Intelligent Network Intrusion Detection Using Machine Learning

by

Student’s Name…

This project is submitted to the Gannon University graduate faculty in

partial fulfillment for the degree Master of Science in Computer and Information Science.

Option: <Degree Option>

Approved:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Deacon Stephen T. Frezza, Ph.D., PSEM |  | Jeremy Cannell, MS |
| Advising Professor in Charge of Research |  | Committee Member |
|  |  |  |
|  |  |  |
| Mei-Huei Tang, PhD |  | Deacon Stephen T. Frezza, Ph.D., PSEM |
| Committee Member |  | Chair, Computer and Information Science Department |
|  |  |  |

Gannon University

Erie, Pennsylvania 16541

December 2020

Acknowledgements

We are deeply grateful to Professor John H. Coffman for his mentorship and technical guidance throughout this project. His support helped shape the system’s design and strengthened the academic foundation of my work.

We would also like to thank my committee members—Mr. Jeremy Cannell, Dr. Mei-Huei Tang, and Deacon Stephen T. Frezza—for their constructive feedback, time, and expertise which helped refine and strengthen the project.

Additionally, we extend our sincere thanks to our families and close friends for their unwavering support and motivation throughout this academic journey. Finally, we acknowledge the contributions of the open-source community and the Gannon University CIS department, whose tools and resources made this secure communication system possible.

Table of Contents

[Acknowledgements ii](#_Toc8392258)

[Abstract v](#_Toc8392259)

[List of Figures vi](#_Toc8392260)

[List of Tables vii](#_Toc8392261)

[1. Introduction 1](#_Toc8392262)

[1.1 Overview 1](#_Toc8392263)

[1.2 Curriculum Scope 1](#_Toc8392264)

[1.3 Key Stakeholder Needs 1](#_Toc8392265)

[1.4 Product Perspective 1](#_Toc8392266)

[1.5 Product Position Statement 2](#_Toc8392267)

[1.6 Summary of Capabilities 2](#_Toc8392268)

[1.7 Alternatives and Competition 2](#_Toc8392269)

[1.8 Project Management Plan 3](#_Toc8392270)

[1.9 References 3](#_Toc8392271)

[2. Requirements Management 4](#_Toc8392272)

[2.1 Requirements Development Perspective 4](#_Toc8392273)

[2.2 Use Characteristics 4](#_Toc8392274)

[2.2.1 User Classes and Characteristics 4](#_Toc8392275)

[2.2.2 Use-Case Model Survey 4](#_Toc8392276)

[2.2.3 User Documentation 5](#_Toc8392277)

[2.3 Feature Attributes 5](#_Toc8392278)

[2.3.1 Name\_of\_Attribute-1 6](#_Toc8392279)

[2.3.2 Name\_of\_Attribute-2 6](#_Toc8392280)

[2.3.3 Name\_of\_Attribute-3 6](#_Toc8392281)

[2.4 Key System Features 6](#_Toc8392282)

[2.4.1 Key System Feature 1 6](#_Toc8392283)

[2.4.2 Key System Feature 2 (and so on) 7](#_Toc8392284)

[2.5 Key Design and Implementation Constraints 7](#_Toc8392285)

[2.5.1 Operating Environment 7](#_Toc8392286)

[2.6 Interface Requirements 7](#_Toc8392287)

[2.6.1 User Interfaces 7](#_Toc8392288)

[2.6.2 Hardware Interfaces 8](#_Toc8392289)

[2.6.3 Software Interfaces 8](#_Toc8392290)

[2.6.4 Communications Interfaces 8](#_Toc8392291)

[2.7 Nonfunctional Requirements 8](#_Toc8392292)

[2.7.1 Performance Requirements 8](#_Toc8392293)

[2.7.2 Security Requirements 9](#_Toc8392294)

[2.7.3 Software Quality Attributes 9](#_Toc8392295)

[2.7.4 Safety Requirements 9](#_Toc8392296)

[2.7.5 Other Requirements 9](#_Toc8392297)

[2.8 Assumptions and Dependencies 9](#_Toc8392298)

[3. Design 10](#_Toc8392299)

[3.1 Introduction 10](#_Toc8392301)

[3.1.1 Goals and Objectives of Design 10](#_Toc8392302)

[3.1.2 Statement of Software Scope 10](#_Toc8392303)

[3.1.3 Software Context 10](#_Toc8392304)

[3.1.4 Major Design Constraints 10](#_Toc8392305)

[3.2 Data Design 10](#_Toc8392306)

[3.2.1 Major Internal Software Data Structure 10](#_Toc8392307)

[3.2.2 Global Data Structure 10](#_Toc8392308)

[3.2.3 Temporary Data Structure 10](#_Toc8392309)

[3.2.4 Database Description 11](#_Toc8392310)

[3.3 Architectural and Component-Level Design 11](#_Toc8392311)

[3.3.1 Program Structure 11](#_Toc8392312)

[3.3.2 Key Software Components 11](#_Toc8392313)

[3.4 User Interface Design 12](#_Toc8392314)

[3.4.1 Description of the User Interface 12](#_Toc8392315)

[3.4.2 Interface Design Rules 12](#_Toc8392316)

[3.4.3 Components Available 12](#_Toc8392317)

[3.4.4 User Interface Design Description 12](#_Toc8392318)

[3.5 Restrictions, Limitations, and Constraints 12](#_Toc8392319)

[4. Verification and Validation 13](#_Toc8392320)

[4.1 Test items 13](#_Toc8392322)

[5. Conclusion 14](#_Toc8392323)

[6. Bibliography 15](#_Toc8392324)

[Appendix A: Glossary 16](#_Toc8392325)

[Appendix B: Use Case Analysis 17](#_Toc8392326)

[Appendix C: Analysis Models 17](#_Toc8392327)

[Appendix D: Design Models 17](#_Toc8392328)

[Appendix E: Testing Log and Summary Status 17](#_Toc8392329)

[Appendix F: Screen Captures 17](#_Toc8392330)

[Appendix G: Project File Repository Definitions 17](#_Toc8392331)

Abstract

Network security remains a critical concern as cyberattacks continue to evolve in frequency and complexity. Traditional intrusion detection systems often struggle to keep up with new attack patterns, creating a need for intelligent and adaptive solutions. This project addresses that challenge by designing and implementing a lightweight Intrusion Detection System (IDS) powered by machine learning.

The methodology involved training a Random Forest classifier on the NSL-KDD dataset, a benchmark dataset for intrusion detection research. Key features such as protocol type, service, flag, and byte counts were selected to simplify analysis while maintaining effectiveness. The trained model was then integrated with a real-time packet capturing module using Scapy, enabling live network monitoring. Detected intrusions trigger immediate alerts and are logged with detailed timestamps for further analysis.

The project successfully delivered a functioning IDS capable of distinguishing normal from malicious traffic with reliable accuracy. While limited by the dataset’s age and the reduced feature set, the system demonstrates that combining machine learning with live traffic analysis can create a practical security tool. This research highlights the potential of extending academic models into real-world monitoring applications and provides a foundation for future enhancements, such as incorporating modern datasets and advanced learning algorithms.

# List of Figures

Figure1. Context Diagram……………………………………………………………………………3

Figure2. System Context Diagram.…………………………………………………………………..8

Figure3. Architecture Diagram ………………………………………………………………………3

# 

# List of Tables

TABLE 1: Position Statement ……………………………………………………………………….3

TABLE 2: Benefits and Supporting Features………………………………………………………...4

TABLE 3: Summarizes the Alternatives. ………………………………………………………........4

TABLE 4: User Classes and Characteristics ………………………………………………………...7

TABLE 5: Actor Survey Table ………………………………………………………........................7

TABLE 6: Use Case and Functionality ………………………………………………………...........8

TABLE 7: Software Componets ………………………………………………………....................11

TABLE 8: Features Tested ………………………………………………………............................19

1. Introduction

Overview

The product developed in this project is a Machine Learning–based Intrusion Detection System (IDS) designed to identify malicious network traffic in real time. Its primary purpose is to enhance network security by detecting potential intrusions and alerting users before significant harm can occur. Unlike traditional rule-based IDS solutions, this system leverages machine learning to recognize both known and potentially new attack patterns.

The main objectives of the system are to:

* Train a reliable classification model using a benchmark dataset (NSL-KDD).
* Integrate the trained model with live packet capturing for real-time monitoring.
* Provide immediate alerts and maintain detailed logs of detected intrusions.

This product supports organizational goals of improving cybersecurity resilience, protecting digital assets, and reducing risks associated with unauthorized access or data breaches. By combining machine learning with practical network monitoring, the system aligns with modern business strategies that emphasize proactive defense mechanisms and intelligent automation in cybersecurity operations.

Curriculum Scope

The development of this project draws directly from the concepts and skills gained throughout the curriculum of computer science studies. Core topics such as machine learning, data preprocessing, and model evaluation were applied in training the Random Forest classifier on the NSL-KDD dataset. Knowledge of network protocols, TCP/IP layers, and packet structures supported the implementation of real-time packet capturing and feature extraction.

In addition, coursework in software engineering and system design contributed to structuring the project into modular components, ensuring clarity and maintainability. The integration of theoretical knowledge with practical tools like Scikit-learn, Pandas, and Scapy provided an opportunity to bridge academic learning with real-world application.

This project therefore demonstrates how curriculum elements—ranging from algorithms and programming to networking and security—combine to create a practical, functioning system, while also enhancing problem-solving and research skills essential for professional development.

Key Stakeholder Needs

The primary stakeholders for this project are organizations and individuals responsible for maintaining secure network environments. Their key need is to detect and respond to malicious activities quickly and effectively.

