Tribhuvan University

Faculty of Computer Science and Information Technology

A Project Report on

“Network Intrusion Detection System”

In partial fulfillment of the requirement for the degree of Bachelor of Computer Science and Information Technology

(BSc.CSIT)

Submitted to:

Department of Computer Science and Information Technology

Kathmandu College of Technology

Submitted by:

Ganesh Chapagain (26887/077)

Ishor Shrestha (26889/077)

Siyon Babu Rai (26904/077)

Under the supervision of

Trailokya Ojha

January 2025

Tribhuvan University

Faculty of Computer Science and Information Technology

Kathmandu College of Technology

STUDENT'S DECLARATION

We hereby declare that we are only the author of this work and that no other sources other than the sources list in the references have been used in this work.

………………………… ……………………………

Ganesh Chapagain **Ishor Shrestha**

26887/077 26889/077

…………………………

Siyon Babu Rai

26904/077

Tribhuvan University

Faculty of Computer Science and Information Technology

Kathmandu College of Technology

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project report prepared under my supervision by **Ganesh Chapagain, Ishor Shrestha, Siyon Babu Rai** entitled **"Network Intrusion Detection System"** in partial fulfillment of the requirements for the degree of B.Sc. CSIT be processed for the evaluation.

………………………

Mr. Trailokya Ojha

Kathmandu College of Technology

(Supervisor)

Tribhuvan University

Faculty of Computer Science and Information Technology

Kathmandu College of Technology

LETTER OF APPROVAL

This is to certify that this project prepared by Ganesh Chapagain, Ishor Shrestha, Siyon Babu Rai entitled **"Network Intrusion Detection System"** in partial fulfillment of the requirement for the degree of bachelor's in computer science and information technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

Evaluation Committee

…………………… **Head of Department** Mr. Santosh Rijal Kathmandu College of Technology	……………….. **Supervisor** Mr. Trailokya Ojha Kathmandu College of Technology
…………………. **External Examiner**	………………. **Internal Examiner**

ACKNOWLEDGMENT

The success and results of this project require a lot of advice and help from many people, and we feel fortunate enough to have been able to receive all of this during the completion of our final year project on Network Intrusion Detection System. Whatever we have achieved is only because of this guidance and help and we will never forget to thank them. We could not have completed this project without the help of our college “Kathmandu College of Technology” who provided us with academic support and other activities related to information and communication technology as well as extracurricular activities where we could participate. They gave us a family environment. We take this opportunity to express our gratitude and deep respect to our internal supervisor “Mr. Trailokya Ojha” sir for his constant guidance, monitoring, and encouragement throughout this thread. We are extremely grateful for the assistance and support from our friends, teacher, and others during this project phase. This experience has taught us valuable lessons and provided unforgettable learning wisdom.

Ganesh Chapagain (26887/077)

Ishor Shrestha (26889/077)

Siyon Babu Rai (26904/077)

ABSTRACT

The internet has grown rapidly and become a crucial part of modern life, enabling communication, information exchange, and everyday activities. However, it also serves as a gateway for various cyber threats, including malware, phishing attacks, ransomware, and unauthorized network intrusions that compromise data security. Ensuring secure network communication and real-time Intrusion Detection is a significant challenge in cybersecurity. Traditional intrusion detection systems often face limitations due to the lack of reliable datasets for testing and validation, affecting their accuracy and performance. Additionally, a robust system must operate in real time with minimal false alarms to be effective. A model is trained in a simulated testbed environment using generated network attacks, allowing it to recognize real-world attack patterns. It effectively identifies various threats, including Port Scanning, Denial-of-Service (DoS), and Brute Force attacks. This approach enhances threat detection while maintaining a low false alarm rate. By incorporating real-time intrusion detection mechanisms, users are safeguarded against malicious cyber activities, strengthening overall network security. Implementing machine learning for intrusion detection offers a reliable and efficient method for identifying and preventing cyber threats, contributing to a safer digital environment.

Keywords: Internet, cybersecurity, machine learning, intrusion detection, network threats, real-time detection, Port Scanning, Denial-of-Service, Brute Force, false alarm rate, network security

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviations	Full Form
ACK	Acknowledgement Flag
API	Application Programming Interface
CICIDS	Canadian Institute for Cyber Security Network Intrusion System
CSV	Comma Separated Value
DoS	Denial of Service
DVWA	Damn Vulnerable Web Applications
FTP	File Transfer protocol
IDS	Intrusion Detection System
IAT	Inter Arrival Time
IP	Internet protocol
IOT	Internet of Things
JSON	JavaScript Object Notation
NIDS	Network Intrusion Detection System
NIC	Network Interface Card
RST	Reset Flag
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URG	Urgent Flag
VM	Virtual Machine

CHAPTER 1: INTRODUCTION

1.1 Introduction

Network Intrusion Detection System (NIDS) is a security tool designed to monitor and analyze network traffic in real time to detect malicious activity by identifying suspicious patterns, events, or known signs of resistance. It sits at key network nodes as routers or gateways to provide traffic management and detect threats such as Denial of Service (DoS) attacks, malware, and unauthorized scans. NIDS analyzes packets and behaviors and sends instant alerts to administrators so they can take quick action to prevent or mitigate breaches. It can be used as a centralized system that monitors traffic across multiple devices and provides connectivity visibility, or as a local system focused on a single host or small subnet. As cyber threats continue to increase, NIDS remains an important part of network security, ensuring sensitive data is protected and network operations are secure.

