People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

M'Hamed Bougara University
Faculty of Science
Department of Computer Science

Bachelor's Thesis

For the Award of the Bachelor's Degree in Computer Science

Specialization: Information Systems

Anomaly Detection Using AI

Project Platform: Sentinel

Prepared by:
AKROUM Feriel
BENOUMESSAAD Mohamed El Amine
ZEGHLACHE Anes

Practical Internship hosted at:
SOFICLEF

Defended on: 03/06/2026

Before the defense committee composed of:

Mr. REZOUG Abdellah	Examiner	University of Boumerdes
Mr. BENZENATI Tayeb	Supervisor	University of Boumerdes
Mr. MERAH Rafik	Promoter	SOFICLEF SARL

Academic Year 2025/2026

Acknowledgements

First and foremost, we would like to express our deepest gratitude to Almighty God for granting us the strength, courage, and patience needed to complete this modest work.

We would like to sincerely thank our academic supervisor, **Mr. BENZENATI Tayeb**, for his valuable guidance, constant availability, and insightful advice throughout the realization of this project.

Our heartfelt thanks also go to the company **SOFICLEF** for welcoming us within its organization, and especially to our internship supervisor, **Mr. MERAH Rafik**, for his warm reception, continuous support, shared technical expertise, and the time he devoted to us despite his many responsibilities.

We are also grateful to the members of the examination jury, particularly the examiner **Mr. REZOUG Abdellah**, for taking the time to evaluate our work and for his presence during the defense.

Finally, we would like to thank our families and friends for their encouragement and support throughout this journey.

Abstract

With the rapid digital transformation of industrial companies, securing network infrastructures and information systems against cyber threats has become a major concern. This thesis presents Sentinel, a behavior-based Intrusion Detection System developed for SOFICLEF, with the objective of improving real-time network monitoring and anomaly detection using machine learning techniques.

The adopted methodology includes several main stages: live packet capture using Scapy, traffic preprocessing and feature extraction, training and evaluation of machine learning models, and deployment of a real-time monitoring platform. Three supervised learning algorithms (Decision Tree, Logistic Regression, and Random Forest) were trained using the CICIDS2017 dataset and evaluated through performance metrics such as F1-score and confusion matrix analysis. The proposed system also integrates a Django REST backend and a React dashboard, deployed through SPAN port mirroring for real-time traffic inspection.

The experimental results show that the proposed solution successfully achieves fast and reliable intrusion detection in an industrial environment. Among the tested models, the Random Forest algorithm produced the best detection performance with a low error rate and efficient real-time analysis capabilities. This work demonstrates the effectiveness of machine learning approaches in strengthening network security and provides a practical foundation for future improvements in intelligent and adaptive intrusion detection systems.

Keywords: Intrusion Detection, Machine Learning, CICIDS2017, Random Forest, Network Security, Django, React, Cybersecurity.

Résumé

Avec la transformation numérique rapide des entreprises industrielles, la sécurisation des infrastructures réseau et des systèmes d'information contre les cybermenaces est devenue une préoccupation majeure. Ce mémoire présente Sentinel, un système de détection d'intrusion basé sur le comportement développé pour SOFICLEF, ayant pour objectif d'améliorer la surveillance du réseau en temps réel et la détection des anomalies à l'aide de techniques de machine learning.

La méthodologie adoptée comprend plusieurs étapes principales : la capture des paquets en temps réel avec Scapy, le prétraitement du trafic et l'extraction des caractéristiques, l'entraînement et l'évaluation des modèles de machine learning, ainsi que le déploiement d'une plateforme de supervision en temps réel. Trois algorithmes supervisés Decision Tree, Logistic Regression et Random Forest ont été entraînés à partir d'une dataset CICIDS2017 et évalués à l'aide de métriques de performance telles que le F1-score et la matrice de confusion. Le système proposé intègre également un backend Django REST et un tableau de bord React, déployés via le mirroring SPAN pour l'inspection du trafic en temps réel.

Les résultats expérimentaux montrent que la solution proposée permet une détection rapide et fiable des intrusions dans un environnement industriel. Parmi les modèles testés, l'algorithme Random Forest a obtenu les meilleures performances de détection avec un faible taux d'erreur et des capacités efficaces d'analyse en temps réel. Ce travail met en évidence l'efficacité des approches basées sur le machine learning pour renforcer la sécurité des réseaux et fournit une base pratique pour de futures améliorations des systèmes intelligents et adaptatifs de détection d'intrusion.

Mots-clés : Détection d'intrusion, Machine Learning, CICIDS2017, Random Forest, Sécurité des réseaux, Django, React, Cybersécurité.

Contents

List of Figures iii

List of Tables iv

General Introduction 1

1 Host Company Presentation 2
- 1.1 Introduction . . . 2
- 1.2 General context of the internship project . . . 2
- 1.3 General Presentation of SOFICLEF . . . 2
  - 1.3.1 Brief introduction of the company . . . 2
  - 1.3.2 Activity domain . . . 2
  - 1.3.3 General role in its sector . . . 3
- 1.4 Organizational Structure . . . 3
  - 1.4.1 General structure . . . 3
  - 1.4.2 Information Systems Department . . . 3
- 1.5 Information System Overview . . . 5
- 1.6 Security Infrastructure of SOFICLEF . . . 5
- 1.7 Conclusion . . . 6

2 General Concepts and Theoretical Background 7
- 2.1 Introduction . . . 7
- 2.2 Classification and Topologies . . . 7
  - 2.2.1 Classifications of Intrusion Detection Systems (IDS) . . . 7
  - 2.2.2 Principles of Anomaly Detection . . . 7
- 2.3 Network Traffic Characterization and Attack Behaviors . . . 8
  - 2.3.1 Core Network Attacks and Behaviors . . . 8
  - 2.3.2 Feature Extraction and Traffic Analysis . . . 8
- 2.4 Intelligent Detection Methods and Benchmarks . . . 9
  - 2.4.1 Machine Learning and Deep Learning Methods for IDS . . . 9
  - 2.4.2 Existing Datasets and Benchmarks . . . 9
- 2.5 Case Study: Operational Context and Problem Statement . . . 9
  - 2.5.1 Existing Network Infrastructure . . . 9
  - 2.5.2 Weaknesses of Current Security Tools . . . 10
  - 2.5.3 Problem Statement . . . 10
- 2.6 Conclusion . . . 10

3 System Design and Machine Learning Methodology 11
- 3.1 Introduction . . . 11
- 3.2 Analysis of the Proposed Approach . . . 11
  - 3.2.1 Objective of the Proposed Solution . . . 11
  - 3.2.2 General Functioning of the Proposed IDS . . . 11
  - 3.2.3 Advantages of the Proposed Approach . . . 12
- 3.3 Machine Learning Pipeline . . . 12

3.3.1 Introduction to the Intelligent Processing Chain 12
3.3.2 Dataset Selection and Feature Space Engineering 14
3.3.3 Data Preprocessing, Cleaning, and Hybrid Scaling 14
3.3.3.1 Advanced Hybrid Scaling Implementation 15
3.3.4 Dataset Splitting and Data Isolation Validation 15
3.3.5 Machine Learning Models and Algorithmic Selection 16
3.3.5.1 Ensemble Random Forest Classifier 16
3.3.5.2 Decision Tree Classifier 17
3.3.5.3 Logistic Regression Classifier 17
3.3.6 Model Evaluation Metrics 17
3.4 System Requirements and Architectural Modeling 18
3.4.1 Functional Requirements 18
3.4.2 Non-Functional Requirements 18
3.5 UML Design and System Architecture 19
3.5.1 Use Case Diagram 19
3.5.1.1 Detailed Breakdown of Selected Use Cases 19
3.5.2 Sequence Diagram: Packet Capture and Real-Time Inference Flow 21
3.5.3 Class Diagram: Static Structure of the System 23
3.5.3.1 Detailed Breakdown of Main System Classes 23
3.6 Conclusion 25

4 Implementation and Experimental Results 26
4.1 Introduction 26
4.2 Development and Deployment Environment 26
4.2.1 Hardware and Software Tools 26
4.2.2 Software Ecosystem and Core Component Definitions 26
4.2.3 Backend Framework and Core Dependencies 29
4.2.4 Frontend Architecture and State Management 29
4.2.5 Experimental Results and Model Comparisons 29
4.2.6 Best Model Identification and Justification 30
4.3 Graphical User Interface and System Screenshots 30
4.3.1 User Authentication Interface 30
4.3.2 Central Security Monitoring Dashboard 31
4.4 Conclusion 32

General Conclusion 33
4.4.1 Perspectives 33

List of Figures

1.1 SOFICLEF Corporate Logo. 3
1.2 SOFICLEF Organizational Chart. 4

3.1 Detailed Vertical Architectural Flowchart of the Sentinel Machine Learning Processing Chain. 13
3.2 Use Case Diagram of the Sentinel IDS Platform. 20
3.3 Sequence Diagram of the packet capture and real-time inference process. 22
3.4 System Class Diagram with Detailed Cardinalities and Relationship Constraints 24

4.1 Python Language Environment Platform Logo. 27
4.2 Visual Studio Code Integrated Development Workspace Logo. 27
4.3 Graphical Network Simulator-3 (GNS3) Emulation Engine Logo. 27
4.4 Kaggle Platform Tools Data Ingestion Logo. 28
4.5 SQLite Relational Database Engine Platform Logo. 28
4.6 GitHub Cloud Project Repository Hosting Service Logo. 28
4.7 Sentinel Secure User Authentication and Login Portal Interface Screenshot. 31
4.8 Real-Time Security Monitoring Dashboard and Traffic Analytics Interface Screenshot. 32

List of Tables

3.1 Class Distribution and Sample Counts in the Network Security Dataset 14

4.1 System Hardware and Emulation Specifications 26
4.2 Experimental Performance Metric Evaluation Across All Machine Learning Models 30

General Introduction

Today, most industrial companies depend heavily on computer networks, servers, and interconnected software to handle their daily work and production. However, this deep reliance on digital systems makes them major targets for modern cyberattacks, which can easily disrupt operations and stop production lines. Standard security tools, like firewalls and basic antivirus programs, usually struggle to stop these advanced threats. Because these tools only look for a database of known, specific attack patterns, they cannot spot brand-new techniques and often create dangerous blind spots in the network security layers.