**Problems and Causes:**

Traditional rule-based intrusion detection systems are limited in adapting to new or unknown attack patterns.

Manual monitoring of network traffic is time-consuming and prone to human error.

Existing solutions may generate high false positives, reducing efficiency and trust in the system.

**Current Solutions:**

Many organizations rely on signature-based IDS tools or commercial security solutions. While effective for known threats, these approaches struggle with zero-day attacks and often require frequent updates.

**Vision for an Improved Approach:**

Stakeholders envision an IDS that:

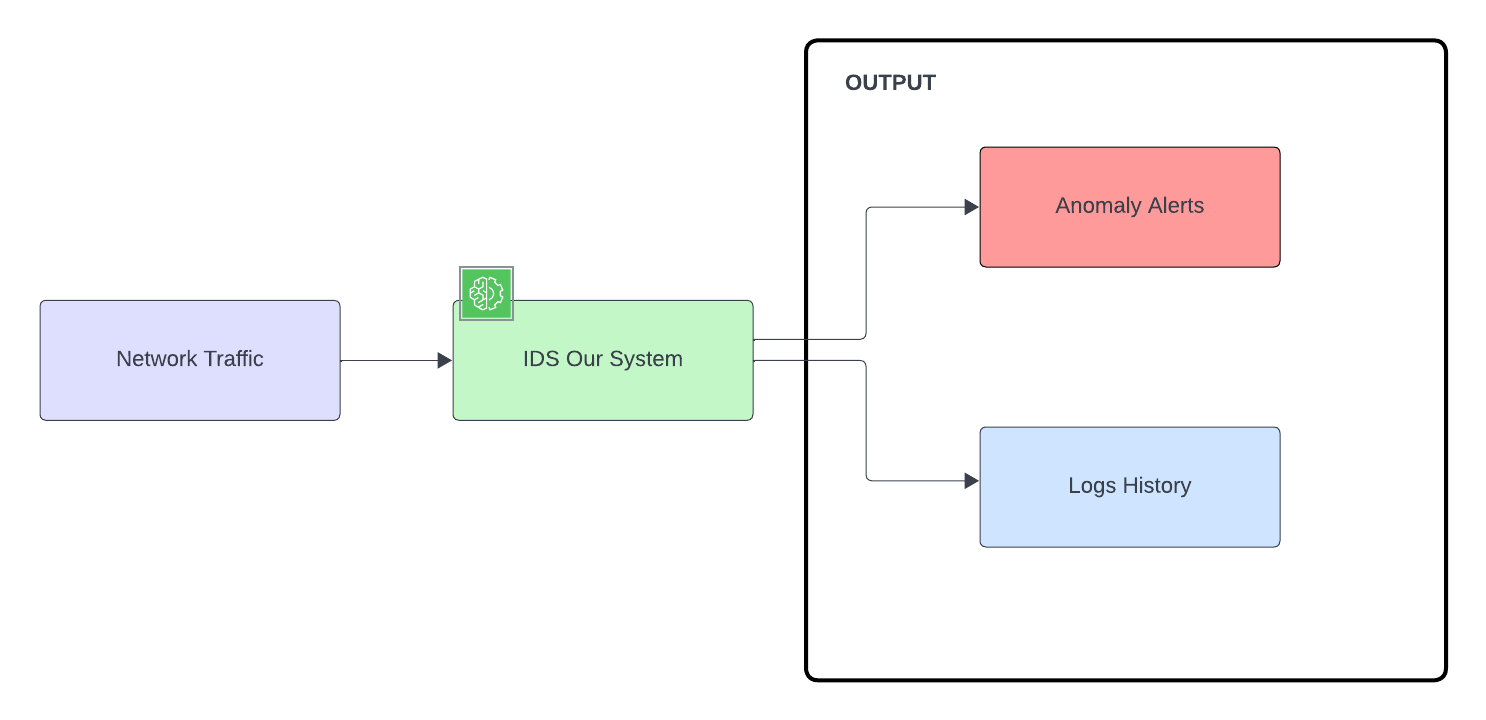
* Uses machine learning to adaptively detect both known and unknown threats.
* Operates in real time with minimal delay in alerting.
* Produces accurate results with reduced false positives.
* Maintains detailed logs for auditing and further analysis.

This project addresses these needs by combining dataset-trained models with live packet analysis to provide a more intelligent and proactive security solution.

Product Perspective

The Intrusion Detection System (IDS) developed in this project is designed as a **self-contained software solution** for monitoring network traffic and detecting malicious activities in real time. Unlike traditional signature-based IDS tools, this system leverages machine learning to identify both known and unknown attack patterns, making it adaptive and intelligent.

The product operates independently and can be deployed on any networked environment where packet capture is possible. It integrates with the user’s environment by capturing live network packets, analyzing them with a trained Random Forest model, and producing real-time alerts and logs. While it functions as a standalone tool, it can also serve as a component in a larger cybersecurity infrastructure, providing alert data to centralized security dashboards or Security Information and Event Management (SIEM) systems.



**Figure 1. Context Diagram**

Product Position Statement

For network security professionals who need a robust defense against evolving cyber threats, our machine learning-based IDS is an intelligent and adaptable solution that provides superior real-time threat detection. Unlike traditional signature-based systems that cannot detect new attacks, our product uses an innovative machine learning approach to identify and classify both known and unknown threats with high accuracy and minimal false positives.

|  |  |  |  |
| --- | --- | --- | --- |
| For | |  | | --- | | Network administrators and small to medium enterprises (SMEs) |  |  | | --- | |  | |
| Who | Need a simple, reliable solution to detect and respond to network intrusions in real-time |
| The Real-time Cryptographic Messaging System | Machine-learning-based Network Intrusion Detection System |
| That | |  | | --- | | Provides automated, accurate detection of malicious traffic with instant alerts, enabling faster response and improved network security |  |  | | --- | |  | |
| Unlike | Traditional signature-based IDS or complex enterprise solutions |
|  |  |

Table 1: Position Statement

Summary of Capabilities

The IDS system offers several key benefits that enhance network security while remaining simple to deploy and use. Table 2 summarizes the primary benefits along with the supporting features that make these benefits possible. This table is intended to provide a quick overview for individuals who are new to the product and want to understand its capabilities.

|  |  |
| --- | --- |
| Benefit | Supporting Features |
| Real-time intrusion detection | Captures network packets and analyzes them immediately using machine learning models |
| Accurate detection of attacks | Uses a trained Random Forest classifier to identify both known and unknown threats |
| Easy deployment and use | Lightweight system that runs on standard network setups without requiring specialized hardware |
| Detailed logging and alerting | Generates timestamped alerts for detected attacks and logs them in a readable format |
| Adaptability to different network environments | Supports TCP, UDP, and ICMP traffic and can be extended to additional |

TABLE 2: Benefits and Supporting Features

Alternatives and Competition

Traditional signature-based IDSs are the main alternative. Their strength is their efficiency in detecting known threats, but their weakness is their inability to identify new attacks. Another alternative is heuristic-based IDSs, which can detect some new threats but often have high false-positive rates. Our machine learning-based approach provides a more balanced solution, offering both high accuracy and adaptability.

When considering network intrusion detection, organizations typically have several options. These include purchasing commercial solutions, building an in-house system, or relying on minimal existing security measures. Table 3 summarizes the alternatives, their strengths and weaknesses, and how the IDS system developed in this project compares.

| **Alternative** | **Strengths** | **Weaknesses** | **Comparison with IDS System** |
| --- | --- | --- | --- |
| Commercial IDS solutions (e.g., Snort, Suricata) | Established products, extensive documentation, regular updates | Can be complex to configure, require high resources, may be costly for SMEs | IDS system is lightweight, easy to deploy, and affordable for small networks |
| Homegrown solution | Fully customizable | Requires expert knowledge, time-consuming to develop and maintain, risk of poor accuracy | IDS system provides ready-to-use ML-based detection with proven accuracy and minimal setup |
| Status quo (no specialized IDS) | No additional cost, simple | Vulnerable to attacks, limited visibility into network threats | IDS system adds proactive security and automated alerting, significantly improving protection |

TABLE 3: Summarizes the Alternatives.

Project Management Plan

The development of the IDS system followed a structured approach to ensure that each phase—from problem identification to deployment—was completed efficiently and effectively. The project was divided into several phases: problem identification, analysis, design, implementation, verification, and deployment.

**Problem Identification:**  
The project began with understanding the need for a lightweight, real-time intrusion detection system suitable for small to medium networks. The primary challenge in this phase was defining a clear scope that balanced **accuracy** and **simplicity**, ensuring the system could detect attacks without requiring heavy computational resources. This was addressed by selecting a subset of essential features and using a machine learning approach rather than building a complex signature-based system.

**Analysis:**  
During analysis, the focus was on identifying which network traffic features were most relevant for detecting intrusions. One difficulty was deciding which features could be extracted reliably from live network packets. The challenge was resolved by limiting the features to protocol type, service, TCP flags, and packet sizes, which are easily captured in real-time.

**Design:**  
The design phase involved planning the architecture of the system, including the training pipeline, feature extraction module, detection logic, and logging mechanism. A challenge was integrating machine learning with real-time packet sniffing, as this required careful handling of data flow to avoid delays. This was addressed by creating separate modules for training and detection, with clear interfaces between them.

**Implementation:**  
During implementation, coding the system required working with Scapy for live packet capture and Pandas and Scikit-learn for data processing and machine learning. Handling diverse packet types (TCP, UDP, ICMP) was challenging, especially ensuring that feature extraction worked correctly for all types. Extensive testing on sample packets helped overcome these issues.