Machine learning (ML) is a major advancement in network intrusion detection systems (NIDS) by detecting complex and novel attacks that rule-based methods often miss. ML-based NIDS use supervised, unsupervised, or semi-supervised learning to identify anomalies in network connections. Supervised learning methods such as logistic regression, decision trees, and support vector machines (SVM) are trained on a dataset with both benign and suspicious traffic to classify the input data, while unsupervised techniques such as K-Means clustering and Autoencoders can capture previously unlabeled patterns and anomalies. These systems extract and analyze features such as packet size, protocol type, IP address, and traffic to detect attacks such as port scans, brute force attempts, denial of service, denial of service (DoS) attacks, and malware. By continuously learning from new data, ML-based NIDS can adapt to changing threats and provide more accurate and faster threat detection. However, challenges remain, such as the need for good registration data, accounting burden, and the vulnerability of the system to fraud, but research is ongoing to resolve the issue using different methodology. More focus needs to be placed on combining different models and developing approaches to address these issues.

1.2 Problem Statement

The increasing size and complexity of modern networks have made network security more difficult to manage. Traditional methods for detecting malicious activities often fail to meet these challenges. These older approaches struggle to keep up with the constantly changing nature of cyber threats, the high amount of network traffic, and the small changes in attack patterns. As cybercriminals develop more advanced techniques, these outdated methods become less effective over time.

In addition, the lack of real-time monitoring and automated alert systems makes the problem worse, leaving administrators unable to respond quickly to potential threats. Current intrusion detection systems (IDS), especially those that rely on signature-matching, are mainly reactive instead of proactive. They focus on detecting known threats but often fail to identify new or changing attack patterns. This leads to limited actionable insights, making it harder to prevent security breaches effectively.

To solve these issues, this project proposes the development of a Machine Learning-based Network Intrusion Detection System (NIDS). This system will be designed to identify new attack patterns and detect unusual behavior in network traffic. By using techniques such as Logistic Regression, the system aims to improve detection accuracy while reducing false positives. By combining real-time monitoring with offline analysis, the NIDS will provide strong protection by analyzing both live network traffic and past data. Additionally, automated alert systems will allow for quick responses, helping administrators secure their networks more effectively.

1.3 Objectives

The objectives of this project are as follows:

- To build a real time, web-based NIDS with machine learning and automated email alerts for suspicious activity.
- To allow users to upload network traffic and get intrusion predictions.
- To create a dashboard to monitor traffic and intrusion trends.

1.4 Scopes and Limitations

Scopes

- Develop a system capable of real-time analysis of network traffic to identify potential threats.
- Utilize machine learning algorithms to classify network traffic as benign or malicious.
- Provide an interface for users to upload network data and access analysis results.
- Implement a dashboard for monitoring network activities, anomalies, and trends.
- Automate the delivery of email alerts for critical security incidents.

Limitations

- The system is limited to detecting three attack types port scanning, brute force attacks and denial of service (DoS) attacks.
- Real-time traffic analysis for large-scale networks requires substantial computational power.
- The system may struggle to scale with growing network traffic volumes.
- Regular updates and model retraining are necessary to maintain accuracy and security.

1.5 Development Methodology

We have divided our projects into modules and then after developing individual components we will integrate the components. In this project, we will be using agile methodology. When developing this project, different tasks will be assigned to different members and progress. We will conduct weekly meetings and based on the meetings we will create and divide work and then develop reports after completion of work.

Agile Software Development is an iterative and incremental software development approach whereby the highest priority is focused on delivering a working product as quickly as frequently as possible. It provides close collaboration between the development team and the customer to ensure that the product conforms to their needs and expectation. The Agile methodology will be adopted for the NIDS system development in a machine learning-based approach to make the project flexible and allow iteration through the development. The process is divided into small, manageable sprints, each focusing on specific tasks such as dataset preprocessing, model training, frontend and backend integration, and system testing. Every sprint is an opportunity for the continuous improvement of the system components by refining the algorithms, such as Logistic Regression, Random Forest, and SVM, considering evaluation metrics. Though user feedback is not collected, iterative testing and evaluation after every sprint ensure that the system will be able to meet its objectives pertaining to classifying network traffic as normal or anomalous with a high degree of accuracy. This approach will let the system evolve with the requirements but systematically implement them.

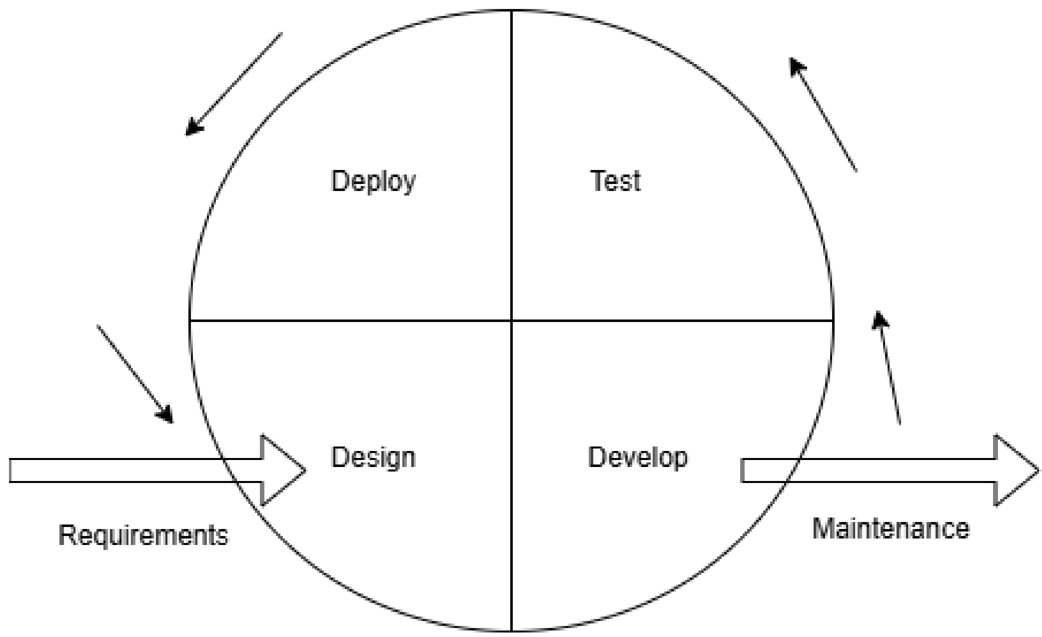


Figure 1. 1: System Development Methodology

1.6 Report Organization

Chapter 1: Introduction

This chapter introduces the problem statement, the goals, or objectives, and provides a brief overview of the project.