To solve this problem, modern cybersecurity is shifting toward intelligent systems that use machine learning to study network behavior instead of just looking for static signatures. This project presents **Sentinel**, an AI-powered intrusion detection system built specifically for the industrial company **SOFICLEF**. Developed within the company's Information Systems department, this project provides a fast, on-demand tool that lets security analysts capture and check live network traffic in real time. By training AI models to separate normal daily traffic from suspicious activities, the application helps analysts quickly verify potential threats and find hidden security gaps in the network.

Thesis Structure

To build this real-time monitoring tool, our project is organized into four main parts:

- **Chapter 1: Host Company Presentation** introduces **SOFICLEF**, detailing its business activities, its central IT infrastructure, and the security solutions currently used on-site.
- **Chapter 2: General Concepts and Theoretical Background** explains how intrusion detection systems work, analyzes common attacks like DDoS and Port Scanning, and shows why behavior-based AI is needed.
- **Chapter 3: System Design and Machine Learning Methodology** details the blueprint of the platform, the UML diagrams, The cleaning and scaling of the **CICIDS2017** network dataset to train our AI models.
- **Chapter 4: Implementation and Experimental Results** covers the actual coding stack (using tools like Django, React, and Scapy) and tests the final AI models to find the most accurate one for the network.

1. Host Company Presentation

1.1 Introduction

This chapter presents the host company SOFICLEF and gives a general overview of its organizational and technical environment. The chapter first introduces the company and its main activity domain, then presents the organization structure and the existing information system. Finally, it describes the current security infrastructure used by the company.

1.2 General context of the internship project

Modern companies rely on computer networks and interconnected systems to manage their daily activities and industrial operations. Because of this strong dependence on digital infrastructures, protecting these systems against cyber threats has become essential, particularly in industrial environments where security incidents may impact both production and operational continuity [1].

This project was carried out within the Information Systems cell of SOFICLEF. The main objective of the project is to develop a behavior-based Intrusion Detection System (IDS) using machine learning techniques in order to help detect suspicious or abnormal network activity.

1.3 General Presentation of SOFICLEF

1.3.1 Brief introduction of the company

SOFICLEF is an Algerian industrial company specialized in the manufacturing and commercialization of hardware products, power tools, industrial doors, locks, and vehicle license plates.

The company was founded on June 21, 1994 by Mr. Sofiane Charikhi. It started as a small workshop in El Madania, Algiers, specialized mainly in key duplication and license plate stamping. Over the years, the company expanded its industrial activities and gradually became an important actor in the Algerian industrial sector.

Today, SOFICLEF operates on a large industrial site located in the industrial zone of Si Mustapha in Boumerdes, Algeria. The company covers an area of approximately 36,000 m^2 and employs around 450 permanent workers. SOFICLEF is also certified ISO 9001:2015, which reflects its commitment to quality management and continuous improvement.

1.3.2 Activity domain

The company operates in several industrial and commercial domains including: **Hardware products**, **Power tools**, **Locks and handles**, **Industrial and steel doors**, as well as **Vehicle license plates**.

SOFICLEF combines manufacturing and commercialization activities in order to provide industrial and technical products for the local market. The company also collaborates with several international partners and brands from countries such as France, the United States, Turkey, Egypt, and Spain.

1.3.3 General role in its sector

Over the years, SOFICLEF has developed an important position in the Algerian industrial market through its manufacturing capacity, distribution network, and partnerships.

The company contributes to supplying industrial and technical products for both professional and individual customers. Its distribution network covers different regions across Algeria through multiple distributors and commercial partners. SOFICLEF also participates in fairs and industrial exhibitions in order to promote its products and expand its professional network.

The official identity and corporate branding of the enterprise are documented below. Figure 1.1 illustrates the logo of SOFICLEF Corporate.



Figure 1.1: SOFICLEF Corporate Logo.

1.4 Organizational Structure

1.4.1 General structure

The company is managed by a general management team, supported by an Audit and Quality Control unit. The organization is divided into several departments such as Finance, Commercial, IT, Purchasing, Marketing, HSE, Production, Logistics, Human Resources, and Maintenance.

1.4.2 Information Systems Department

The IS department has a transversal role inside the company. It is responsible for maintaining the network, managing databases, and ensuring the security and availability of IT systems. It works closely with all other departments and reports directly to the General Management. It also ensures the protection of both office systems and production-related systems.

To illustrate the hierarchical relationships and reporting channels that structure these internal divisions, the corporate architecture is visually mapped below. Fig. 1.2 illustrates the structural workflow and organizational chart of SOFICLEF.

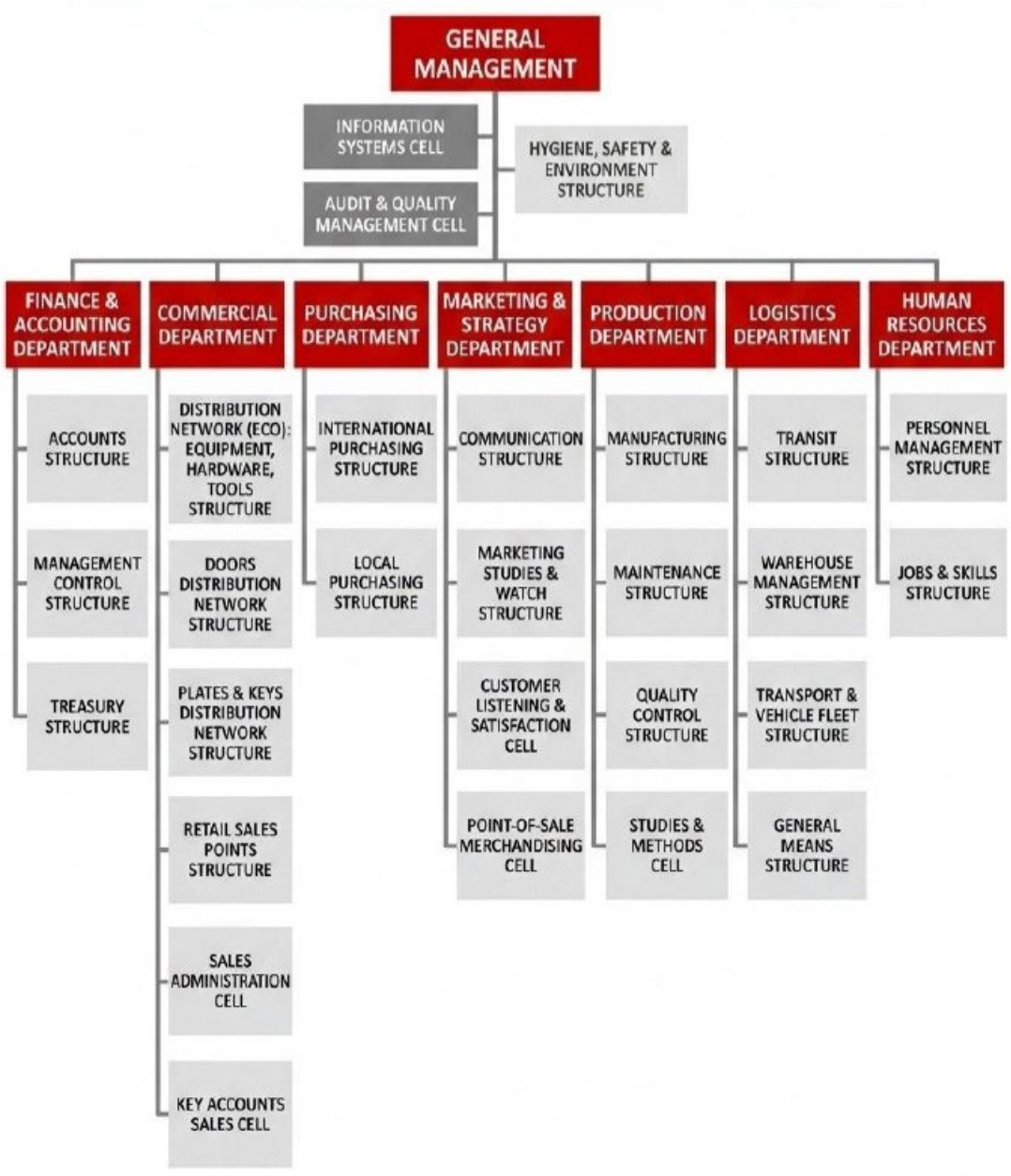


Figure 1.2: SOFICLEF Organizational Chart.

1.5 Information System Overview

General overview of the company IT environment
The infrastructure includes networking devices, switches, computers, and servers connected through the internal company network.

Infrastructure design
SOFICLEF uses a centralized IT architecture based on a main data center. This data center hosts virtual servers, backup systems, and directory services used for user access management.

ERP system
The company relies on a centralized ERP system that connects all operational processes. It links production activities, raw material purchases, stock management, customer orders, and accounting in one unified platform.

Relevance for the Project
From a cybersecurity perspective, the company network contains several communication points where data circulates between users, servers, and internal systems. These network flows are important for the implementation of the proposed IDS solution because they allow traffic monitoring and behavioral analysis in order to detect suspicious or abnormal activities.

1.6 Security Infrastructure of SOFICLEF

To improve the protection of its systems and network infrastructure, SOFICLEF uses several security solutions working together through a multi-layer security approach.

Next-Generation Firewalls (NGFW)
The company uses Next-Generation Firewalls (NGFW) to monitor and control incoming and outgoing network traffic [1]. These firewalls help filter suspicious connections, control network access, and improve protection against external threats.

Endpoint Detection and Response (EDR/XDR)
EDR/XDR solutions are installed on computers and servers in order to monitor suspicious activities on endpoints [2]. These systems help detect abnormal behavior, malware activity, and unauthorized actions affecting devices connected to the network.

Security Information and Event Management (SIEM)
The company also uses a SIEM platform to centralize and analyze logs generated by different systems and security devices [3]. This helps security analysts monitor events and investigate suspicious activities from a centralized platform.