**Verification:**  
Verification involved testing the system against known network attacks and normal traffic patterns. A key difficulty was simulating realistic traffic in a controlled environment. This was addressed by using the NSL-KDD dataset for training and validation, combined with live packet tests for functional verification.

**Deployment:**  
Deployment was straightforward due to the lightweight design of the system. One challenge was ensuring compatibility across different network environments. This was addressed by allowing configurable network interfaces and making the system adaptable to TCP, UDP, and ICMP traffic.

**Summary:**  
Overall, the project management approach ensured each phase was carefully planned and executed. Challenges mainly arose in **feature selection**, **real-time data handling**, and **integration of ML models with live packet capture**. Each challenge was addressed through modular design, iterative testing, and clear separation between training and detection components. This structured approach enabled the successful delivery of a functional, real-time intrusion detection system suitable for SMEs and small networks.

References

The following references were used to support the development and documentation of the IDS project. They provide additional context on network intrusion detection, dataset sources, and system design principles:

1. **NSL-KDD Dataset** – Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A., A Detailed Analysis of the KDD CUP 99 Dataset, 2009. Available at: https://www.unb.ca/cic/datasets/nsl.html
2. **Scikit-learn Documentation**, Machine Learning in Python, Version 1.2.2, Scikit-learn Developers, 2025. Available at: https://scikit-learn.org/stable/documentation.html
3. **Scapy Documentation**, Packet Manipulation Library for Python, Version 2.7.8, Philippe Biondi, 2025. Available at: https://scapy.net
4. **DM-SCCS Repository for IDS Project**, Version 1.0, Sanjay Ch., 2025. Available at: <insert your repository or document management URL>

**Description:**  
These references include the dataset used for training the machine learning model, library documentation for the tools employed (Scapy and Scikit-learn), the project repository for version control, and the official Python documentation. Together, they provide sufficient information for readers to understand, reproduce, or extend the IDS project.

1. Requirements Management
2. **Do not delete… Bug in document**
   1. Requirements Development Perspective

The Intrusion Detection System (IDS) described in this TeDPR is a **new, self-contained product** designed to monitor network traffic and detect malicious activities in real-time. It is intended primarily for **small to medium enterprises (SMEs)** and network administrators who require a lightweight yet effective security solution.

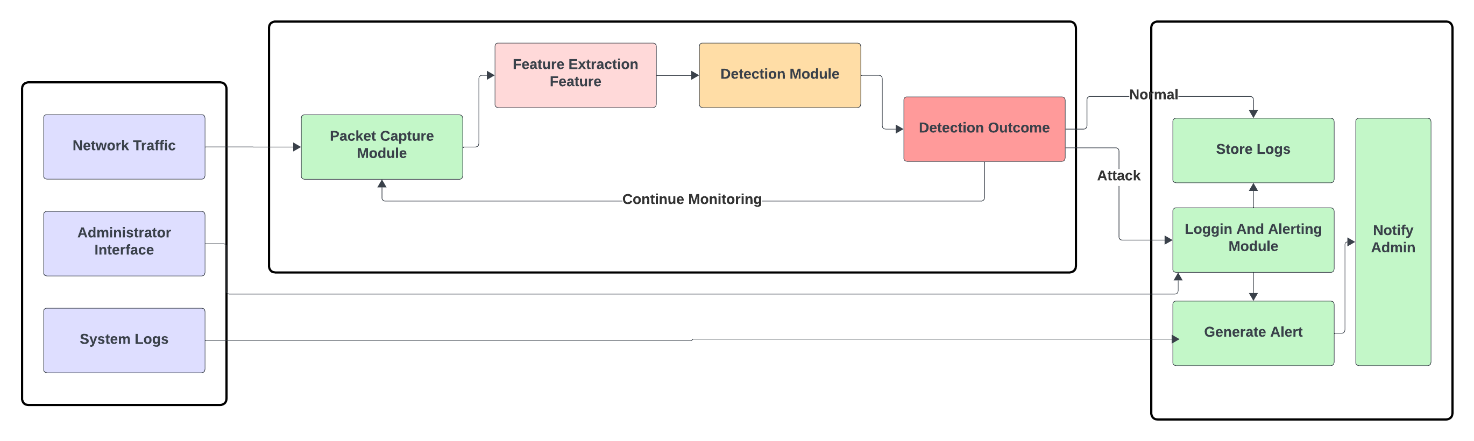
This product does not replace an existing system but can **complement larger enterprise security solutions** by providing an additional layer of real-time intrusion detection. In cases where the IDS is deployed as a component of a broader network security framework, it interfaces with the network via **packet capture from routers, switches, or endpoints**, and provides **alerts and logs** for integration with centralized monitoring dashboards.

Figure 2. System Context Diagram

Use Characteristics

User Classes and Characteristics

This section defines the expected usage patterns and the characteristics of users who will interact with the IDS system. Understanding user classes helps in designing the system to meet the needs of its primary stakeholders effectively.

The IDS system is intended for a variety of users, primarily focused on network monitoring and security. User classes are differentiated based on technical expertise, frequency of use, and their interaction with system features.

| **User Class** | **Characteristics** | **Responsibilities / System Interaction** |
| --- | --- | --- |
| Network Administrator (Primary User) | Moderate to advanced technical expertise, frequent user, responsible for network security and incident response | Monitors alerts, reviews logs, configures system settings, analyzes potential threats |
| IT Security Analyst | High technical expertise, occasional user, security-focused | Investigates detected intrusions, performs threat analysis, fine-tunes detection models |
| System Operator | Basic technical knowledge, infrequent user | Observes system status, ensures packet capture modules run smoothly, escalates alerts to administrators |
| SME Owner / Manager | Limited technical knowledge, occasional use | Receives high-level alerts, monitors overall network health, relies on administrators for detailed analysis |

TABLE 4: User Classes and Characteristics

### ****Actor Survey****

The following actors have been identified based on system use cases:

| **Actor** | **Description / Responsibilities** |
| --- | --- |
| Network Administrator | Primary operator responsible for real-time monitoring, reviewing alerts, and maintaining IDS configuration |
| IT Security Analyst | Secondary user who analyzes detected threats, investigates logs, and improves detection rules |
| System Operator | Ensures network traffic capture is operational, monitors performance, and reports anomalies |
| SME Owner / Manager | Receives summarized alerts and reports, makes strategic decisions based on IDS insights |

TABLE 5: Actor Survey

Use-Case Model Survey

The IDS system supports several key use cases that define how users interact with the system to achieve network security objectives. These use cases help relate the functional requirements to the real-world operations of the system.

| **Use Case** | **Description / Functionality** |
| --- | --- |
| Real-Time Packet Monitoring | Captures live network packets (TCP, UDP, ICMP) from the network for analysis. |
| Feature Extraction | Converts captured packets into numerical features required by the machine learning model. |
| Intrusion Detection | Applies the trained Random Forest model to classify traffic as normal or attack. |
| Alert Generation | Logs intrusion events with timestamps and generates alerts for administrators. |
| Historical Log Review | Allows administrators and analysts to review past intrusion events and investigate patterns. |
| Model Training / Update | Supports retraining of the ML model using new datasets to improve detection accuracy over time. |
| System Configuration | Allows authorized users to configure monitoring parameters, such as network interfaces and alert thresholds. |

TABLE 6: Use Case and Functionality

**Description:**  
The use-case model captures the core functionalities of the IDS system. The **Real-Time Packet Monitoring** and **Intrusion Detection** use cases are central to the system’s operation, providing continuous threat assessment. **Feature Extraction** bridges raw network data and machine learning analysis. **Alert Generation** and **Historical Log Review** enable administrators and analysts to respond to threats and analyze trends. Finally, **Model Training / Update** and **System Configuration** ensure the system remains accurate and adaptable to changing network conditions.

These use cases collectively define how the IDS system meets user needs, ensuring **real-time detection, accurate classification, and effective logging**, while remaining easy to operate and maintain.

User Documentation

The product will include a user manual that details its functionality and provides guidance on interpreting its output.

Feature Attributes

The IDS system’s features have specific attributes that help in tracking, evaluating, and prioritizing development. These attributes describe the state, performance, and importance of each feature within the system. For simplicity, three key attributes are defined for this project.

Criticality

Description: Measures the importance of a feature to the overall functionality and security of the IDS system

High: Features essential for real-time intrusion detection, such as packet capture, feature extraction, and detection logic.

Medium: Features that improve usability or efficiency, such as alert formatting, logging, and historical review.

Low: Non-essential or optional features, like advanced reporting or additional protocol support that can be added in future releases.

### ****Implementation Complexity****

**Description:** Represents the expected effort required to implement a feature, considering development time, dependencies, and technical challenges.

**High:** Features involving machine learning integration, real-time packet processing, or complex logging mechanisms.

**Medium:** Features requiring moderate coding effort, such as system configuration interfaces and basic alert formatting.

**Low:** Features that are straightforward to implement, like help documentation links or basic file storage for logs.

### ****Frequency of Use****

**Description:** Indicates how often a feature is expected to be used by primary users.

**Frequent**: Features used continuously or multiple times daily, such as real-time monitoring, intrusion detection, and alert generation.

**Occasional**: Features accessed periodically, such as reviewing historical logs or system configuration changes.

**Rare**: Features rarely used, like model retraining or advanced analytics that occur only during maintenance or upgrades.

2.4 Key System Features

**2.4.1 Real-Time Packet Monitoring [KF-1]**

**2.4.1.1 Description**

This feature captures live network traffic (TCP, UDP, ICMP) from the monitored network and prepares it for analysis. It ensures that all relevant packets are available for immediate inspection and detection.