Chapter 2: Background Study & Literature Review

This chapter presents the content of previous research and other studies related to our body. This chapter also touches on various analysis possibilities.

Chapter 3: System Analysis

This chapter describes on what our system can do which includes system requirements, feasibility study, and using an object-oriented approach shows the concepts by using class, object, state, sequence, and activity diagrams.

Chapter 4: System Design

This chapter details on how our system implements the design, including the relationships between components and the implementation of algorithms.

Chapter 5: Implementation and Testing

This chapter describes the different applications, tools, testing procedures and result analysis during development.

Chapter 6: Conclusion and Future Recommendation

The final chapter presents the results of our project and discusses potential

improvements and future developments.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

With the increasing sophistication and volume of cyber threats, organizations are facing unprecedented challenges in securing their networks. Traditional security measures are no longer sufficient to detect and respond to evolving attack patterns, making the implementation of Network Intrusion Detection Systems (NIDS) a critical necessity. NIDS are designed to monitor and analyze network traffic to identify malicious activities, providing organizations with real-time threat detection and prevention capabilities. The market for NIDS solutions is driven by factors such as the rising frequency of cyberattacks, the growing adoption of cloud services, and the increasing regulatory requirements for data security. Machine learning (ML) models, such as Logistic Regression, play a crucial role in enhancing the effectiveness of NIDS by enabling them to learn from historical attack patterns and adapt to new threats, offering improved detection accuracy and faster response times. However, implementing an effective NIDS comes with significant challenges, including handling large volumes of network traffic, reducing false positives, and ensuring scalability for different network environments.

Despite the advantages of machine learning (ML) based NIDS, several challenges must be addressed to ensure their effectiveness. One of the key challenges is the presence of imbalanced datasets, where certain types of attacks may be underrepresented, leading to biased detection performance. Additionally, the complexity of network traffic makes it difficult to extract meaningful features that accurately represent malicious activities. Traditional rule-based methods struggle with identifying novel attack patterns, and ML based approaches require continuous updates to adapt to evolving threats. Furthermore, deploying an efficient real-time detection system that balances accuracy and performance without overwhelming system resources is a significant hurdle.

The proposed NIDS addresses these challenges by implementing logistic regression, a simple yet effective ML algorithm, to classify network traffic efficiently. The CICIDS 2018 dataset is used for training, ensuring a comprehensive representation of attack scenarios such as Port Scanning, Brute Force, and Denial of Service (DoS). Feature selection techniques are applied to improve classification accuracy while minimizing computational overhead. The system is implemented as a web-based application using Flask, providing a user-friendly interface for real-time monitoring and response. Additionally, the inclusion of an offline mode allows organizations to upload network traffic data manually for retrospective analysis, enabling deeper insights into historical attack patterns and improving overall security measures. Furthermore, the system incorporates an admin notification feature that alerts administrators when malicious activity crosses a predefined threshold, ensuring timely responses to potential threats.

2.2 Literature Review

R. Saha [1] proposed an adaptive classifier-based intrusion detection system using logistic regression and Euclidean distance, focusing on resource-constrained environments. Their study demonstrated that lightweight classifiers can effectively identify anomalies with minimal computational overhead.

M. University [2] conducted a comparative analysis of machine learning algorithms for detecting port scanning and Distributed Denial-of-Service (DDoS) attacks, concluding that Support Vector Machines (SVM), Decision Trees, and Neural Networks perform well in distinguishing malicious activities from normal traffic. These studies reinforce the effectiveness of machine learning in enhancing NIDS capabilities by offering adaptive and scalable solutions.

One of the major challenges in NIDS is handling class imbalance, where attack traffic represents only a small fraction of the overall network activity. G. Haixiang [3] provided a comprehensive review of methods to address class imbalance in intrusion detection, emphasizing techniques such as oversampling, under sampling, cost-sensitive learning, and ensemble methods. Their findings suggested that hybrid approaches, such as combining the Synthetic Minority Over-sampling Technique (SMOTE) with ensemble classifiers, enhance attack detection rates and reduce false negatives. Addressing class imbalance is crucial to improving the reliability and accuracy of NIDS models, ensuring that minority class attacks do not go undetected.

As cyber threats become more sophisticated, adversarial training has emerged as a key strategy to strengthen NIDS resilience against evasion attacks. Z. Zhong [4] proposed a hybrid adversarial training approach integrating Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) to improve model robustness. Their study demonstrated that training models with adversarial samples significantly enhances their ability to withstand sophisticated attacks designed to bypass traditional detection mechanisms. The concept of adversarial training is particularly relevant in modern cybersecurity, as attackers continuously evolve their techniques to exploit weaknesses in machine learning-based security systems. Deploying IDS as web-based applications has gained traction due to the accessibility and usability it offers. Flask, a light-weight Python web framework, has been widely used for such implementations.