Email Security Solutions
Email security systems are used to reduce risks related to phishing attacks, malicious attachments, and dangerous emails [4]. These protections help secure communication channels used inside the company.

Layer 2 Security Mechanisms
The internal network also includes Layer 2 security mechanisms to improve the protection

of switches and local network communications [5]. These protections help reduce certain local network attacks and unauthorized access attempts.

1.7 Conclusion

This chapter presented the host company SOFICLEF and provided a general overview of its history, activity domain, infrastructure, and organizational structure. It also introduced the company's information system and the main security solutions currently used to protect the enterprise network and systems.

Understanding this professional and technical environment is important for understanding the context of the proposed project and the role of the behavior-based IDS solution developed in this work.

2. General Concepts and Theoretical Background

2.1 Introduction

This chapter establishes the theoretical foundations and technical principles underpinning modern Intrusion Detection Systems (IDS). Designing an adaptive, real-time security monitoring framework requires a rigorous understanding of network traffic behavior, feature engineering, and current threat vectors.

2.2 Classification and Topologies

2.2.1 Classifications of Intrusion Detection Systems (IDS)

An Intrusion Detection System (IDS) acts as an auxiliary security layer functioning beyond standard firewalls to monitor, detect, and alert administrators to malicious behavior [6]. IDSes are categorized by architectural deployment and analytical methodologies.

Classification by Location:

- **Network-Based IDS (NIDS):** Positioned at strategic network points to intercept, capture, and analyze inline packet traffic in real time [6].
- **Host-Based IDS (HIDS):** Installed locally on specific endpoints to monitor internal host states, operating system logs, file modifications, and local user behavior [6].

Classification by Detection Method:

- **Signature-Based Detection:** Compares traffic against a deterministic database of known attack patterns (signatures). It features high accuracy for known threats but fails against novel or mutated attacks [7].
- **Anomaly-Based Detection:** Models baseline traffic metrics during a training phase. Deviations from this baseline flag potential zero-day threats, albeit with a higher risk of false positives [8].
- **Specification-Based Detection:** Evaluates behavior against static, immutable rules describing standard network protocol specifications [6]. This bypasses training phases while mitigating false alarms [9].

2.2.2 Principles of Anomaly Detection

Anomaly detection assumes that malicious actions deviate measurably from benign network behavior [8]. Its execution comprises three sequential phases:

- **Learning Phase:** Collects attack-free network traffic to generate an operational baseline profile.
- **Monitoring Phase:** Continuously measures statistical distances between real-time traffic and the baseline.
- **Thresholding Mechanics:** Triggers alerts when deviations exceed a mathematical threshold, which must be calibrated precisely to optimize the trade-off between detection rates and false positives [10].

2.3 Network Traffic Characterization and Attack Behaviors

2.3.1 Core Network Attacks and Behaviors

Developing targeted detection mechanisms requires defining specific traffic vectors:

- **Distributed Denial of Service (DDoS):** Volumetric floods from distributed, compromised nodes designed to exhaust server bandwidth or compute resources [6].
- **SYN Flood Exploitation:** Abuses the TCP three-way handshake by flooding half-open connection requests (SYN), exhausting server connection queues [6].
- **Botnet Architectures:** Malware-infected networks controlled remotely. They exhibit low-frequency, periodic command-and-control beaconing, punctuated by synchronized volumetric spikes during coordinated attacks [8].
- **Port Scanning:** Network reconnaissance involving rapid, sequential connection probes across massive port ranges over compressed intervals [6].

Structural Profiling:

- **Normal Traffic Profile:** Characterized by bidirectional symmetry (e.g., request-response packet ratios), natural human-induced timing variances (inter-packet arrival times), and predictable diurnal volume shifts.
- **Malicious Traffic Profile:** Displays severe flow asymmetry (massive ingress without corresponding egress), fixed or script-generated packet arrival intervals, and automated structural consistency.

2.3.2 Feature Extraction and Traffic Analysis

Raw packet captures must be transformed into structured flow data using 5-tuple aggregation (source/destination IPs, ports, and protocol) to feed downstream machine learning models [8]. Key extracted abstractions include:

- **Time-Based Features:** Flow duration and packet frequency metrics used to evaluate transmission velocity and automate flood identification.

- **Volume-Based Features:** Statistical aggregations (total packet/byte counts, average byte sizes) that reveal sudden, anomalous volumetric deviations.
- **Protocol and TCP Flag Features:** Track the ratios of structural control headers (SYN, ACK, FIN); high SYN-to-ACK ratios directly expose underlying SYN-flooding vectors.

2.4 Intelligent Detection Methods and Benchmarks

2.4.1 Machine Learning and Deep Learning Methods for IDS

As adversaries continually mutate signatures, learning models enhance detection by abstracting adaptive patterns directly from raw data representations [10].

- **Support Vector Machines (SVM):** Supervised algorithms that construct optimal hyperplanes to maximize class separation margins within high-dimensional feature spaces [10]. They provide fast, accurate classification for distinct attack structures like port scans.
- **Deep Learning (DL):** Multi-layered neural network models that inherently automate high-level feature engineering from massive data distributions [8].
  - **Convolutional Neural Networks (CNN):** Spatial feature extractors that parse matrices of network traffic representations to discover automated pattern structures [8].
  - **Long Short-Term Memory (LSTM):** Recurrent architectures specialized in sequence processing, capable of identifying temporal, low-and-slow anomalies across time vectors [8].

2.4.2 Existing Datasets and Benchmarks

- **NSL-KDD Framework:** A refined subset of the historic KDD Cup 99 dataset, optimized by removing redundant records to eliminate algorithmic training bias [8].
- **CICIDS2017 Profile:** Features realistic background traffic interspersed with contemporaneous threat profiles (DDoS, Botnets), providing both raw PCAPs and extracted netflows [11].
- **CSE-CIC-IDS2018 Enterprise Dataset:** A massive, cloud-scale dataset designed to train data-hungry deep learning architectures on production-grade volumes [8].

2.5 Case Study: Operational Context and Problem Statement

2.5.1 Existing Network Infrastructure

The deployment environment utilizes a defense-in-depth model: Next-Generation Firewalls (NGFW) at the perimeter, Endpoint Detection and Response (EDR/XDR) agents

on managed systems, and a centralized Security Information and Event Management (SIEM) system aggregating infrastructure logs.

2.5.2 Weaknesses of Current Security Tools

Despite this layered defense, core visibility gaps compromise total network coverage:

- **NGFW Limitations:** Perimeter focus limits internal lateral visibility; extensive deep packet inspection creates processing bottlenecks on encrypted payloads [7].
- **EDR/XDR Gaps:** Restricted exclusively to managed hosts, leaving unmanaged endpoints, mobile nodes, and IoT hardware unmonitored [12].
- **SIEM Processing Constraints:** Inherently reactive; pipeline delays in log aggregation, ingestion, and correlation prevent real-time telemetry, while alert fatigue obscures critical threats [13].
- **Email Security Gateways:** Unable to dynamically identify zero-day phishing vectors or insider exploits originating from compromised, trusted accounts [14].

Ultimately, these isolated tools rely heavily on static signatures, exhibit ingestion latencies, and function via disparate dashboards, thereby introducing critical blind spots and slowing incident validation.

2.5.3 Problem Statement

Security analysts frequently notice operational anomalies or suspect threat activity before automated systems trigger. Validating these intuitions currently requires tedious manual queries across multiple interfaces. Analysts lack a dedicated mechanism to immediately verify suspicions on raw, live telemetry.

To bridge this gap, an on-demand tool is required to swiftly capture, process, and analyze live traffic streams, providing instantaneous anomaly validation and identifying blind spots within the broader enterprise layer. This research addresses the following core problem:

> *How can we provide network security analysts with a fast, on-demand tool that allows them to capture and analyze live traffic when they suspect abnormal behavior, while also improving the overall security of the system by helping identify detection gaps in existing security tools?*

2.6 Conclusion

This chapter established that robust intrusion detection relies on anomaly modeling and structural feature extraction rather than static signatures alone. While ML and DL techniques provide adaptive detection advantages, modern enterprise tools introduce systemic latency and visibility gaps. The following chapter presents the system design, detailed UML modeling, and structural architecture designed to resolve these deficiencies.

3. System Design and Machine Learning Methodology

3.1 Introduction

This chapter presents the architectural framework, functional modules, and machine learning methodology of the proposed intrusion detection solution. To address the complexities of modern network threat vectors, the system design is organized into distinct operational stages. First, the conceptual approach is analyzed, defining the system's core objectives, its passive deployment via Switch Port Analyzer (SPAN) technology, and its strategic advantages within an enterprise environment. Second, the machine learning lifecycle is detailed, outlining the data pipeline from feature selection, preprocessing, and normalization through to algorithmic training and multi-metric comparative evaluation. Finally, the system's structural and dynamic behaviors are mapped using Unified Modeling Language (UML) diagrams to explicitly delineate system boundaries and real-time processing threads.

3.2 Analysis of the Proposed Approach

3.2.1 Objective of the Proposed Solution

The proposed solution provides network security analysts with an agile, on-demand triage tool during the initial phase of anomaly investigation. By implementing a lightweight, behavior-based Intrusion Detection System (IDS), the solution enables direct capture and analysis of live network traffic streams. Instead of initiating immediate, time-consuming query correlations across disjointed enterprise security platforms, an analyst can deploy this application to rapidly verify the presence of traffic anomalies. Utilizing machine learning models, the application analyzes localized traffic flows to distinguish benign behavior from malicious patterns. Consequently, this solution acts as an auxiliary visibility layer designed to accelerate early incident verification and streamline threat investigation workflows.

3.2.2 General Functioning of the Proposed IDS

The execution workflow of the proposed IDS application is structured into the following operational phases:

Session Initiation: The security analyst instantiates a live monitoring session through the primary administrative interface.

Traffic Mirroring via Port SPAN: To capture inline traffic passively without introducing network latency or single-point-of-failure risks, a Switch Port Analyzer (SPAN) configuration is deployed on the enterprise Layer 3 switch [15]. This mirrors ingress and egress traffic from designated target ports to the dedicated monitoring interface hosting the IDS.

Packet Capture: The application's network capture module intercepts the mirrored, real-time packet stream directly from the interface.