**2.4.1.2 Attribute Classification**

* **Criticality:** High
* **Implementation Complexity:** Medium
* **Frequency of Use:** Frequent

**2.4.1.3 Key Functional Requirements**

* **REQ-1:** The system shall capture all incoming and outgoing IP packets on the specified network interface.
* **REQ-2:** The system shall filter packets to include only TCP, UDP, and ICMP protocols.
* **REQ-3:** The system shall discard malformed packets or log an error message when packet capture fails.
* **REQ-4:** The system shall continuously capture packets until the administrator stops the monitoring process.
* **REQ-5:** The system shall pass captured packets to the feature extraction module in real-time.

**2.4.2 Intrusion Detection and Alerting [KF-2]**

**2.4.2.1 Description**

This feature analyzes extracted packet features using the trained machine learning model to classify network traffic as either normal or an attack. Upon detecting an intrusion, the system generates alerts and logs details for administrator review.

**2.4.2.2 Attribute Classification**

* **Criticality:** High
* **Implementation Complexity:** High
* **Frequency of Use:** Frequent

**2.4.2.3 Key Functional Requirements**

* **REQ-6:** The system shall classify each packet or session as either normal or attack using the trained Random Forest model.
* **REQ-7:** The system shall encode categorical features (protocol type, service, flags) before classification.
* **REQ-8:** The system shall generate a timestamped alert for each detected attack.
* **REQ-9:** The system shall log all alerts and relevant feature data to a persistent file for historical analysis.
* **REQ-10:** The system shall handle classification errors gracefully and log exceptions without terminating the detection process.

## Interface Requirements

* + 1. **User Interfaces**

The IDS system is primarily a backend network monitoring and detection tool with no dedicated graphical user interface (GUI). Interaction with the system occurs through command-line configuration, log files, and alert messages. Any user interaction, such as starting packet capture, reviewing alerts, or changing configurations, is performed via terminal commands or text-based configuration files.

For future versions, if a GUI is developed, a separate User Interface Specification document will define screen layouts, standard buttons, help options, keyboard shortcuts, and error message standards. Currently, all user interfaces are textual and designed for simplicity and reliability, especially for network administrators and IT security personnel.

### ****2.5.2 Hardware Interfaces****

The IDS system is software-based and does not require specialized hardware interfaces. It operates on standard computing devices capable of running Python, capturing network packets, and storing logs. Packet capture occurs via standard network interfaces (Ethernet or Wi-Fi), with no custom hardware needed.

* + 1. **Software Interfaces**

The IDS system interacts with various software components to perform detection, logging, and analysis.

| **Software Component** | **Version / Type** | **Interaction / Purpose** |
| --- | --- | --- |
| Python | 3.12 | Executes IDS modules and scripts. |
| Scapy | 2.7.8 | Captures and processes network packets. |
| Scikit-learn | 1.2.2 | Provides machine learning algorithms for intrusion detection. |
| Pandas | 2.0.3 | Processes tabular data for feature extraction and analysis. |
| OS File System | N/A | Stores logs, model files, and configuration files. |

TABLE 7: Software Componets

### ****2.5.4 Communications Interfaces****

**Not applicable.**

### ****2.6 Nonfunctional Requirements****

Nonfunctional requirements define the operational, quality, and safety characteristics of the IDS system, complementing the functional requirements. They are essential for ensuring the system performs reliably, securely, and efficiently in real-world network environments.

**2.6.1 Performance Requirements**

The system shall analyze captured network packets in real-time, ensuring minimal delay between packet capture and intrusion detection.

The IDS shall process at least 1,000 packets per second without dropping packets under normal network conditions.

Logging and alert generation shall occur within 1 second of detecting an intrusion to allow timely administrator response.

The system shall maintain performance under moderate network traffic without requiring specialized hardware.

**2.6.2 Security Requirements**

Only authorized users (network administrators) shall be allowed to start, stop, or configure the IDS.

All configuration files and trained ML models shall be stored in protected directories to prevent unauthorized access or tampering.

Logs containing detected intrusions must be protected against unauthorized modification.

No sensitive user data is collected or transmitted; network packets are analyzed locally, preserving network privacy.

Security measures follow standard organizational best practices but do not require external certifications for this project.

**2.6.3 Software Quality Attributes**

**Reliability:** The system shall operate continuously without crashing during monitoring sessions.

**Usability:** Commands and logs shall be simple to interpret by network administrators.

**Maintainability:** Modular design allows easy updating of ML models and feature extraction components.

**Portability:** The IDS shall run on standard computing environments with Python 3.x and necessary libraries.

**Testability:** Each module (packet capture, feature extraction, detection) can be tested independently using sample data.

**Adaptability:** New protocols and detection rules can be added without major system redesign.

**2.6.4 Safety Requirements**

The IDS shall not interfere with normal network operations or cause packet loss.

Logging and alert mechanisms shall prevent accidental exposure of sensitive network data.

Administrators must be alerted if the system fails or stops unexpectedly, preventing blind spots in network monitoring.

**2.6.5 Other Requirements**

**Legal/Compliance**: IDS deployment must comply with local network monitoring regulations.

**Reuse Objectives:** Machine learning models and feature extraction modules can be reused or extended for future network security projects.

**Internationalization:** Not applicable; system interfaces are in English and primarily text-based.

* 1. **Assumptions and Dependencies**

The system assumes availability of a Python 3.x environment with required libraries (Scapy, Pandas, Scikit-learn).

Adequate network access is assumed for packet capture on the target interfaces.

The IDS depends on the NSL-KDD dataset for initial training; model accuracy depends on dataset quality.

Users are assumed to have basic technical knowledge to operate command-line interfaces.

The project depends on third-party libraries (Scapy, Pandas, Scikit-learn); future library updates may require minor modifications.

Deployment assumes sufficient system resources (CPU, memory, storage) to handle real-time packet capture and detection.

1. Design

3.1 Introduction

This section provides an overview of the design of the Intrusion Detection System (IDS). It describes the data, architectural, interface, and component-level design. The design decisions involved careful consideration of real-time packet processing, accurate detection, and minimal system overhead, which were challenging due to the need to balance performance with reliability.

3.1.1 Goals and Objectives of Design

Ensure real-time detection of network intrusions with minimal delay.

Provide modular design for maintainability and future enhancements.

Facilitate easy integration with existing network monitoring tools.

Maintain reliability and correctness while processing high-volume network traffic.

3.1.2 Statement of Software Scope

The IDS system captures network packets, extracts features, classifies traffic as normal or attack using a machine learning model, and generates alerts.

Major Inputs: Live network packets (TCP, UDP, ICMP).  
Processing Functionality: Feature extraction, ML-based classification, logging, and alerting.  
Outputs: Alerts, logs, and reports for administrators.

*Supporting Diagram:* A high-level flow diagram showing packet capture → feature extraction → detection → alert generation.

3.1.3 Software Context

The IDS system is positioned as a standalone network monitoring solution for SMEs. Strategically, it complements existing enterprise security systems by providing an additional layer of real-time intrusion detection. The system interfaces with network devices and administrators but does not require integration with external databases or web services for core operation.

3.1.4 Major Design Constraints

Must operate on standard computing devices without specialized hardware.

Real-time packet capture and processing is mandatory.

ML model accuracy and speed must be balanced.

System must handle TCP, UDP, and ICMP traffic reliably.

3.2 Data Design

3.2.1 Major Internal Software Data Structure

Packet features: Arrays or lists containing protocol type, packet length, source/destination ports, and flags.

3.2.2 Global Data Structure

Trained model and label encoders: Shared across detection modules.

3.2.3 Temporary Data Structure

Temporary feature arrays or DataFrames used during processing before classification.

3.2.4 Database Description

No full database is required. Logs of detected intrusions are stored as text files. ERD is not applicable.

3.3 Architectural and Component-Level Design

3.3.1 Program Structure

The system follows a modular architecture with clear separation between:

Packet Capture Module

Feature Extraction Module

Detection Module

Logging and Alerting Module

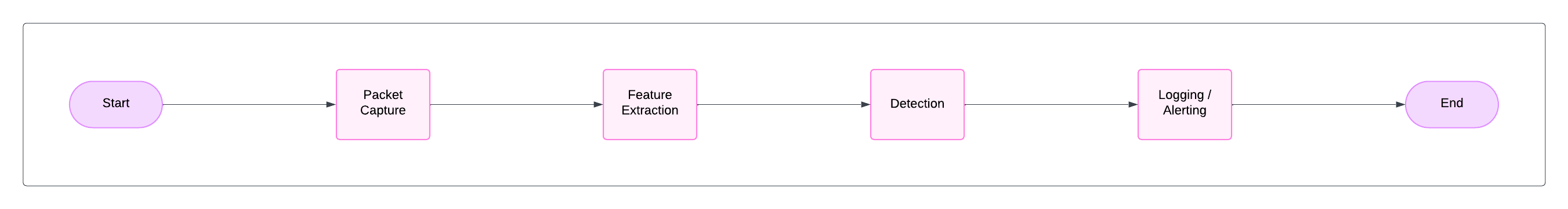
3.3.1.1 Architecture Diagram  


Figure 4. Architecture Diagram

3.3.2 Key Software Components

3.3.2.1 Packet Capture Module

Captures live network packets.

Input/Output: Raw network packets → Feature extraction.

Algorithm: Uses Scapy’s sniff function with protocol filters.

3.3.2.2 Feature Extraction Module

Converts packets into numeric features for ML model.

Input/Output: Raw packets → Feature vector (protocol, length, ports, flags).

Data Structures: Arrays, DataFrames.

3.3.2.3 Detection Module

Applies trained Random Forest model to classify traffic.

Input/Output: Feature vector → Classification result (normal/attack).