M. K. Baklizi [5] explored multiple machine learning techniques for detecting web-based attacks, highlighting the role of supervised learning models in improving detection accuracy. Scapy has been widely used for packet-level analysis due to its ability to capture, manipulate, and extract network traffic features in real time.

R. R. S, R. R. Yadav [6] states real-time traffic analysis and anomaly detection are crucial for modern NIDS, as immediate response mechanisms significantly reduce the risk of large-scale security breaches. SCAPY, a powerful packet manipulation tool, has been widely used in cybersecurity research for real-time network traffic analysis.

Port scanning and DDoS detection remain crucial areas of research in intrusion detection. Port scanning is a common reconnaissance technique used by attackers to identify open ports and vulnerabilities in a network. For Port Scanning, M. Bhuyan [7] conducted a comprehensive survey on port scanning detection methodologies, evaluating statistical, rule-based, and machine learning approaches. Their study emphasized the need for real-time detection mechanisms to mitigate threats before they escalate into full-scale cyberattacks.

Another critical aspect of NIDS is detecting brute-force attacks, which target authentication systems through repeated login attempts. For Brute Force Attacks, R. A. Grimes [8] investigated brute-force attack methodologies and prevention strategies, emphasizing the role of multi factor authentication (MFA) and rate-limiting techniques in mitigating unauthorized access attempts. Their study suggested that incorporating behavioral analytics and anomaly detection further enhances NIDS capabilities in detecting brute-force attempts in real-time. Given the increasing frequency of authentication-based intrusions, integrating advanced security mechanisms into NIDS is essential.

Despite significant advancements in NIDS research, several challenges remain in practical implementation. K. B. Adedeji [9] highlighted issues such as high false positive rates, computational complexity, and the difficulty of deploying NIDS in large-scale environments. Additionally, the increasing use of encrypted network traffic presents a major challenge, as traditional intrusion detection techniques struggle to analyze encrypted payloads. Addressing these challenges requires the development of lightweight, scalable, and privacy-preserving NIDS solutions that balance security and efficiency.

With the rise of deep learning in intrusion detection, researchers have demonstrated that neural networks offer superior detection accuracy compared to traditional machine learning models. To maintain real time performance, R. Tahri [10] explored the use of deep learning algorithms, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), in NIDS. Their findings showed that deep learning models exhibit higher performance in detecting complex attack patterns and outperform classical classifiers in intrusion detection tasks. Additionally, the use of autoencoders for anomaly detection has been explored as an efficient way to detect novel intrusions without relying on labeled datasets. The continuous development of deep learning techniques presents promising advancements in intrusion detection, offering improved adaptability to evolving cyber threats.

CHAPTER 3: SYSTEM ANALYSIS

3.1 System Analysis

3.1.1 Requirement Analysis

i. Functional Requirements:

The functional requirements for the proposed system are:

- The system should allow real-time network traffic capture.
- The system should support CSV file upload containing network traffic data and apply a trained machine learning model to classify traffic as either normal or an attack.
- The system should provide a secure admin login and registration feature to manage access to the application.
- The system should support the detection of specific attack categories such as port-scanning, brute force, dos, infiltration.
- The system should alert the admin when intrusions are detected.
- The system should allow the admin to view logs of all analyzed traffic and results.

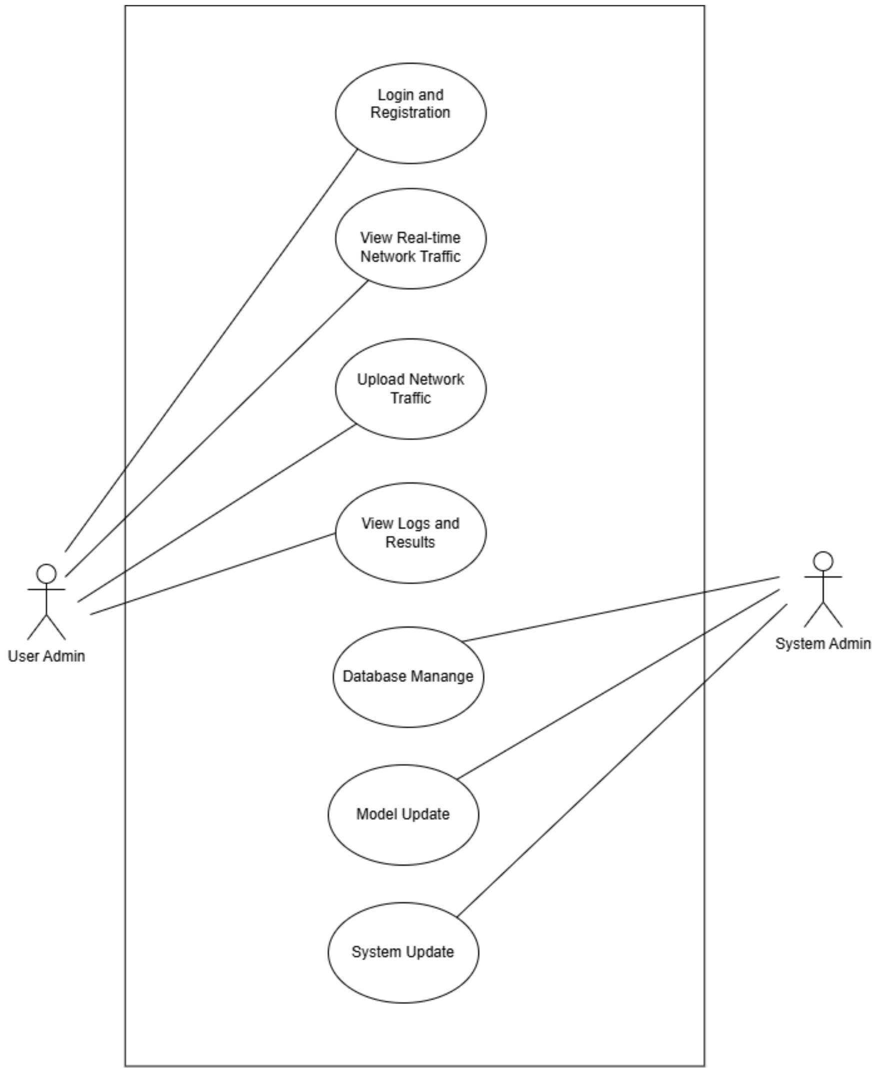


Figure 3. 1: Use Case Diagram

ii. Non-functional Requirements:

- Process real time network traffic efficiently with minimal latency for capture, preprocessing, and classification.
- Provide a user-friendly interface for traffic capture, CSV upload, and admin login, with clear instructions and error handling.
- Ensure the dashboard loads quickly and displays real-time data seamlessly, even with large volumes.
- Implement strong encryption and secure session management to protect admin credentials and prevent unauthorized access.

3.1.2 Feasibility Study:

i. Technical Feasibility

Technical Feasibility refers to the practicality and viability of implementing a proposed solution, considering factors like technology, resources, and expertise. For the development of the Network Intrusion Detection System (NIDS) using Machine Learning, the project utilizes Flask for the backend framework to handle web application functionalities. Tools like Wireshark are employed for packet capturing, enabling real-time data analysis. XAMPP is used for database operations to store and manage datasets effectively. A virtual machine (VM) is set up for attack simulation, ensuring a controlled environment for testing. Machine Learning algorithms are implemented using Scikit-learn, with additional libraries like NumPy and Pandas aiding in model training and data preprocessing. The front-end and backend development is managed in VS Code, ensuring a streamlined and efficient workflow. The required resources, including hardware for simulations and storage, software tools, and training datasets, are readily available, along with the expertise to utilize these technologies effectively. This confirms the project's technical feasibility.

ii. Operational Feasibility

Operational feasibility addresses whether the project can be effectively implemented in its intended environment to meet user requirements. The system demonstrates strong operational feasibility with an intuitive web interface that minimizes user training and provides real-time detection results, making it accessible even to non-technical users. It will be based on real-time monitoring with actionable alerts, making it highly comprehensible and usable by system administrators and security teams for proactive threat detection. With ML models trained on datasets like CICIDS-2018, it ensures high accuracy in detecting known attack patterns and reliable real-time threat identification. The system integrates seamlessly into existing network infrastructures, causing minimal disruption while offering maximum adaptability. Additionally, it is designed for scalability, ensuring consistent performance under increased network traffic or a growing number of monitored endpoints. These attributes confirm the system's effectiveness in meeting all user needs.

iii. Economic Feasibility

The project is economically feasible, leveraging open-source tools like Python, Scapy, and Flask to minimize software costs, with the primary expense being the developer's time and effort manageable within a student project’s scope. Operationally, it utilizes existing hardware, such as standard desktops or laptops, reducing additional investment, while deployment on local or small scale servers keeps hosting expenses low. By enhancing network security and mitigating potential losses from breaches, the system delivers significant value, serving as a cost effective, customizable alternative to expensive commercial NIDS solutions. Future scaling to enterprise-level traffic may require additional investment in cloud infrastructure or high performance computing resources.

iv. Schedule Feasibility

The project spans 121 days, with tasks strategically divided and assigned specific durations within this time frame. To ensure timely delivery of the system, it is crucial to adhere to deadlines and execute tasks efficiently within their designated periods. Proper planning, monitoring, and coordination are essential to maintaining progress and meeting the overall project timeline.

Working Time	5th Aug	13th Aug	5th Sept	20th Sept	5th Oct	28th Nov
Planning						
Design						
Implementation						
Testing						
Maintenance						
Documentation						

Table 3. 1: Gantt Chart

3.1.3 Analysis (Object Oriented Approach)

i. Object Modeling using Class and Object Diagram

Object modeling using class and object diagrams is a structured approach to visually represent the components and interactions within a system. In the context of Network Intrusion Detection System, this involves defining classes to represent entities such as Packet Capture, Features, ML Model and illustrating their relationships and attributes through class diagrams. Object diagrams depict instances of these classes as objects and showcase how they interact with each other during runtime, providing a clear understanding of the system's architecture and behavior.