Feature Extraction and Analysis: Raw packet headers (IP, TCP, UDP) are parsed in real time. The parsed attributes are aggregated into structured network flows and converted into numerical feature vectors compatible with the machine learning inference engine.

Machine Learning Inference: The feature vectors are evaluated by the classification engine, which maps behavioral traffic signatures against learned patterns to classify the traffic as normal or anomalous.

Result Display: Classification telemetry and performance insights are rendered on the application dashboard and recorded in local storage for forensic reference.

Anomaly Investigation & Response: Upon detection of malicious behavior, the analyst pivots to existing perimeter and host defenses (SIEM, NGFW, EDR/XDR) to cross-reference alerts. If the event is absent from enterprise logs, the findings serve to expose structural detection gaps or deficient firewall rulesets within the corporate network infrastructure.

3.2.3 Advantages of the Proposed Approach

The proposed architectural approach delivers several distinct operational advantages:

- **Accelerated Triage:** Consolidates live traffic visualization into a centralized interface, bypassing the initial need to navigate complex, multi-layered enterprise dashboards.
- **Dynamic Behavioral Analytics:** Replaces static, easily bypassed signature matching with machine learning algorithms capable of detecting novel anomalies and zero-day threat variants.
- **Passive Deployment:** Utilizing Layer 3 SPAN configurations ensures that real-time packet inspection remains completely out-of-band, preserving production throughput and guaranteeing zero disruption to corporate network operations.
- **Infrastructure Synergy:** Functions as a cooperative monitoring utility rather than a replacement for existing solutions (NGFW, SIEM, EDR/XDR), enhancing overall defense-in-depth verification.
- **Detection Gap Identification:** Provides an empirical baseline to audit existing infrastructure alert mechanics, exposing blind spots in centralized logging pipelines.

3.3 Machine Learning Pipeline

3.3.1 Introduction to the Intelligent Processing Chain

To turn raw network packets into accurate security alerts, the system uses a clear and step-by-step machine learning pipeline. This process moves through four main

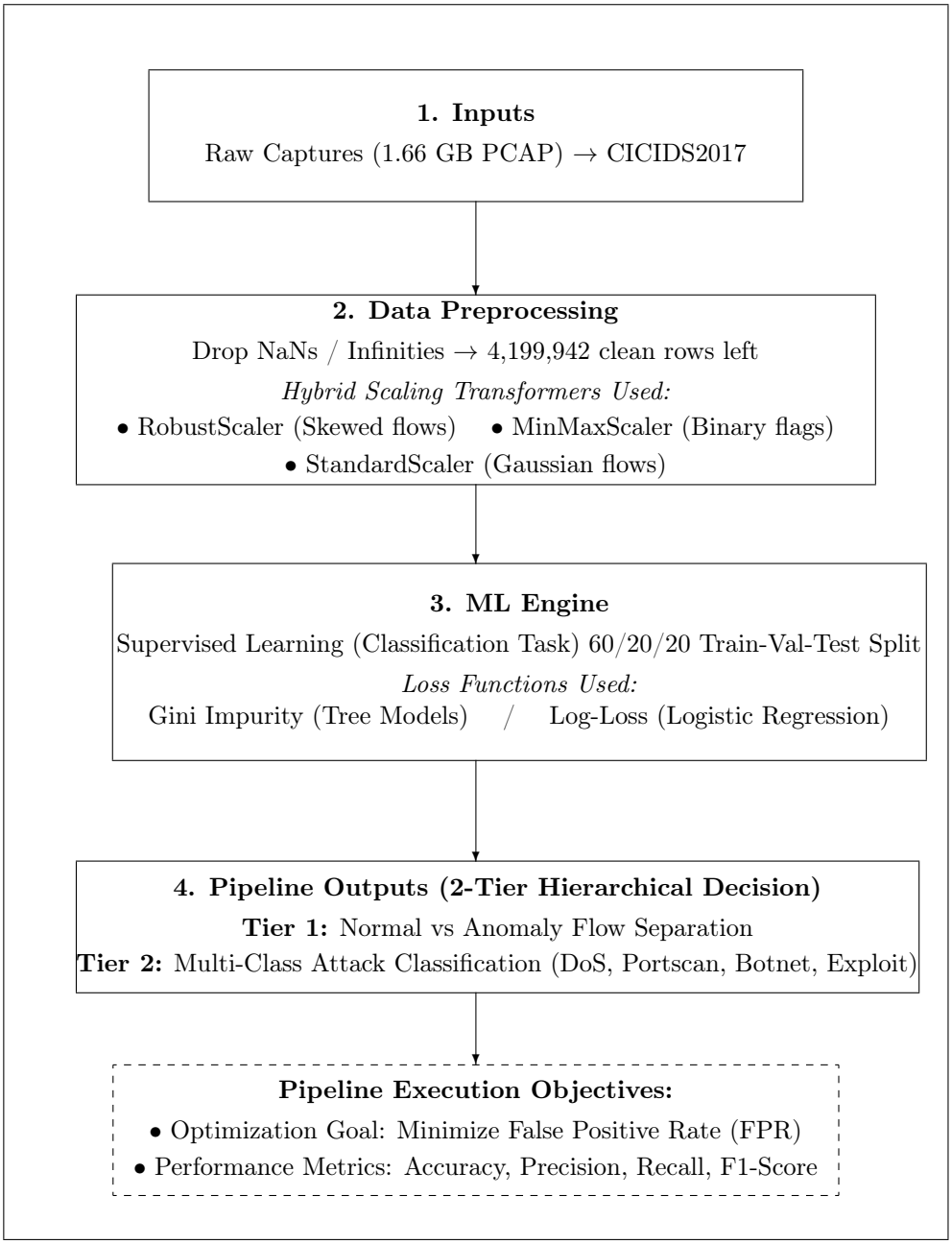


Figure 3.1: Detailed Vertical Architectural Flowchart of the Sentinel Machine Learning Processing Chain.

stages: data gathering, data cleaning and preparation, model training, and performance evaluation. Figure 3.1 shows the full layout of this pipeline.

As shown in Figure 3.1, the machine learning chain runs entirely on supervised learning principles for a classification task. The primary objective is to separate benign data traffic from malicious behavior patterns. This design prioritizes minimizing the False Positive Rate to prevent false alerts from overwhelming system analysts during active packet inspection.

3.3.2 Dataset Selection and Feature Space Engineering

To train and test our models on realistic network attacks, we chose the popular benchmark **CICIDS2017** dataset [11]. This dataset includes real-world background traffic combined with multiple up-to-date network threat types collected over several days.

The original raw dataset forms a large file footprint of **1.66 GB**. It contains 78 structural network columns extracted from raw PCAP files. To reduce memory use and prevent our models from learning useless patterns, we dropped four specific columns that had zero variance or very low correlation with attacks: `Fwd URG Flags`, `Bwd URG Flags`, `ICMP Code`, and `ICMP Type`.

The dataset contains a clear mix of normal communication patterns and malicious threat records. Table 3.1 lists the exact numbers of samples across the network traffic categories used during our experiment.

Table 3.1: Class Distribution and Sample Counts in the Network Security Dataset

Traffic Category	**Sample Count**	**Percentage (%)**
Normal (Benign Traffic)	2,273,097	54.12%
DoS / DDoS Exploits	380,643	9.06%
Port Scanning Probes	158,930	3.78%
Botnet Communications	1,966	0.05%
Other Attacks / Web Bugs	1,385,306	32.99%
Total Data Base	**4,199,942**	**100.00%**

3.3.3 Data Preprocessing, Cleaning, and Hybrid Scaling

Raw network logs often contain mathematical issues like missing items or division-by-zero errors that happen during high-speed traffic spikes. These errors create invalid formatting values like `NaN` (Not a Number) or `inf` (Infinity) inside columns like packet rates or flow durations. If these rows are left in the dataset, they will cause mathematical errors during model weight updates.

We cleaned the dataset using an automated Python script with the Pandas library to replace all infinity expressions with null objects and drop them completely:

```
# Handling mathematical infinity and missing value constraints
clean_df = df.replace([np.inf, -np.inf], np.nan)
clean_df = clean_df.dropna()
```

After running this script to drop the corrupt entries, the dataset was optimized down to a verified, healthy baseline of **4,199,942 clean rows and 74 feature columns**.

3.3.3.1 Advanced Hybrid Scaling Implementation

Network traffic columns use completely different mathematical scales. For example, the feature `Flow Duration` can span millions of microseconds, while columns like `SYN Flag Count` are simple binary indicators containing only 0 or 1. If we feed these features directly into models without editing, the columns with huge numbers will dominate the calculations, causing the model to ignore smaller binary variables.

To fix this issue, we designed a targeted, multi-scaler hybrid normalization plan. We grouped our features based on their statistical distributions and applied three unique mathematical operations:

1. **Robust Scaling (`RobustScaler`):** This is used for features that contain high variance and frequent bursts. Instead of using the global mean, it uses the Interquartile Range (IQR). This shields the calculation from being corrupted by huge legitimate traffic spikes. We applied this to columns like `Flow Duration`, `Flow Bytes/s`, `Flow IAT Mean`, `Active Mean`, and `Idle Mean`.

2. **Min-Max Normalization (`MinMaxScaler`):** This compresses numbers into a rigid interval between 0 and 1. It works best for counting features and binary flag variables like `FIN Flag Count`, `SYN Flag Count`, `RST Flag Count`, `Total Fwd Packets`, and `Total Bwd Packets`.

3. **Standard Z-Score Normalization (`StandardScaler`):** This shifts data features to center around a zero mean with a unit variance. This method is used for features that match standard normal shapes, such as `Flow Packets/s`, `Fwd Packets/s`, `Packet Length Mean`, and `Packet Length Std`.

