17 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 197.219.210.121 | 172.20.10.6 | DoS | 0 | 197 | 172 | |
| 18 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 73.190.67.183 | 172.20.10.6 | DoS | 0 | 73 | 172 | |
| 19 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 96.40.45.239 | 172.20.10.6 | DoS | 0 | 96 | 172 | |
| 20 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 250.139.253.253 | 172.20.10.6 | DoS | 0 | 250 | 172 | |
| 21 | 05/24-10:18:22.286777 | Alert: Possible TCP SYN flood | 113.178.32.121 | 172.20.10.6 | DoS | 0 | 113 | 172 | |
| 22 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 139.147.10.191 | 172.20.10.6 | DoS | 0 | 139 | 172 | |
| 23 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 94.25.117.239 | 172.20.10.6 | DoS | 0 | 94 | 172 | |
| 24 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 28.166.46.182 | 172.20.10.6 | DoS | 0 | 28 | 172 | |
| 25 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 58.94.182.209 | 172.20.10.6 | DoS | 0 | 58 | 172 | |
| 26 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 159.138.218.27 | 172.20.10.6 | DoS | 0 | 159 | 172 | |
| 27 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 159.73.90.198 | 172.20.10.6 | DoS | 0 | 159 | 172 | |
| 28 | 05/24-10:18:22.288518 | Alert: Possible TCP SYN flood | 194.51.16.181 | 172.20.10.6 | DoS | 0 | 194 | 172 | |
| 29 | 05/24-10:18:22.288519 | Alert: Possible TCP SYN flood | 151.248.159.214 | 172.20.10.6 | DoS | 0 | 151 | 172 | |
| 30 | 05/24-10:18:22.289603 | Alert: Possible TCP SYN flood | 3.193.41.157 | 172.20.10.6 | DoS | 0 | 3 | 172 | |
| 31 | 05/24-10:18:22.289603 | Alert: Possible TCP SYN flood | 238.170.227.165 | 172.20.10.6 | DoS | 0 | 238 | 172 | |
| 32 | 05/24-10:18:22.289604 | Alert: Possible TCP SYN flood | 224.77.183.103 | 172.20.10.6 | DoS | 0 | 224 | 172 | |
| 33 | 05/24-10:18:22.289604 | Alert: Possible TCP SYN flood | 7.190.73.176 | 172.20.10.6 | DoS | 0 | 7 | 172 | |
| 34 | 05/24-10:18:22.289604 | Alert: Possible TCP SYN flood | 58.37.41.45 | 172.20.10.6 | DoS | 0 | 58 | 172 | |
| 35 | 05/24-10:18:22.289604 | Alert: Possible TCP SYN flood | 248.213.51.239 | 172.20.10.6 | DoS | 0 | 248 | 172 | |
| 36 | 05/24-10:18:22.289604 | Alert: Possible TCP SYN flood | 163.40.238.41 | 172.20.10.6 | DoS | 0 | 163 | 172 | |
| 37 | 05/24-10:18:22.290804 | Alert: Possible TCP SYN flood | 50.151.69.253 | 172.20.10.6 | DoS | 0 | 50 | 172 | |