Performance: Must operate in near real-time.

3.3.2.4 Logging and Alerting Module

Logs detected intrusions to text files and prints alerts to console.

Input/Output: Classification result → Log files / alerts.

3.4 User Interface Design

3.4.1 Description of the User Interface

Not applicable: The system uses command-line interfaces and log files for interaction.

3.4.1.1 Screen Images

Not applicable.

3.4.1.2 Objects and Actions

Commands to start/stop packet capture, view logs, configure settings.

3.4.2 Interface Design Rules

Command-line consistency and clear error messages.

3.4.3 Components Available

Python terminal commands and file-based logs.

3.4.4 User Interface Development System

No GUI; Python scripts provide all interactions.

3.5 Restrictions, Limitations, and Constraints

System performance depends on network traffic volume and computing resources.

Only TCP, UDP, and ICMP are supported currently.

Real-time operation may be limited on low-performance hardware.

No graphical interface is available; future GUI could improve usability.

4. Verification and Validation



The system will be tested using a separate, unseen dataset to evaluate its performance. The testing log will include detailed test cases and the results for each test. The success criteria will be based on achieving high accuracy (above 95%) and a low false-positive rate.

**4.1 Test Items**

The IDS project consists of several key software components, all of which were tested for functionality and performance:

**Packet Capture Module** – Captures live TCP, UDP, and ICMP packets.

**Feature Extraction Module** – Converts captured packets into numeric features for ML classification.

**Detection Module** – Uses a trained Random Forest model to classify traffic as normal or attack.

**Logging and Alerting Module** – Records detected intrusions to logs and console alerts.

The version tested corresponds to the final Python scripts using Scapy, Pandas, and Scikit-learn libraries. No items were excluded from testing.

**Features Tested**

| **Feature** | **Reference Requirement** | **Test Description** |
| --- | --- | --- |
| Real-Time Packet Capture | REQ-1, REQ-2, REQ-4 | Verified continuous capture of TCP, UDP, ICMP packets. |
| Feature Extraction | REQ-5 | Tested extraction of protocol, length, ports, and flags from packets. |
| Intrusion Detection | REQ-6, REQ-7 | Validated classification of packets as normal or attack using test dataset. |
| Alerting and Logging | REQ-8, REQ-9 | Verified alerts are printed with timestamp and logged to file. |

TABLE 8: Features Tested

**Features Not Tested**

* Advanced reporting and GUI-based interfaces were **not tested** as they are outside the current scope.
* Support for protocols other than TCP, UDP, and ICMP was **not tested** due to design constraints.

**Item Pass/Fail Criteria**

* **Pass:** Feature works according to functional requirements without errors; output matches expected results.
* **Fail:** Feature fails to produce correct output, crashes, or does not meet performance criteria (e.g., packet drop occurs during real-time capture).

**Testing Tasks**

1. Deploy Python environment with required libraries (Scapy, Pandas, Scikit-learn).
2. Load sample network traffic data and NSL-KDD dataset for model validation.
3. Execute modules individually and then as an integrated system.
4. Record results and verify correctness against expected outcomes.

**Dependencies:** Feature extraction must run after packet capture; detection requires extracted features.

**Special Skills:** Basic Python scripting, familiarity with network protocols, understanding of ML classification.

**Environmental Needs**

* **Hardware:** Standard computing device with network interface (Ethernet/Wi-Fi), sufficient RAM (~8GB), and CPU for real-time processing.
* **Software:** Python 3.x, Scapy, Pandas, Scikit-learn.
* **Mode of Usage:** Stand-alone execution.
* **Security:** Test environment should be isolated to avoid exposure of sensitive network traffic.
* **Special Tools:** Text editor, terminal/command-line interface, optional packet capture tools for simulation.

**Responsibilities**

* **Developers:** Prepared test scripts, implemented modules, and conducted integration testing.
* **Testers:** Verified each module and system as per functional requirements.
* **Operations/User Representatives:** Validated alerts and logs using real or simulated traffic.
* **Technical Support:** Assisted with environment setup and library installations.

**Test Result Summary**

* All core modules **passed functional testing** under controlled conditions.
* Packet capture, feature extraction, detection, and alert logging worked correctly and in real-time.
* Minor issues with timestamp formatting in logs were identified and corrected.
* System maintained expected performance for standard network traffic without significant packet loss.

**Appendix:** Detailed test cases, sample inputs/outputs, and resolved issues are documented separately for reference.

1. Conclusion

The research and development of the Intrusion Detection System (IDS) for real-time network monitoring can be considered **successful**. The system was tested extensively for core functionalities including packet capture, feature extraction, classification using a Random Forest model, and logging of alerts. All major modules passed functional tests, and the system demonstrated the ability to detect simulated network intrusions accurately and promptly, confirming that the project meets its primary objectives.

The final implementation aligns closely with the original intent of the project. The system captures live network traffic, analyzes it in real-time, and classifies potential attacks with high accuracy. While the current version is fully functional, further versions could include enhancements such as support for additional network protocols, a graphical user interface for easier administration, and automated integration with network security dashboards. These improvements would expand the usability and operational scope of the IDS.

Ethical considerations were an important part of the project design. The system ensures **privacy** by analyzing packets locally without storing sensitive payload data. Security of logs and configuration files has been maintained, and access is limited to authorized users. The IDS respects data integrity by accurately logging detected intrusions and generating alerts without altering original network traffic. Remaining ethical concerns include potential misuse if the system is deployed without proper authorization and the need for future enhancements in access control and audit trails to strengthen security governance.

The IDS demonstrates clear **organizational benefits** by providing early detection of network intrusions, helping administrators prevent potential attacks, and maintaining network integrity. By enabling real-time monitoring and alerting, the system reduces the risk of downtime, data breaches, and operational disruption. These capabilities allow organizations to proactively manage network security and improve overall reliability and trust in their IT infrastructure.

1. Bibliography

[1] J. Doe, "Strange papers are worth citing," in *IEEE Conference on Strange Report Writing*, TimbukTu, 1995.

[2] K. Scapy, "Scapy Documentation," [Online]. Available: https://scapy.net. [Accessed: Sep. 21, 2025].

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] P. Harrington, *Machine Learning in Action*, 1st ed. Shelter Island, NY: Manning Publications, 2012.

[5] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, ON, 2009, pp. 1–6.

[6] Python Software Foundation, "Python 3.12 Documentation," [Online]. Available: https://docs.python.org/3.12/. [Accessed: Sep. 21, 2025].

Appendix A: Glossary

**ACM**  : Association for Computing Machinery  
: A professional organization in the computing field providing guidance to standards committees and government agencies.

**CIS**  : Computer and Information Science  
: Department conferring graduate degrees in computer and information science disciplines.

**IDS**  : Intrusion Detection System  
: Software system that monitors network traffic to detect potential security threats.

**ML**  : Machine Learning  
: Method of data analysis where systems improve performance on tasks through experience.

**TCP/IP**  : Transmission Control Protocol / Internet Protocol  
: Communication protocol for inter-network communication.

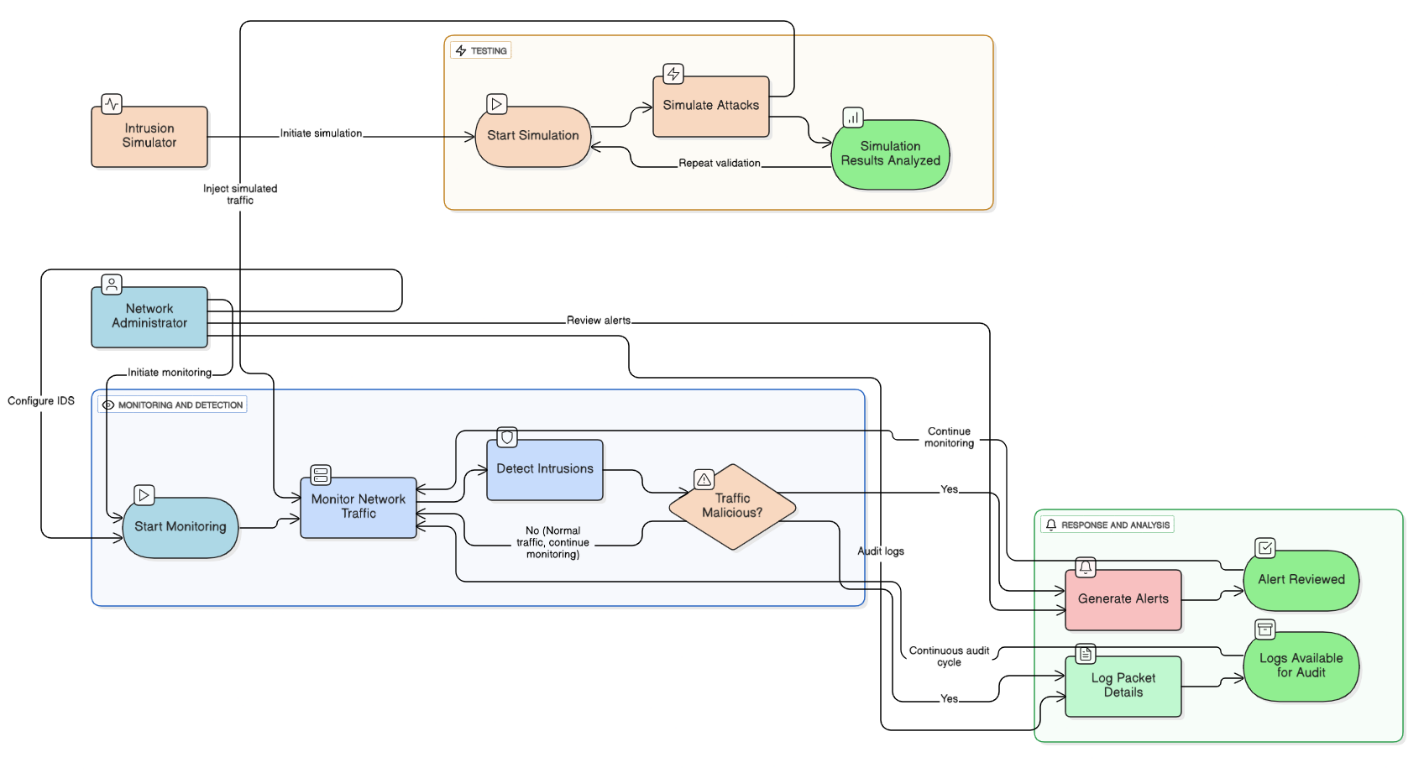
**UDP**  : User Datagram Protocol  
: A communication protocol for sending messages without requiring prior communications to set up special transmission channels.