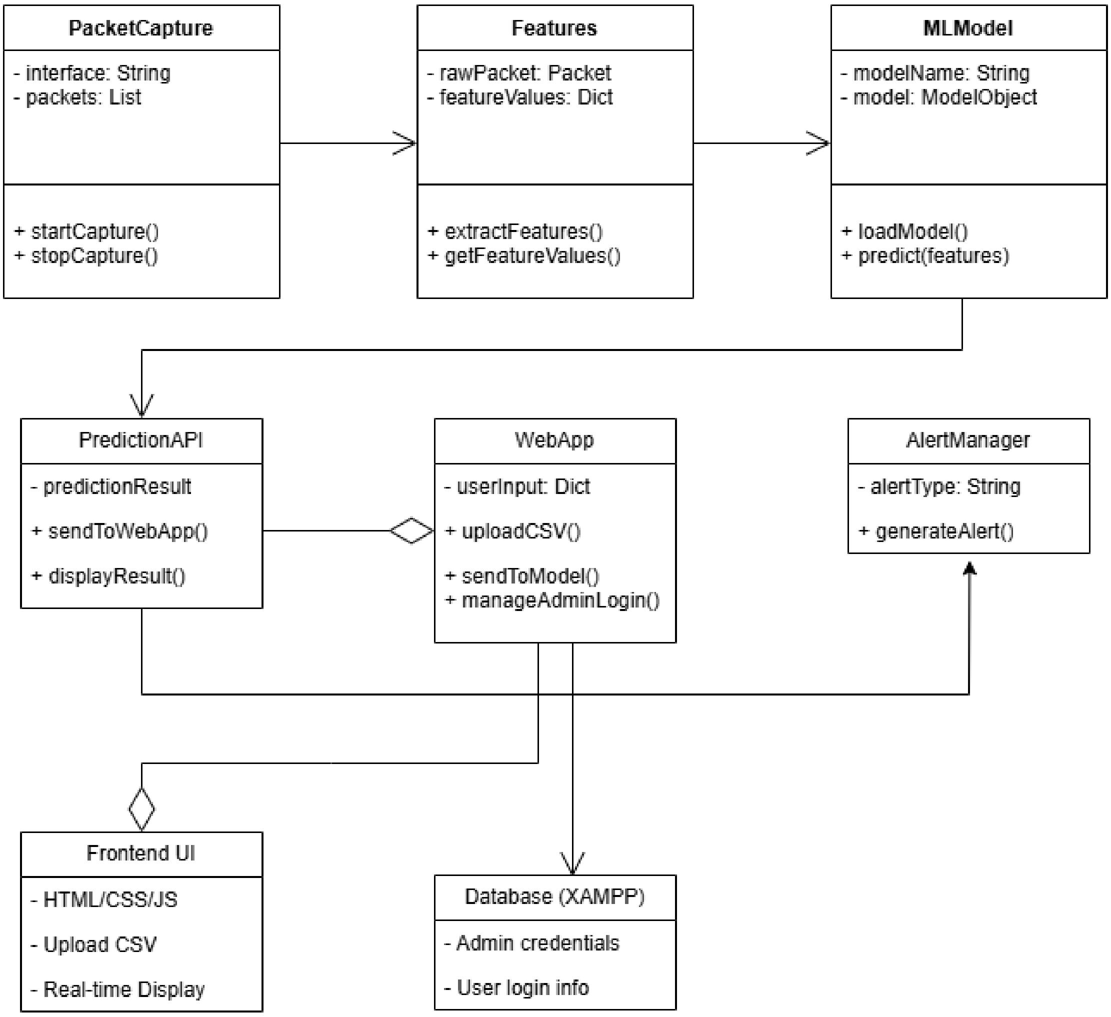


Figure 3. 2: Class and Object Diagram

ii. Dynamic Modeling Using State and Sequence Diagram

Dynamic Modeling with state and sequence diagrams visualizes the behavior and interactions within a system. State diagrams depict different system states and transitions, while sequence diagrams illustrate the flow of messages between system components. For Network Intrusion Detection System, these diagrams help to understand how the system responds to events, aiding in system design and analysis.

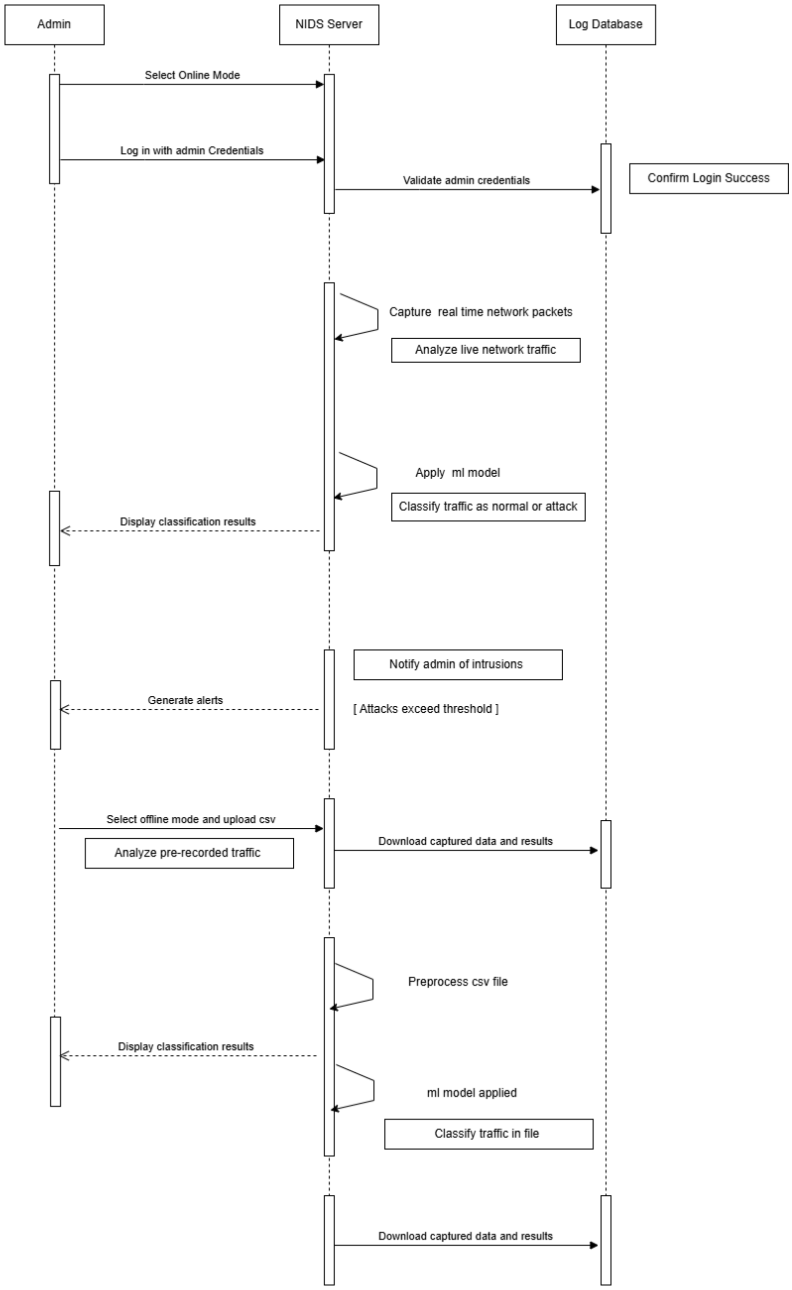


Figure 3. 3: State and Sequence Diagram

iii. Process Modeling using Activity Diagram

Activity Diagrams visually represent the flow of activities or actions within a system, illustrating the sequence of steps and decision points in a process.