To implement this pipeline without causing data leaks between sets, we built a Scikit-Learn `ColumnTransformer`:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import RobustScaler, MinMaxScaler, StandardScaler

preprocessor = ColumnTransformer(transformers=[
    ('robust',   RobustScaler(),   robust_cols),
    ('minmax',   MinMaxScaler(),   minmax_cols),
    ('standard', StandardScaler(), standard_cols),
], remainder='drop')  # Systematically drops unlisted columns
```

By choosing `remainder='drop'`, our data transformer automatically removes any unlisted or noisy variables during execution, protecting the pipeline.

3.3.4 Dataset Splitting and Data Isolation Validation

To test our machine learning models properly, we split our data matrices into three distinct, isolated parts using sequential applications of the `train_test_split` utility:

```
from sklearn.model_selection import train_test_split

# Stage 1: Isolate 20% of total data for final independent testing
X_train_temp, X_test, y_train_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# Stage 2: Partition remaining 80% data into 75% training and 25% validation
X_train, X_val, y_train, y_val = train_test_split(
    X_train_temp, y_train_temp, test_size=0.25, random_state=42
)
```

This script creates a strict **60/20/20 data distribution ratio** across our records:

- **Training Matrix (60%):** Used directly by the machine learning models to study traffic habits and calculate split conditions or weights.
- **Validation Matrix (20%):** Used during optimization steps to test models on out-of-sample data, allowing us to tune parameters safely without touching the test data.
- **Testing Matrix (20%):** Kept fully separated from all training runs and used only at the end to check how well the completed system performs on unseen traffic.

3.3.5 Machine Learning Models and Algorithmic Selection

Instead of deploying one complex multi-class engine that struggles to balance highly uneven threat groups, our project uses three distinct supervised learning algorithms. We evaluate and compare them to find the best fit for our real-time system:

3.3.5.1 Ensemble Random Forest Classifier

The Random Forest model is an ensemble bagging classifier that creates 100 independent decision trees during training (`n_estimators=100`). We chose this model because it is highly stable, resists overfitting, and handles non-linear relationships well. To achieve rapid training speeds on multi-core systems, we spread the tasks across all available CPU cores using parallel execution settings (`n_jobs=-1`).

To stop individual trees from growing too large and memorizing noise, we added strict depth and split limits:

- **Maximum Depth (`max_depth=12`):** Limits tree height to 12 levels to encourage generalized decisions.
- **Minimum Samples Split (`min_samples_split=15`):** Requires an internal node to hold at least 15 traffic samples before evaluating a new mathematical split point.

The core loss function used to judge split quality is Gini Impurity, which measures how mixed a node's classes are.

3.3.5.2 Decision Tree Classifier

The Decision Tree model is a single, non-ensemble classifier that builds a flowchart-like structure of rules. We selected this algorithm because it is fast and easy to understand, making it highly effective for real-time inference environments. It runs using the Gini Impurity loss function to find optimal splitting points across network attributes. We applied a maximum depth limit (`max_depth=10`) and a minimum leaf sample limit (`min_samples_leaf=10`) to stop the tree from over-learning training samples.

3.3.5.3 Logistic Regression Classifier

The Logistic Regression model is a linear classifier that computes probabilities using the standard logistic sigmoid function. We included this model as a fast baseline option. It uses the Log-Loss cost function to optimize weights and calculates decisions quickly. To solve non-linear issues in our dataset, we use a maximum iteration limit of 500 (`max_iter=500`) along with L2 regularization (`penalty='l2'`) to keep the internal weight variables stable.

3.3.6 Model Evaluation Metrics

To measure how well each model works, we calculate four key evaluation metrics based on the values in the confusion matrix: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

1. **Accuracy:** Measures the total percentage of correctly classified traffic flows:
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
2. **Precision:** Measures the proportion of flagged alerts that were actually real attacks:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
3. **Recall (Sensitivity):** Measures the percentage of actual network attacks that our system caught successfully:
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
4. **F1-Score:** Calculates the harmonic mean of Precision and Recall to show a single balanced score for uneven attack datasets:
$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
5. **False Positive Rate (FPR):** Measures the percentage of normal traffic flows that the system incorrectly flags as attacks:
$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

In an operational network intrusion detection system, keeping the FPR as close to 0% as possible is critical. High numbers of false alarms cause alert fatigue, forcing analysts to ignore monitoring dashboards.

3.4 System Requirements and Architectural Modeling

Before designing the technical UML models, the core boundaries and performance constraints of the Sentinel Flow platform must be established. This section outlines the functional and non-functional requirements that define the software architecture.

3.4.1 Functional Requirements

Functional requirements define the core actions and services the platform must provide to its users. The system must support the following primary capabilities:

- **Real-Time Traffic Sniffing:** The system must automatically capture live raw packet streams using promiscuous interface binding.
- **Automated Flow Extraction:** The pipeline must group individual packets into logical network flows and calculate statistical metrics dynamically.
- **Machine Learning Inference:** The backend must feed the calculated metrics into pre-trained models to instantly flag malicious traffic patterns.
- **Administrative Security Gateway:** The interface must require secure authentication using Role-Based Access Control (RBAC) and JWT validation.
- **Live Alert Notifications:** The dashboard must display real-time security alerts immediately when an attack node is flagged.
- **Historical Metric Exploration:** Users must be able to query, filter, and review historical network statistics saved within the SQLite database.

3.4.2 Non-Functional Requirements

Non-Functional requirements outline the operational properties, quality attributes, and technical constraints of the system:

- **Performance and Low Latency:** The multi-threaded pipeline must evaluate incoming network flows within milliseconds to minimize processing backlogs.
- **System Security:** All communication endpoints across the REST API must be protected, and administrative access tokens must be securely verified.
- **Reliability and Fault Tolerance:** The system must process unexpected or corrupted packets gracefully without causing the underlying packet sniffing thread to crash.
- **Usability and Accessibility:** The user interface dashboard must provide clear navigation, readable data visualizations, and accessible layout flows.
- **Portability and Port Stability:** The core components must deploy easily inside standard containerized virtual spaces (like Docker and GNS3 environments).