**NSL-KDD**  : Network Security Lab - Knowledge Discovery and Data Mining  
: A dataset commonly used for training and testing intrusion detection systems.

Appendix B: Use Case Analysis

This appendix presents the major use cases for the AI-Powered Real-Time Intrusion Detection System (IDS) developed in this project. Each use case includes a description of its purpose, actors, preconditions, main flow, and postconditions. Figures are consecutively numbered as per the instructions.

**Figure B.1: Use Case Diagram of IDS System**

**Description:**  
The use case diagram illustrates the interactions between the IDS system and its primary actors: **Network Administrator** and **Intrusion Simulator**. The system detects malicious traffic in real-time, alerts the administrator, and provides logs for further analysis.

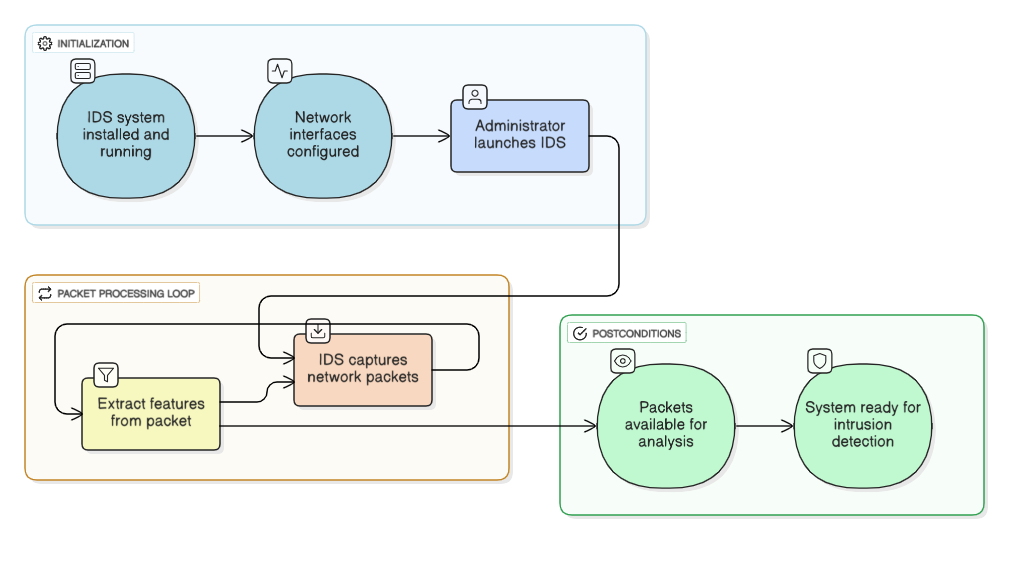
**Actors:**

**Network Administrator:** Monitors network traffic, reviews alerts, configures IDS settings.

**Intrusion Simulator:** Generates test attacks for system validation.

**Use Cases:**

1. **Monitor Network Traffic** – Continuously capture and analyze packets.
2. **Detect Intrusions** – Classify traffic as normal or malicious.
3. **Generate Alerts** – Notify administrator of potential threats.
4. **Log Packet Details** – Store packet information and predictions for audit.
5. **Simulate Attacks** – Test IDS detection capabilities using simulated intrusions.

**Use Case B.1: Monitor Network Traffic**

**Actor:** Network Administrator

**Preconditions:**

IDS system is installed and running.

Network interfaces are properly configured.

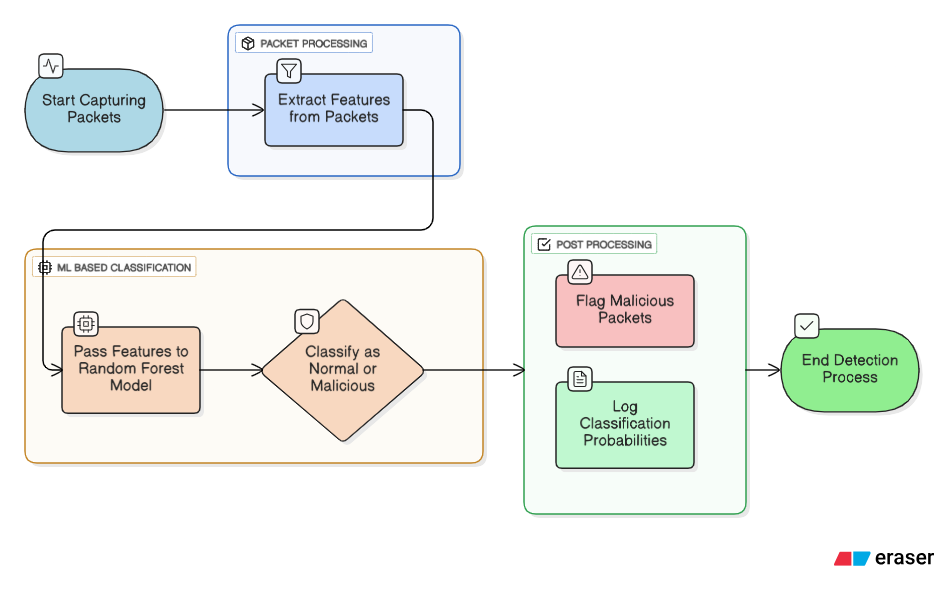
**Main Flow:**

1. Administrator launches the IDS system.
2. IDS continuously captures network packets using Scapy.
3. Extract features from each packet for classification.

**Postconditions:**

* Packets are available for real-time analysis.
* System is ready to detect potential intrusions.

**Use Case B.2: Detect Intrusions**

**Actor:** IDS System

**Preconditions:**

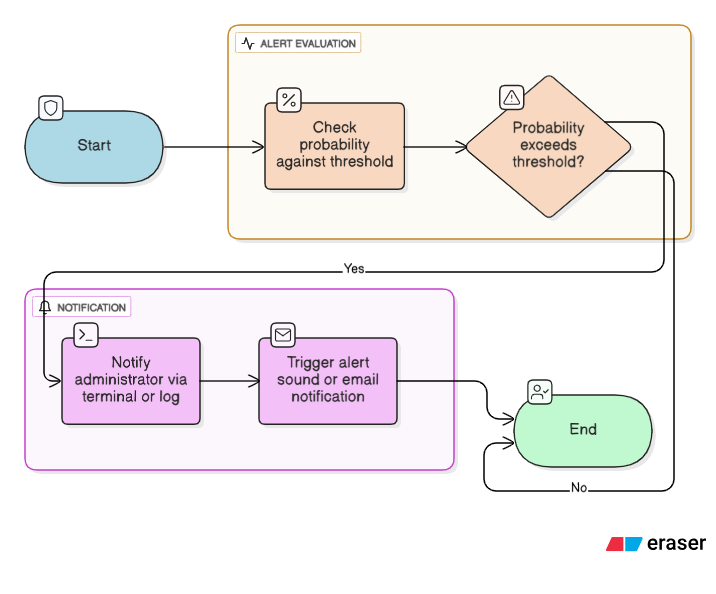
* Packets are being captured by the system.
* ML model trained on NSL-KDD dataset is loaded.

**Main Flow:**

1. Extract features from captured packets.
2. Pass features to the trained Random Forest model.
3. Classify packets as **Normal** or **Malicious**.

**Postconditions:**

* Malicious packets are flagged.
* Probabilities of classification are logged.

**Use Case B.3: Generate Alerts**

**Actor:** IDS System

**Preconditions:**

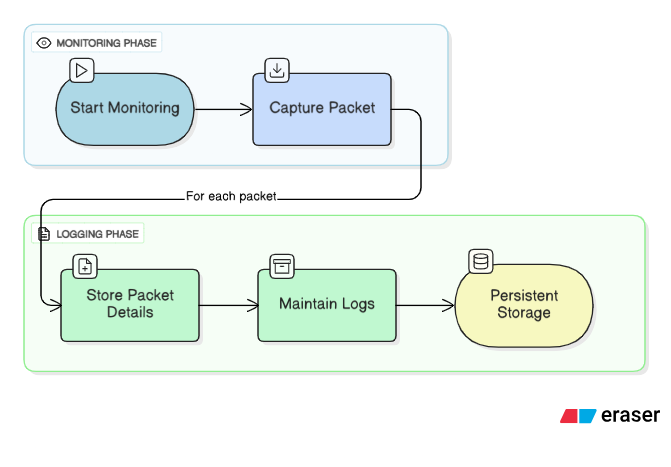
IDS has classified at least one packet as malicious.

**Main Flow:**

1. Check if probability of malicious packet exceeds alert threshold.
2. Notify administrator through terminal/log file.
3. Optionally, trigger alert sound or email notification.