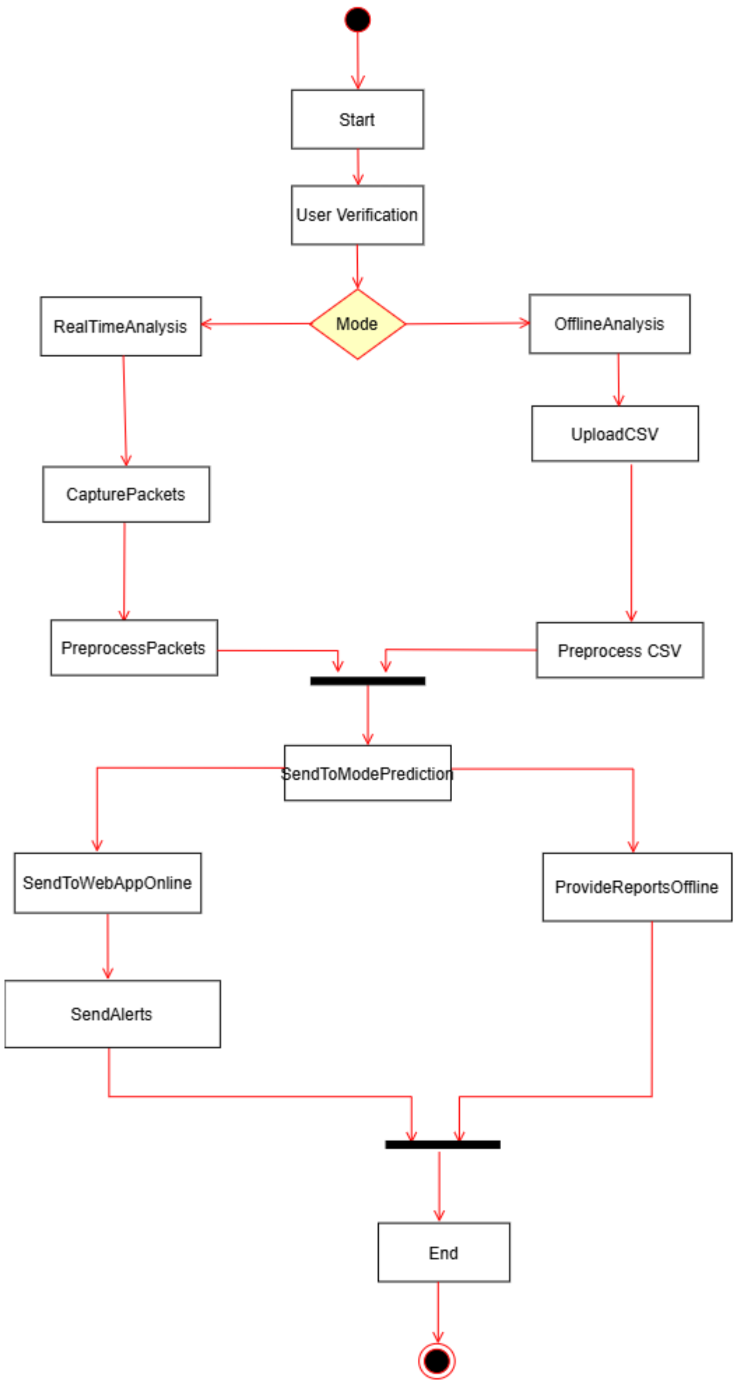


Figure 3. 4: Activity Diagram

CHAPTER 4: SYSTEM DESIGN

4.1 Design

System Flow Diagram

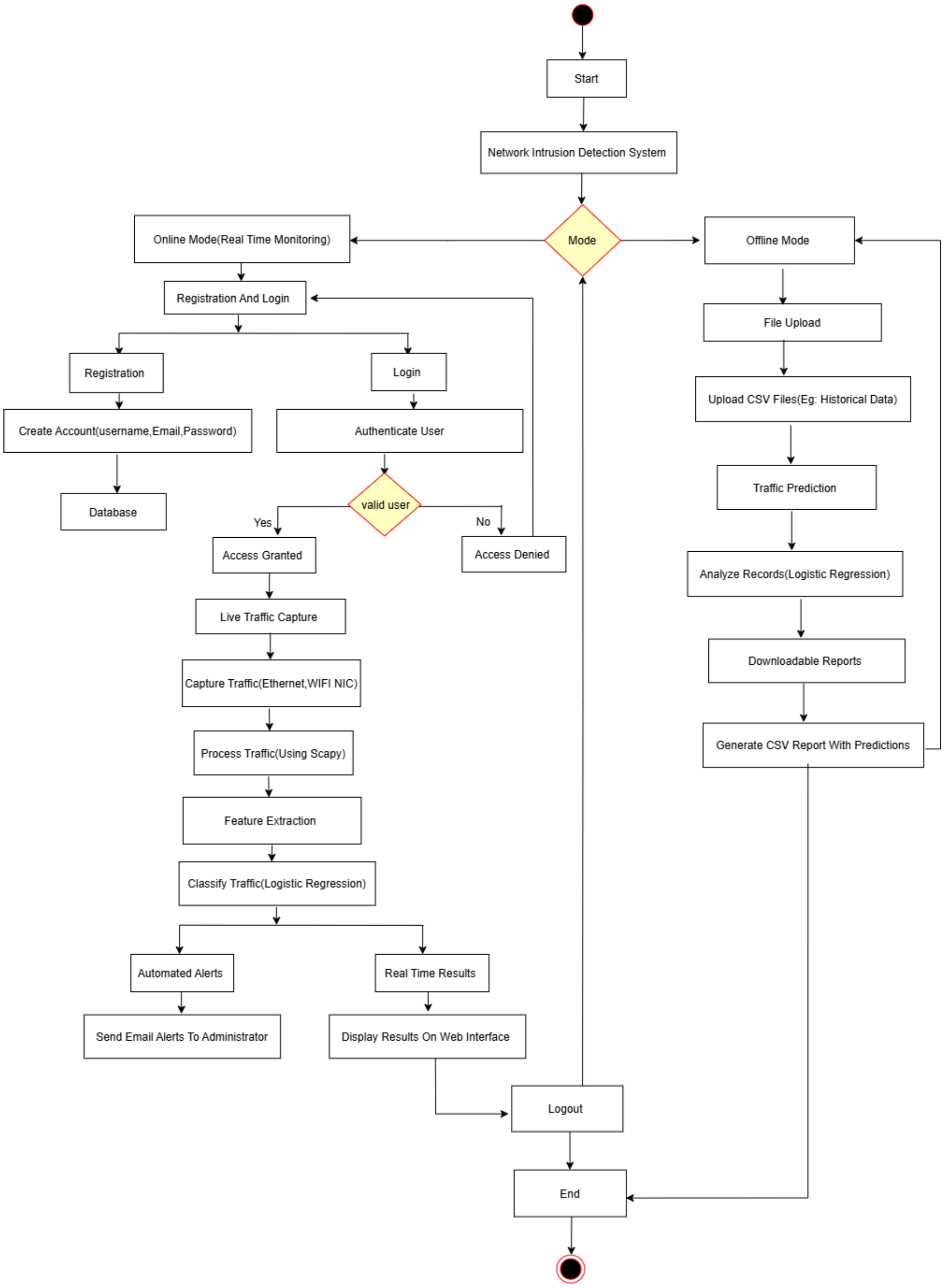


Figure 4. 1: System Flow Diagram