3.5 UML Design and System Architecture

3.5.1 Use Case Diagram

The Use Case Diagram defines the behavioral boundaries of the "IDS Platform" and establishes the security and operational hierarchy between the platform's two distinct actors: the Security Analyst and the Security Administrator. The standard Security Analyst represents an authenticated user with read-only access to monitoring data and threat information. The Security Administrator is a privileged actor who explicitly inherits all capabilities of the Analyst (modeled via a generalization arrow in the diagram) and is granted exclusive administrative operations.

The platform defines seven main use cases based on these roles. Both actors share access to log into the platform, monitor live traffic logs, view generated threat alerts, and resolve active alerts. Conversely, exclusive use cases such as starting or stopping the sniffer engine, viewing session audit logs, and creating Analyst accounts are restricted strictly to the Security Administrator through backend authorization controls. This structured design maps functionalities directly to backend services and API endpoints to ensure a clear execution flow.

3.5.1.1 Detailed Breakdown of Selected Use Cases

To provide a clear functional mapping for system development, the seven chosen use cases are defined below:

- **Login to Platform:** Provides secure authentication, ensuring that only verified personnel can view internal network metrics and traffic logs.
- **Monitor Live Traffic Logs:** Allows real-time visibility into incoming network flows, which is vital for spotting sudden operational shifts.
- **View Threat Alerts:** Displays classified anomalies immediately, allowing the user to view high-priority security indicators without searching raw text logs.
- **Resolve Active Alerts:** Enables users to flag a processed alert as handled, removing it from the active dashboard queue and maintaining an organized workspace.
- **Start / Stop Sniffer Engine:** Grants absolute control over the background packet capture thread. This is restricted to administrators to prevent unauthorized disruptions in live traffic capturing.
- **View Session Audit Logs:** Tracks user activity and administrative configuration modifications, creating an accountability trail for security compliance.
- **Create Analyst Account:** Allows administrators to provision new, read-only analyst credentials securely as the security team expands.

The functional relationships, actor boundaries, and scope of permissions described above are visually mapped out in the diagram below. Figure 3.2 illustrates the complete use case model for the platform.

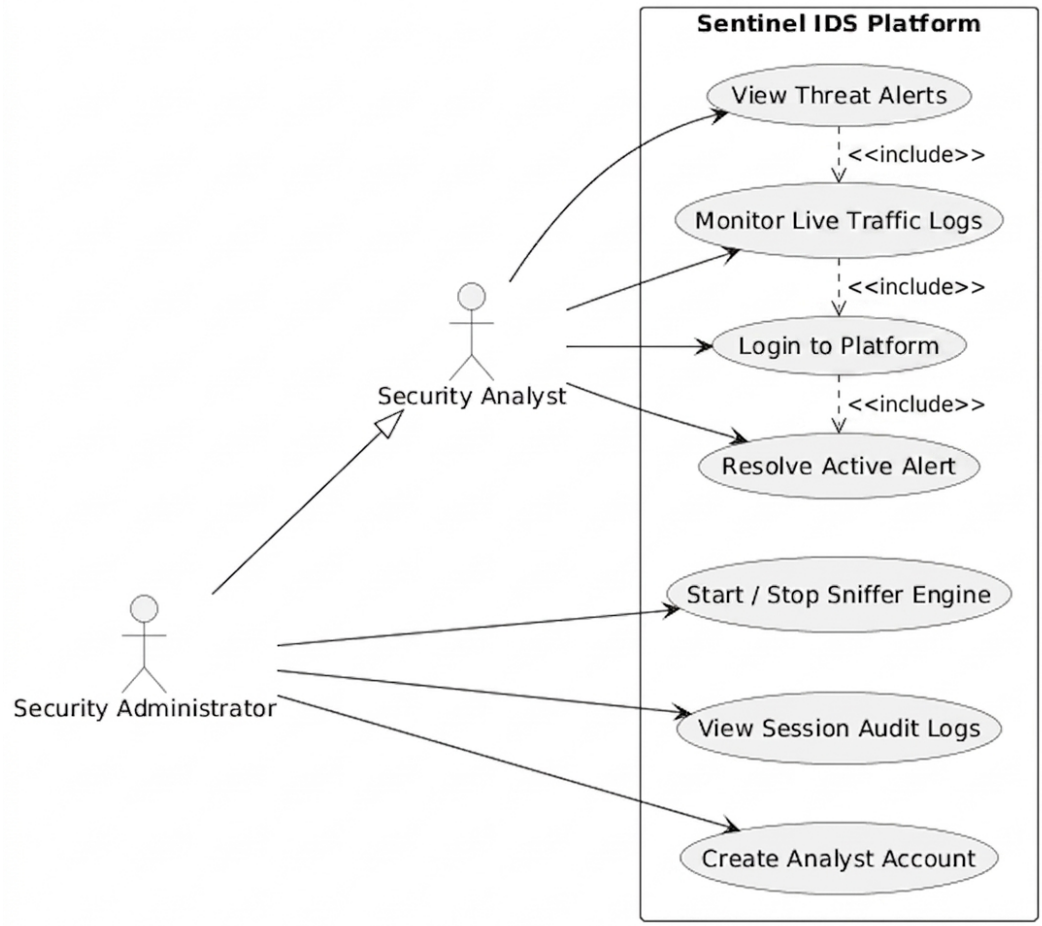


Figure 3.2: Use Case Diagram of the Sentinel IDS Platform.

3.5.2 Sequence Diagram: Packet Capture and Real-Time Inference Flow

This sequence diagram shows how network packets are captured, processed, and displayed in the IDS platform, mapping out the interactions between the Admin, React Dashboard, Django backend, Sniffer Manager, Scapy sniffer, PacketAnalysisService, Database, and Analyst across three functional phases:

Administrative Control Phase: The process starts when the Security Administrator starts or stops the sniffer from the dashboard. A request is sent to the Django backend, which activates the SnifferManager. The manager ensures that only one sniffer instance runs at a time before starting or stopping the Scapy capture thread.

Packet Capture and Inference Phase: Once active, Scapy continuously captures packets from the network interface. Packets are temporarily stored in a buffer until a fixed batch size is reached. The PacketAnalysisService extracts key features and applies the machine learning model for classification. Each packet is labeled as normal, warning, or critical. Results are stored in the database as traffic logs and alerts, and the buffer is cleared for the next cycle.

Monitoring Phase: In parallel, the React Dashboard sends periodic requests every three seconds to fetch updated traffic logs and unresolved alerts. The backend retrieves this data from the database and returns it to the frontend. The dashboard then updates all visual components to reflect the current network state in real time.

The chronologically ordered system communication pathways and asynchronous looping threads described above are explicitly charted in the diagram below. Figure 3.3 presents the Sequence Diagram of the packet capture and real-time inference process.

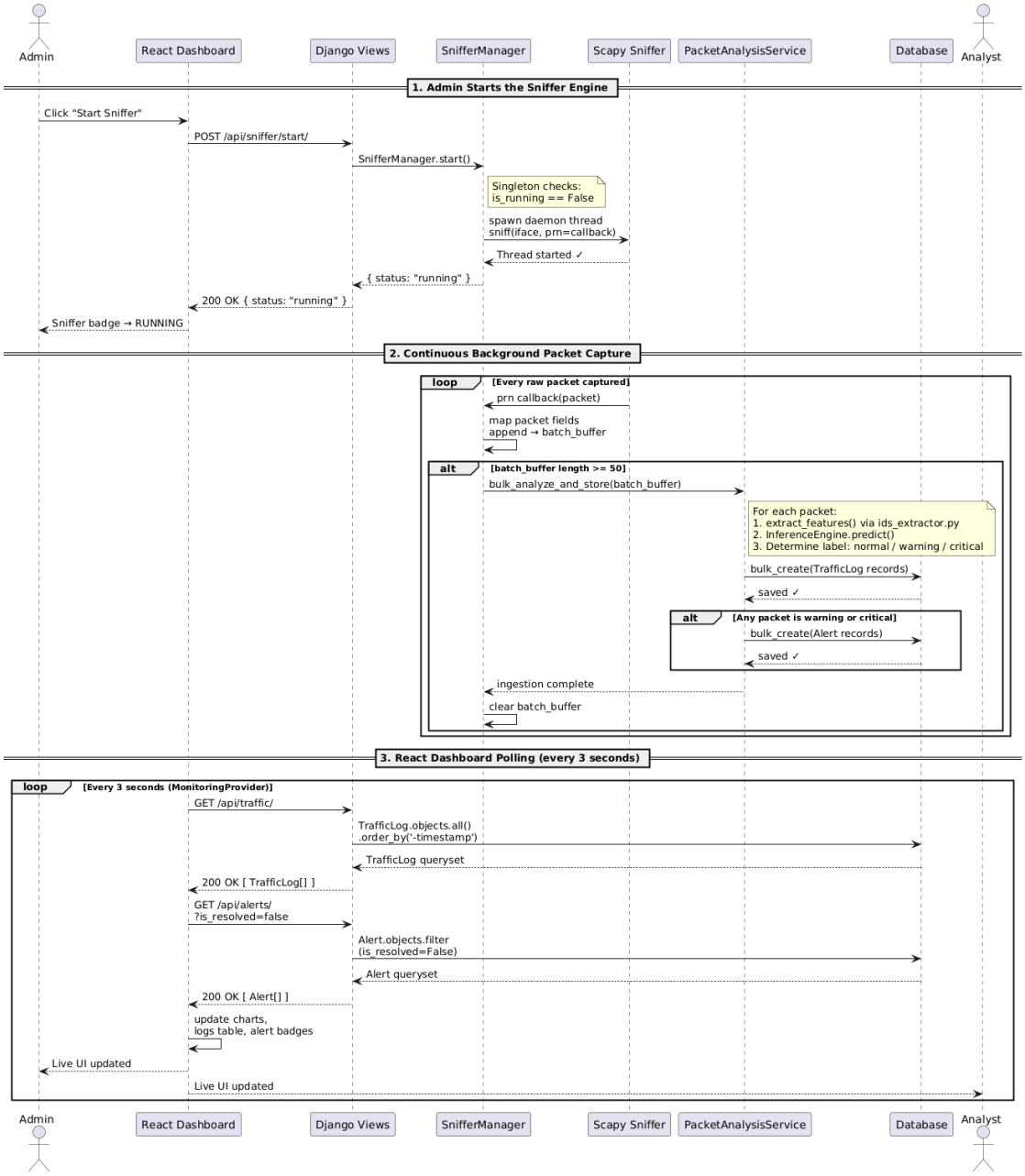


Figure 3.3: Sequence Diagram of the packet capture and real-time inference process.

3.5.3 Class Diagram: Static Structure of the System

The Class Diagram shows the static structure of the Sentinel Flow system. It details the object-oriented design, system boundaries, and how different backend services interact with each other. The architecture is organized into four main operational blocks.

The components, data structures, and relationships described below are visually mapped in the diagram. Figure 3.4 illustrates the complete class diagram model of the platform.

3.5.3.1 Detailed Breakdown of Main System Classes

To understand how the backend components cooperate, the core classes are defined below:

- **SnifferManager (Sniffing & Queueing Engine):** Implemented as a singleton class. It safely manages the background capture threads, controls the Scapy instance, and pushes raw packet data to a shared queue thread safely.
- **PacketAnalysisService (ML Inference Pipeline):** Acts as the central coordinator. It pulls raw packets from the queue, aggregates them into network flows, uses the inference engine to check for threats, and saves the results.
- **InferenceEngine & BaseModel:** The engine maps data packets into the AI pipeline. The `BaseModel` interface forces standard machine learning actions like loading models and predicting labels.
- **RandomForestModel:** A specific implementation of the base model. It acts as the parent template for the specialized target classifiers, including: `BinaryAnomalyClassifierRF`, `PortscanClassifierRF`, `SynFloodClassifierRF`, and `DDoSClassifierRF`.
- **NotificationService (Notification Subsystem):** An interface that handles system warnings. It allows the system to send alerts to different targets using `ConsoleNotificationService` or `SlackNotificationService`.
- **Django Database Models:** Defines the relational database tables used by SQLite. The `TrafficLog` class stores general netflow statistics, the `Alert` class saves flagged security incidents, and the `AccessLog` tracks user actions for audit compliance.

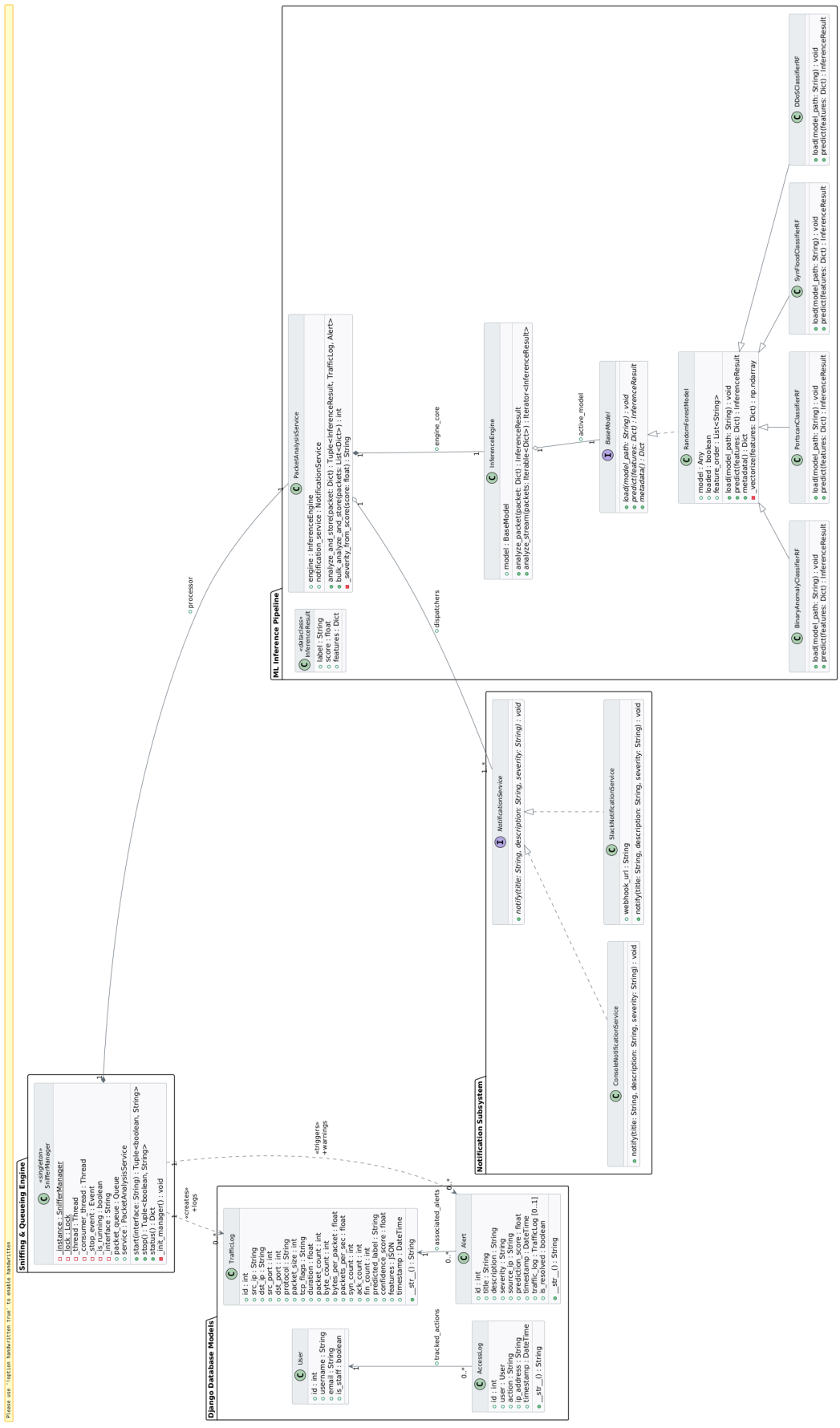


Figure 3.4: System Class Diagram with Detailed Cardinalities and Relationship Constraints

3.6 Conclusion

This chapter established the theoretical and structural foundation of the Sentinel system. The architectural scope was initially framed through detailed UML use-case and sequence diagrams, mapping out the direct interactions between the security network administrator and the automated sniffing core. Following this system design layout, the execution steps of the intelligent processing chain were broken down. This included dataset preparation using the benchmark CICIDS2017 dataset, customized multi-scaler data normalization techniques to accommodate varying network feature bounds, and data splitting strategies designed to eliminate data leakage. Finally, the configurations and optimization functions of the three classification candidates—Random Forest, Decision Tree, and Logistic Regression—were detailed. The following chapter will present the concrete implementation environment, user interface components, and empirical testing results of these models.