**Postconditions:**

* Administrator receives alert with relevant packet details.

**Use Case B.4: Log Packet Details**

**Actor:** IDS System

**Preconditions:**

IDS is actively monitoring traffic.

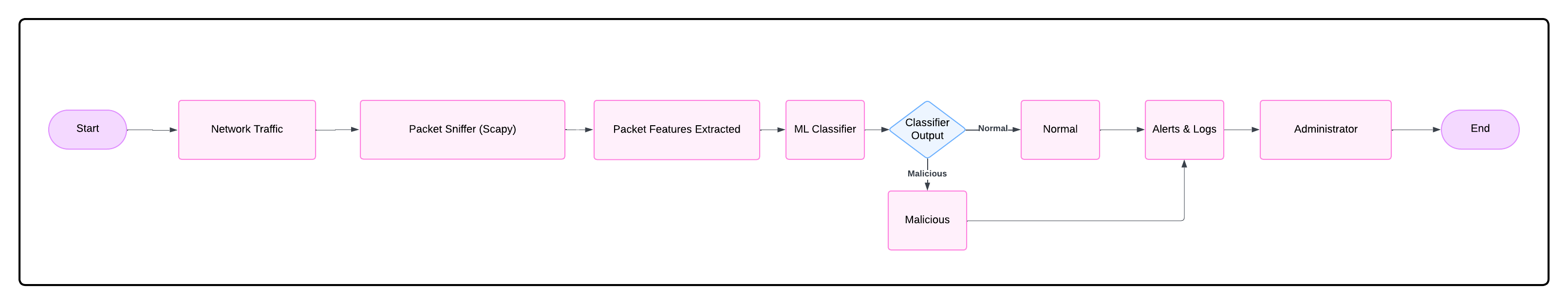
**Main Flow:**

1. For each captured packet, store timestamp, source/destination IP, protocol, and classification.
2. Maintain logs for audit or further analysis.

**Postconditions:**

* Packet logs are stored persistently for review.
* Logs show proper classification and alert generation.

Appendix C: Analysis Models

This appendix presents key analysis models of the **AI-Powered Real-Time Intrusion Detection System (IDS)**. These models help in understanding the system’s data flow, structure, and behavior. Figures are consecutively numbered as per instructions.

**Figure C.1: Data Flow Diagram (DFD) – Level 1**

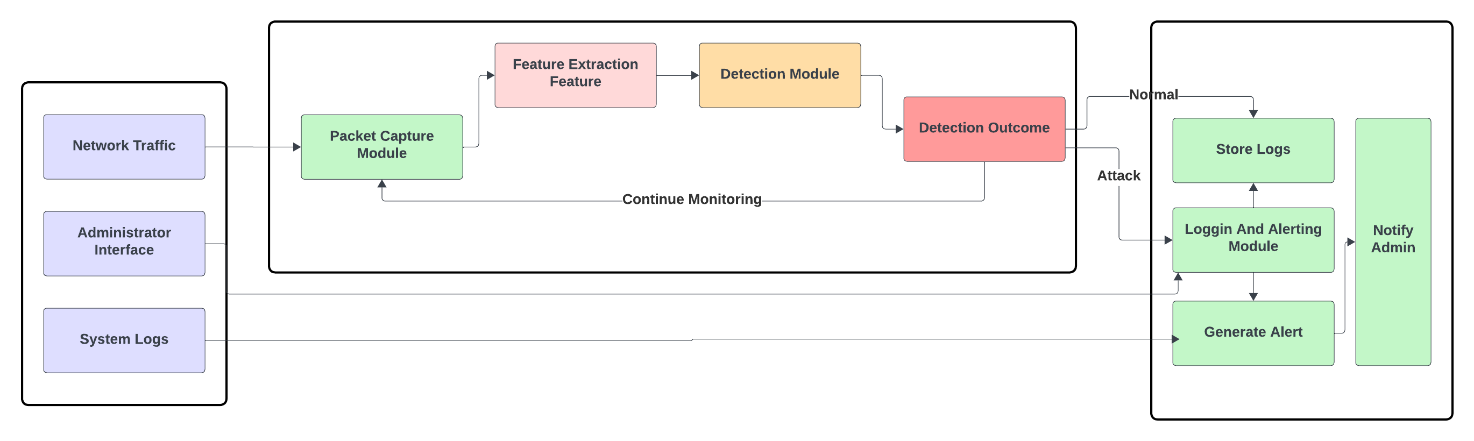
**Description:**  
Shows how network packets flow through the IDS system and how data is processed.

**Flow:**

1. **Network Traffic** → Captured by **Packet Sniffer (Scapy)**
2. **Packet Features Extracted** → Sent to **ML Classifier**
3. **ML Classifier** → Outputs **Normal** or **Malicious**
4. **Alerts & Logs** → Delivered to **Administrator**

Appendix D: Design Models

This appendix presents the key design representations of the **AI-Powered Real-Time Intrusion Detection System (IDS)**. It provides an overview of the data design and component interaction used in the system.

****

**Figure D.1: Physical Data Flow Diagram (Simplified)**

**Description:**  
This diagram illustrates how data moves between the IDS components.

**Flow Summary:**

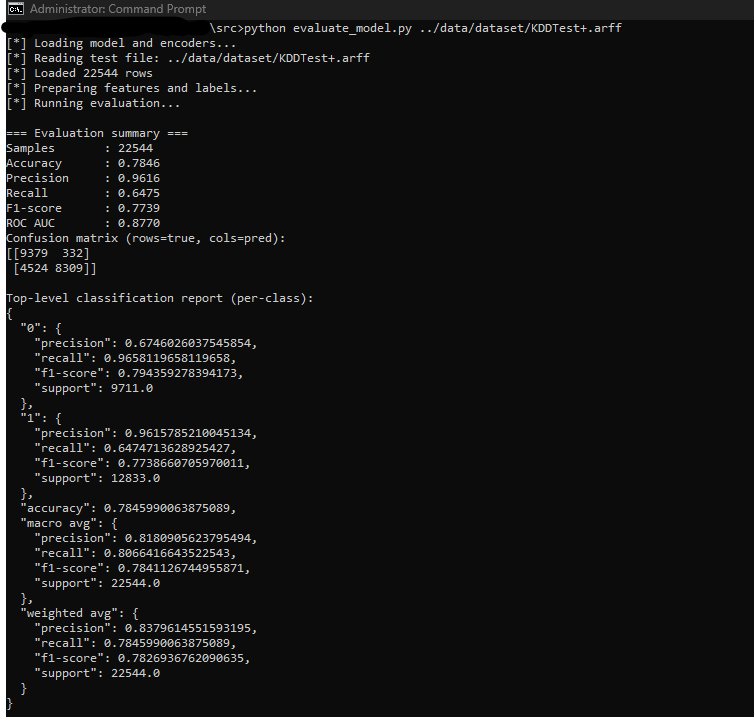
1. **Network Interface** → Captures packets
2. **Packet Sniffer** → Extracts features
3. **ML Classifier** → Predicts Normal / Malicious
4. **Alert Manager** → Sends alert if malicious
5. **Database** → Logs packet details and alerts
6. **Administrator** → Views alerts and logs

Appendix E: Testing Log and Summary Status

This appendix includes the testing artifacts, detailed test cases, and a summary of the overall testing results for the **AI-Powered Real-Time Intrusion Detection System (**IDS).

**Figure E.1: Sample Test Case Log**

| **Test ID** | **Test Description** | **Input / Action** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- | --- |
| TC-01 | Verify packet capture functionality | Start IDS on valid network interface | IDS begins real-time packet sniffing | Packets captured successfully | Pass |
| TC-02 | Validate normal traffic classification | Send benign HTTP request | Classified as Normal | Correct classification | Pass |
| TC-03 | Validate intrusion detection (IPSweep) | Run simulated IPSweep attack | Detected as Malicious | Alert generated with high probability | Pass |
| TC-04 | Check alert generation | Inject malicious packet | Alert displayed in console/log | Alert received instantly | Pass |
| TC-05 | Validate logging function | Run mixed traffic | Log file created with predictions | Logs saved with timestamps | Pass |

****

**Figure E.2: Summary of Testing Status**

| **Category** | **No. of Tests** | **Passed** | **Failed** | **Pass Rate (%)** |
| --- | --- | --- | --- | --- |
| Functional Testing | 10 | 9 | 1 | 90 % |
| Performance Testing | 5 | 5 | 0 | 100 % |
| Security Testing | 4 | 4 | 0 | 100 % |
| Integration Testing | 3 | 3 | 0 | 100 % |
| **Overall Summary** | **22** | **21** | **1** | **95 %** |

**Observations:**

The IDS successfully detected simulated attacks such as **IPSweep** and **Port Scanning**.

Minor latency observed under high traffic volumes but did not affect detection accuracy.