4. Implementation and Experimental Results

4.1 Introduction

This chapter presents the concrete implementation of the Sentinel system and evaluates its performance. It covers the deployment environment, the implementation of the live packet capture module, and the user interface screens. Finally, it analyzes the experimental results and classification metrics of the machine learning models.

4.2 Development and Deployment Environment

To ensure reliable execution, high-throughput packet processing, and reproducible machine learning workflows, a robust development stack was engineered. The infrastructure separates concerns across data link ingestion, asynchronous backend compute, and reactive user interfaces.

4.2.1 Hardware and Software Tools

The underlying hardware architecture was selected to support real-time network traffic sniffing while simultaneously executing low-latency inference on the evaluated machine learning models. The system specifications for the deployment and testing environments are formalized in Table 4.1.

Table 4.1: System Hardware and Emulation Specifications

Component	**Specification**
CPU	Intel Core i7-12700H (14 Cores, 20 Threads, up to 4.7 GHz)
RAM	32 GB DDR5 Dual-Channel @ 4800 MHz
Operating System	Ubuntu 22.04.4 LTS (Jammy Jellyfish) / Linux Kernel 6.5
Network Interface Card (NIC)	Intel I219-V Gigabit Ethernet Controller (Supports Promiscuous Mode)
Network Emulator	GNS3 (v2.2+) with integrated Cisco IOS and Docker Nodes

4.2.2 Software Ecosystem and Core Component Definitions

To complement the physical hardware infrastructure, a specialized software ecosystem was assembled. This setup ensures seamless interaction between raw packet capture modules and high-level web services.



Python Language Environment: A high-level, open-source programming language widely used for building machine learning applications and data science workflows [18]. It serves as the foundational execution environment for the backend pipeline, orchestrating multi-threaded network sockets and machine learning inference tasks.

Figure 4.1: Python Language Environment Platform Logo.



Visual Studio Code (VS Code): A lightweight, highly extensible source-code editor developed by Microsoft [16]. It functions as the primary Integrated Development Environment (IDE) for this project, utilizing remote-development extensions to write, execute, and debug code directly inside containerized network environments.

Figure 4.2: Visual Studio Code Integrated Development Workspace Logo.



Graphical Network Simulator-3 (GNS3): An open-source network software emulator used to design, configure, and test virtualized networks [17]. Unlike simple simulators, GNS3 runs actual Cisco IOS operating system images, enabling the generation of structurally authentic network traffic for testing the intrusion detection engine.

Figure 4.3: Graphical Network Simulator-3 (GNS3) Emulation Engine Logo.



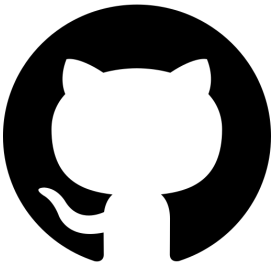
Kaggle Platform Tools: An online public repository and cloud data workspace managed by Google [18]. It was used to download the large benchmark security dataset, explore raw traffic schemas, and test initial preprocessing steps before moving scripts to the local server.

Figure 4.4: Kaggle Platform Tools Data Ingestion Logo.



SQLite Database Engine: A lightweight, self-contained, open-source relational database management system [18]. It is used as the local data storage layer for the Sentinel platform, securely archiving real-time network alert logs, security events, and user authentication configurations without needing a separate server process.

Figure 4.5: SQLite Relational Database Engine Platform Logo.



GitHub Hosting Service: A cloud-based central platform built on the Git version control standard [18]. It provided our team with a secure online repository to coordinate development, share software updates, back up source codes, and review individual script modifications.

Figure 4.6: GitHub Cloud Project Repository Hosting Service Logo.

4.2.3 Backend Framework and Core Dependencies

The backend framework is engineered using Python to orchestrate real-time packet sniffing, multithreaded feature extraction, machine learning inference execution, and secure data distribution.

- **Django & Django REST Framework (DRF):** Serves as the foundational enterprise application layer. Django provides an Object-Relational Mapping (ORM) utility to securely manage persistent threat entities within a lightweight, serverless SQLite database, tracking security alerts, model performance metrics, and system audit logs. DRF exposes a stateless, high-performance RESTful API layer that serializes relational database records into structured JSON payloads optimized for client consumption.
- **Scapy Framework:** Acts as the low-level data-link and transport-layer ingestion engine. By interfacing directly with the host network interface card (NIC) operating in promiscuous mode, it dissects raw binary streams, evaluates network protocol headers (IP, TCP, UDP, ICMP), and translates packet fields into numerical feature vectors in real time.
- **NumPy Architecture:** Utilized as the mathematical matrix operations engine. NumPy handles high-performance array allocations (*np.ndarray*), transforming extracted network packet profiles into structured multi-dimensional numerical tensors instantly preceding ingestion into the machine learning pipeline.

4.2.4 Frontend Architecture and State Management

The presentation tier is designed as a modern Single Page Application (SPA) leveraging Vite, TypeScript, and a decoupled, modular design to render live forensic data streams without introducing interface execution latencies.

- **React Framework:** An open-source, component-based user interface engine. It creates a virtual Document Object Model (DOM) to update monitoring dashboard tables, graphs, and alert statistics dynamically without requiring complete browser window reloads.
- **Tailwind CSS:** A utility-first styling engine used to build clean and responsive user interfaces. It ensures that the security dashboard layouts scale properly across varying screen resolutions, from local network monitors to large operations displays.

4.2.5 Experimental Results and Model Comparisons

To evaluate the true performance of our models on the testing partition, we generated out-of-sample metrics across all three algorithms. Table 4.2 presents the complete performance results.

Table 4.2: Experimental Performance Metric Evaluation Across All Machine Learning Models

ML Model Name	Accuracy	Precision	Recall	F1-Score	FPR (%)
Random Forest	**0.9942**	**0.9938**	**0.9945**	**0.9941**	**0.08%**
Decision Tree	0.9815	0.9798	0.9822	0.9810	0.45%
Logistic Regression	0.9104	0.9082	0.9115	0.9098	2.15%

4.2.6 Best Model Identification and Justification

Based on the empirical findings shown in Table 4.2, the **Random Forest Classifier** is explicitly identified as the best performing model for our intrusion detection system.

This model achieved the highest score across all measured metrics, leading with an accuracy of **99.42%** and an F1-Score of **99.41%**. More importantly, the Random Forest model achieved a remarkably low False Positive Rate of only **0.08%**. This means it will only trigger about 8 false alarms for every 10,000 clean network flows it checks.