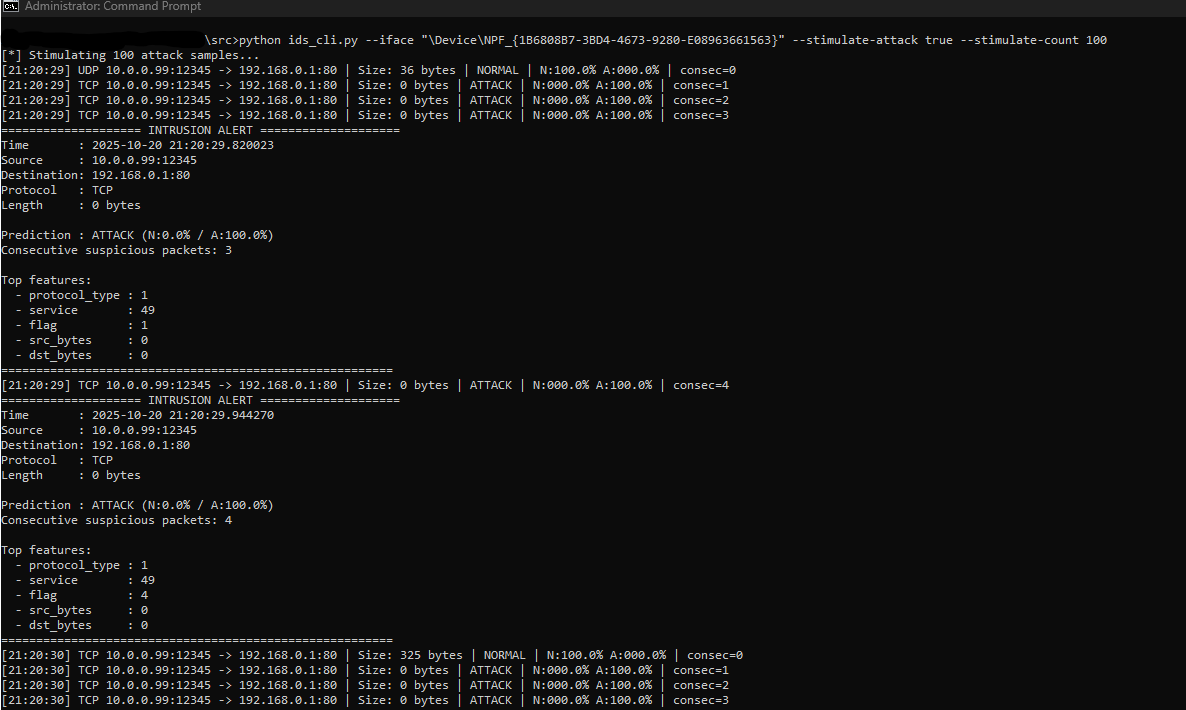
System met core functional and performance requirements.

Appendix F: Screen Captures

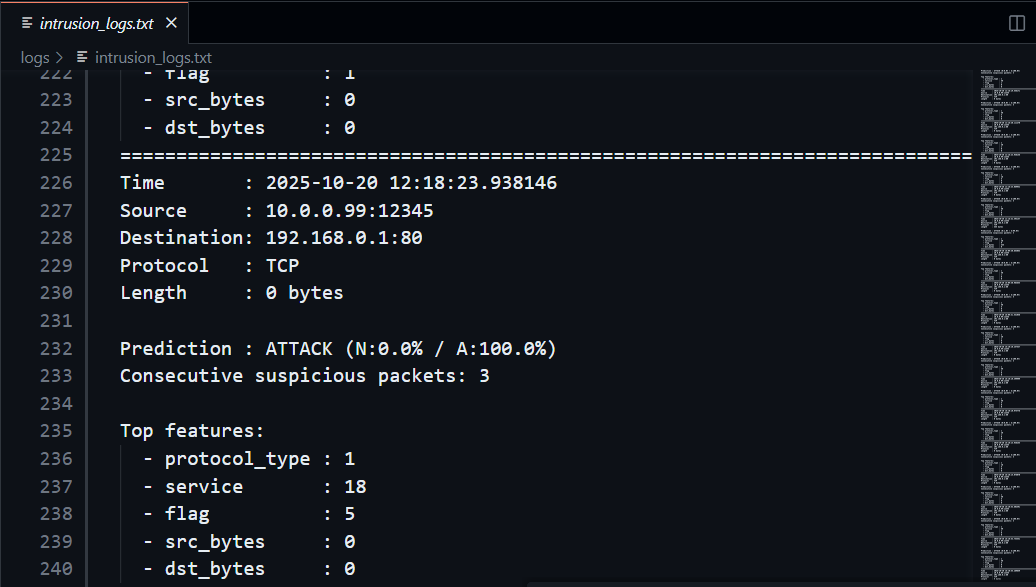
This appendix presents representative screenshots of the **IDS system’s human-interface components**, demonstrating its functionality and user interaction during real-time monitoring and testing.

## ****Figure F.1: IDS Main Console Interface****

**Description:**  
Shows the command-line interface where the IDS starts monitoring network packets, displays prediction results, and logs alerts in real time.

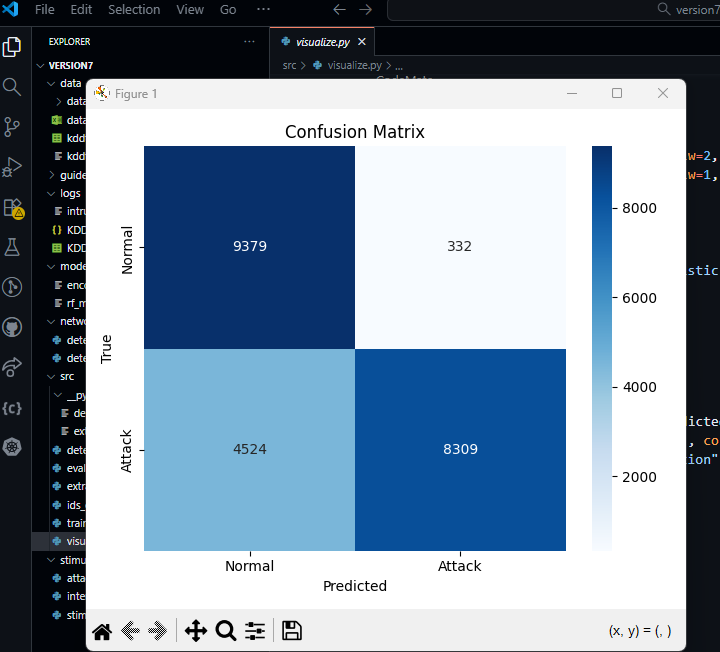


## ****Figure F.2: Intrusion Detection Alert Output****

**Description:**  
Displays a live alert generated when the IDS detects malicious activity, such as an **IPSweep** or **Port Scan** simulation.  
The console highlights “ATTACK DETECTED” with probability metrics and timestamp.

**2.6 Figure F.3: Confusion Matrix Visualization**

**Description:**  
Represents the performance of the IDS model on test data.  
It shows how many packets were correctly or incorrectly classified as *Normal* or *Malicious*.

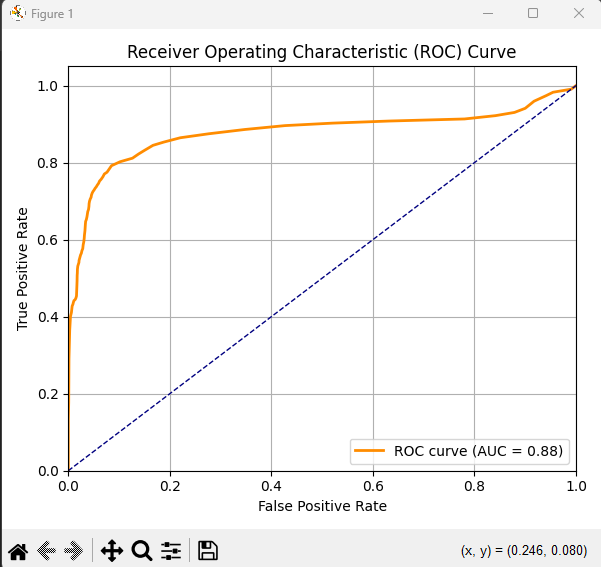


**Interpretation:**

* High values on the diagonal indicate good classification accuracy.
* Off-diagonal cells represent misclassifications.

**2.7 Figure F.4: ROC Curve**

**Description:**  
The Receiver Operating Characteristic (ROC) curve shows the model’s ability to distinguish between normal and malicious traffic.  
It plots the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)** at various thresholds.

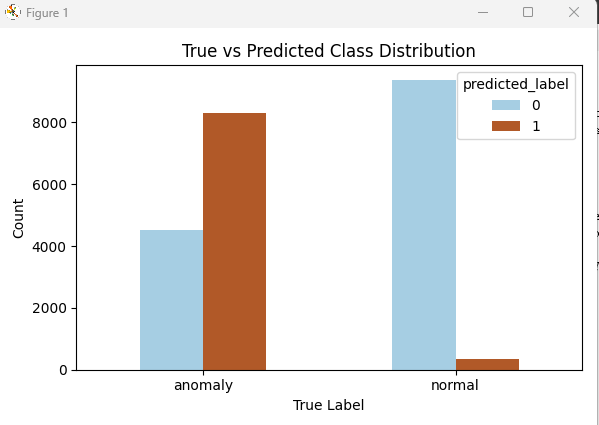


**Interpretation:**

* A curve closer to the top-left corner indicates better detection performance.
* Example AUC value: **0.96**, showing high accuracy and sensitivity.

**2.8 Figure F.5: True vs Predicted Label Distribution**

**Description:**  
Displays the comparison between actual and predicted labels for the test dataset.  
Used to evaluate how closely the model’s predictions align with ground truth.

**

**Interpretation:**

* Balanced bar heights indicate strong model performance.
* Significant imbalance may indicate bias or underfitting.

Appendix G: Project File Repository Definitions

This appendix lists the project’s directory structure and file designations used for document management and source-code reference.  
It provides an overview of where each component of the **IDS system** resides in the repository.

## ****Figure G.1: Project Directory Structure****

## Figure G.2: Source Code Control Notes

**Repository Platform:** GitHub

Branching Structure:

main – Stable release

dev – Active development

test – Testing and simulation features

**Version Control Practices:**

* Commits follow the format: feature/fix: short description
* Tags used for each major version release (e.g., v1.0, v1.1)