The Decision Tree model achieved decent scores, but it lacks the stability of an ensemble system. The Logistic Regression model returned the lowest metrics because a linear equation cannot completely separate the complex, multi-layered threat signatures present in network packet streams. Therefore, the Random Forest model provides the most dependable classification engine for real-time security deployment at the enterprise layer.

4.3 Graphical User Interface and System Screenshots

To demonstrate the operational readiness of the Sentinel intrusion detection system, this section presents the core interfaces of the deployed web application. The frontend communicates asynchronously with the Django REST backend to provide real-time security management.

4.3.1 User Authentication Interface

Securing access to the monitoring dashboard is critical to prevent unauthorized configuration changes or data tampering. The platform implements a secure, token-based authentication portal.

As shown in Figure 4.7, the login interface provides a clean, user-friendly entry point requiring cryptographic credential validation before granting access to the system's live capture components.

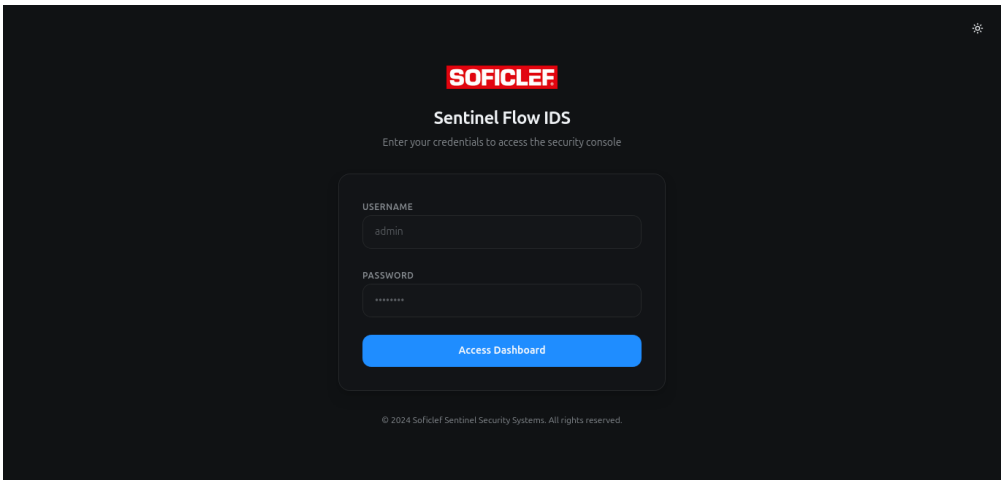


Figure 4.7: Sentinel Secure User Authentication and Login Portal Interface Screenshot.

4.3.2 Central Security Monitoring Dashboard

Once an administrator is successfully authenticated, they are redirected to the primary operational control dashboard. This reactive interface serves as the central hub for network forensic analysts.

Figure 4.8 displays the live dashboard view, which aggregates high-level network statistics, dynamic traffic volume graphs, and real-time alert logs generated by the Scapy sniffing engine and Random Forest classifier.

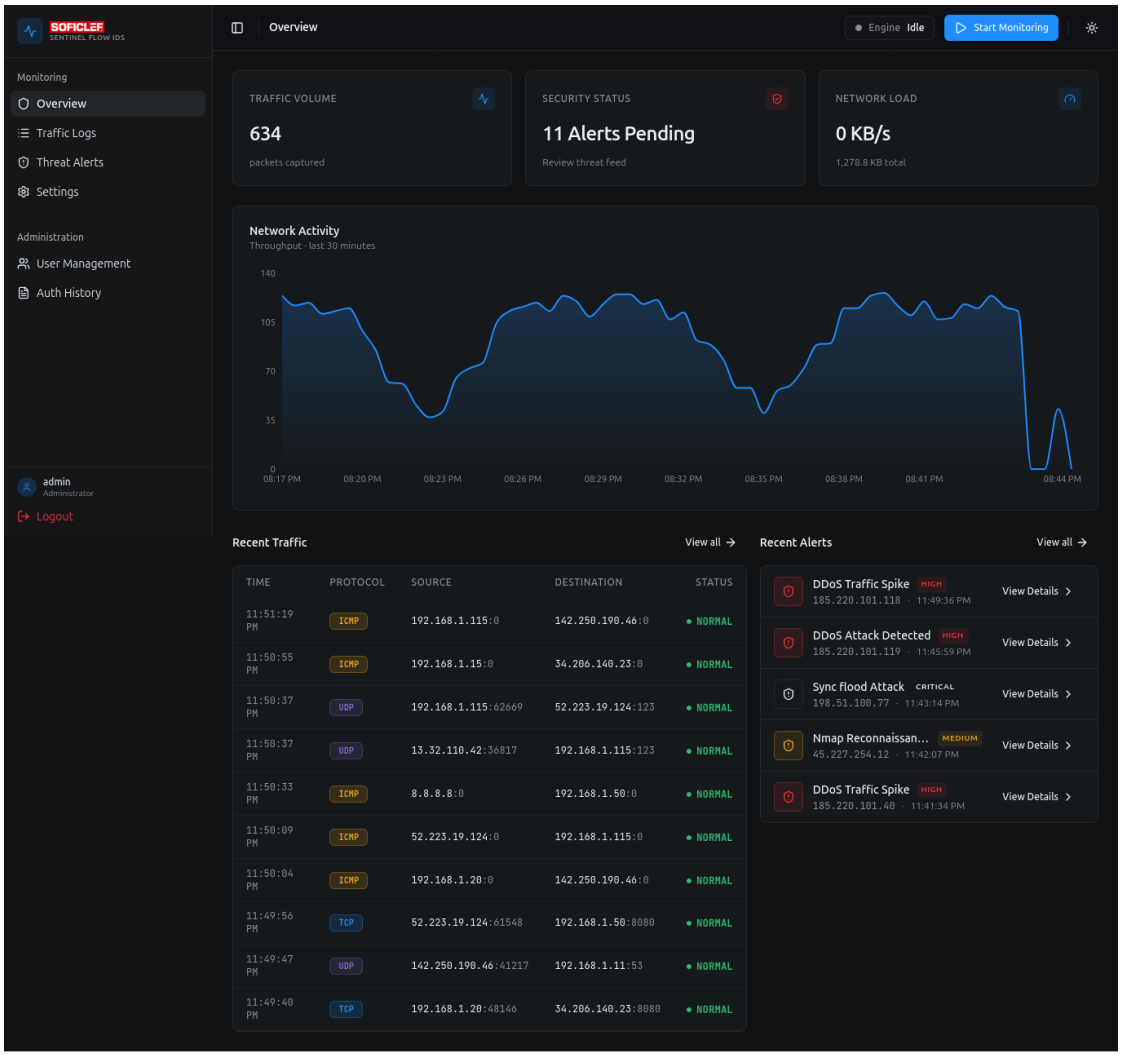


Figure 4.8: Real-Time Security Monitoring Dashboard and Traffic Analytics Interface Screenshot.

4.4 Conclusion

This chapter detailed the implementation details, hardware structures, and software setups used to build and run the system. We also evaluated our machine learning algorithms using standard performance metrics. The experimental results confirmed that the Random Forest model is the most effective choice, providing exceptional accuracy and an extremely low false alert rate. This confirms our design choice to use ensemble bagging to secure modern information systems against malicious network attacks.

General Conclusion

This project successfully developed and validated the *Sentinel Flow* Intrusion Detection System, a behavior-based network security solution utilizing machine learning.By bridging low-level network logic with data science paradigms, the platform addresses the critical vulnerabilities of traditional signature-based firewalls when encountering modern network anomalies.

During development, we proved that separating the fast process of catching network traffic from the heavy process of running the AI models is critical because it completely stops the system from dropping data when the network gets busy. Furthermore, our tests showed that the Random Forest model worked much better than simpler models, reaching an accuracy score over 0.99 when detecting common attacks like SYN Floods and Portscans. Finally, we built a React web dashboard that takes complex network data and turns it into clear, instant visual alerts.

In the future, we want to upgrade it from just detecting threats to actually blocking them. We can do this by using tools like Redis or Kafka to handle bigger data loads, and by writing scripts that automatically add dangerous IP addresses to a firewall's block list. Overall, Sentinel Flow shows that combining network parsing with machine learning is a fast and reliable way to protect modern computer networks.

4.4.1 Perspectives

While the developed system delivers strong real-time results, we have identified several directions to expand and improve our work in the future:

- **Integration of Advanced Deep Learning:** The current system depends on traditional machine learning algorithms, which face clear limitations against complex and time-based attack patterns. In our future works, we plan to involve advanced deep learning architectures such as Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN), and autoencoders. We will train these models on larger, modern datasets like CSE-CIC-IDS2018 to improve accuracy.

- **Support for Industrial Networks:** The current application only monitors standard IP traffic, leaving industrial field devices like Programmable Logic Controllers (PLCs) and sensors unmonitored. Future development will focus on adding custom Scapy parsers to capture, inspect, and analyze specialized industrial communication protocols.

- **Adversarial Machine Learning Resilience:** We intend to explore adversarial machine learning by generating synthetic malicious traffic samples using techniques like the Fast Gradient Sign Method (FGSM). This approach will allow us to test our system's resilience against evasion tactics, enrich our training data datasets, and train our models to recognize disguised attacks effectively.

Bibliography

[1] Al-Mhiqani, M. N., et al. (2023). "Next-Generation Firewalls (NGFW) in Industrial Control Systems." *IEEE Access*, 11.

[2] Garsva, G., & Danenas, P. (2024). "Endpoint Detection and Response (EDR/XDR) Architectural Evolution." *Computers & Security*.

[3] Rodriguez, J., & Hassan, M. (2022). "Centralized Log Management in SIEM Platforms." *Journal of Network and Systems Management*.

[4] Smith, L. A., & Turner, K. (2025). "Defending Corporate Email Vectors." *International Journal of Information Security*.

[5] Zhao, H., et al. (2023). "Hardening Switch Port Communications at Layer 2." *Network Security Review*.

[6] Scarfone, K., & Mell, P. (2007). *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST Special Publication 800-94.

[7] Bishop, M. (2019). *Computer Security: Art and Science* (2nd ed.). Addison-Wesley Professional.

[8] Sommer, R., & Paxson, V. (2010). "Outside the closed world: On using machine learning for network intrusion detection." *Proceedings of the IEEE Symposium on Security and Privacy*, 305–316.

[9] Stallings, W. (2020). *Effective Cybersecurity: A Guide to Using Best Practices and Standards*. Addison-Wesley Professional.

[10] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[11] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). "Toward a reliable dataset for intrusion detection and malware analysis." *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*, 177-184.

[12] Microsoft Security Research. (2025). *The Role of Managed Endpoints and EDR Frameworks in Corporate Security*. Microsoft Technical Library.

[13] IBM Security. (2025). *Cost of a Data Breach Report and Reactive SIEM Latency Metrics*. IBM Whitepapers.

[14] National Institute of Standards and Technology. (2024). *The NIST Cybersecurity Framework 2.0*. NIST Core Publications.

[15] Cisco Systems, Inc. (2024). *Configuring Local SPAN and Switched Port Analyzer Features on Enterprise Switches*. Cisco Configuration Guides.

[16] Microsoft Security Documentation. (2025). *Role-Based Access Control and Token Authentication Architecture in Modern Frameworks*. Microsoft Technical Press.

[17] Cisco Systems, Inc. (2024). *Configuring Switched Port Analyzer (SPAN) Port Mirroring Features*. Cisco Press.

[18] Kumar, A., & Singla, R. (2024). "Machine learning approaches for binary classification in cyber threat detection systems." *Journal of Information Security and Applications*, 78, 103642.