Refinement of Class, Object, State, Sequence and Activity Diagram

i. Refinement of Class and Object Diagram

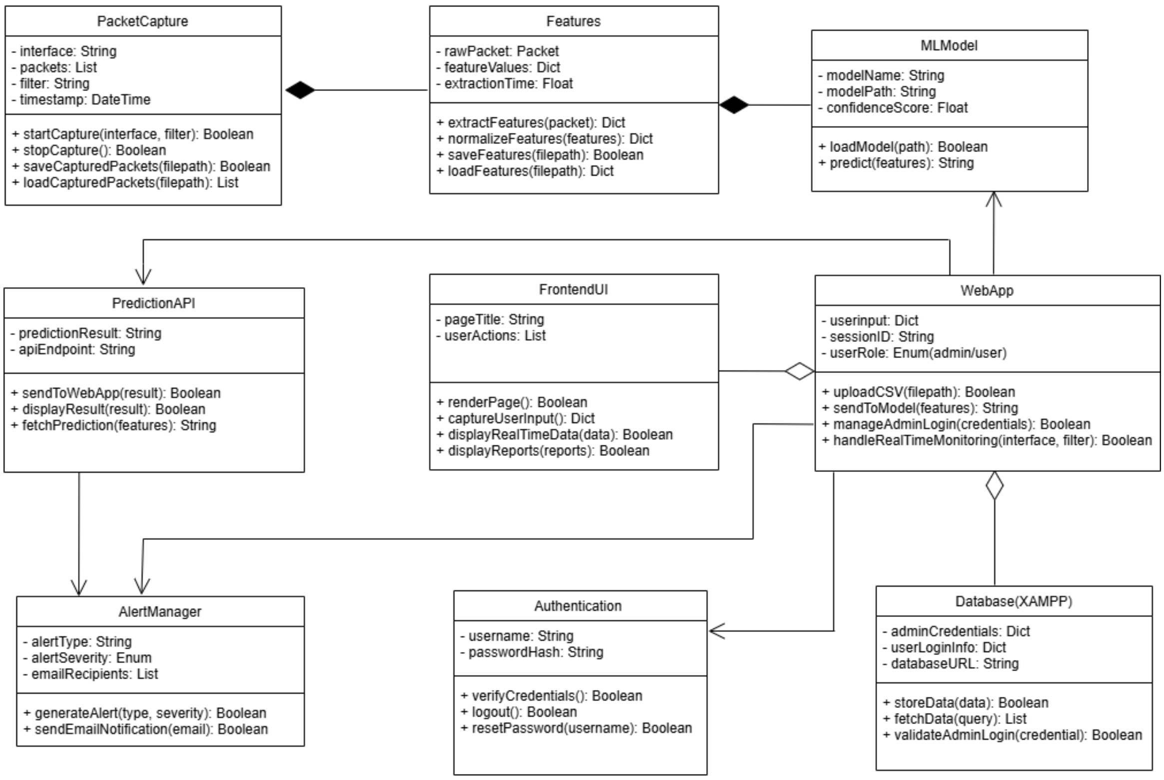


Figure 4. 2: Refined Class and Object Diagram

ii. Refinement of State Diagram

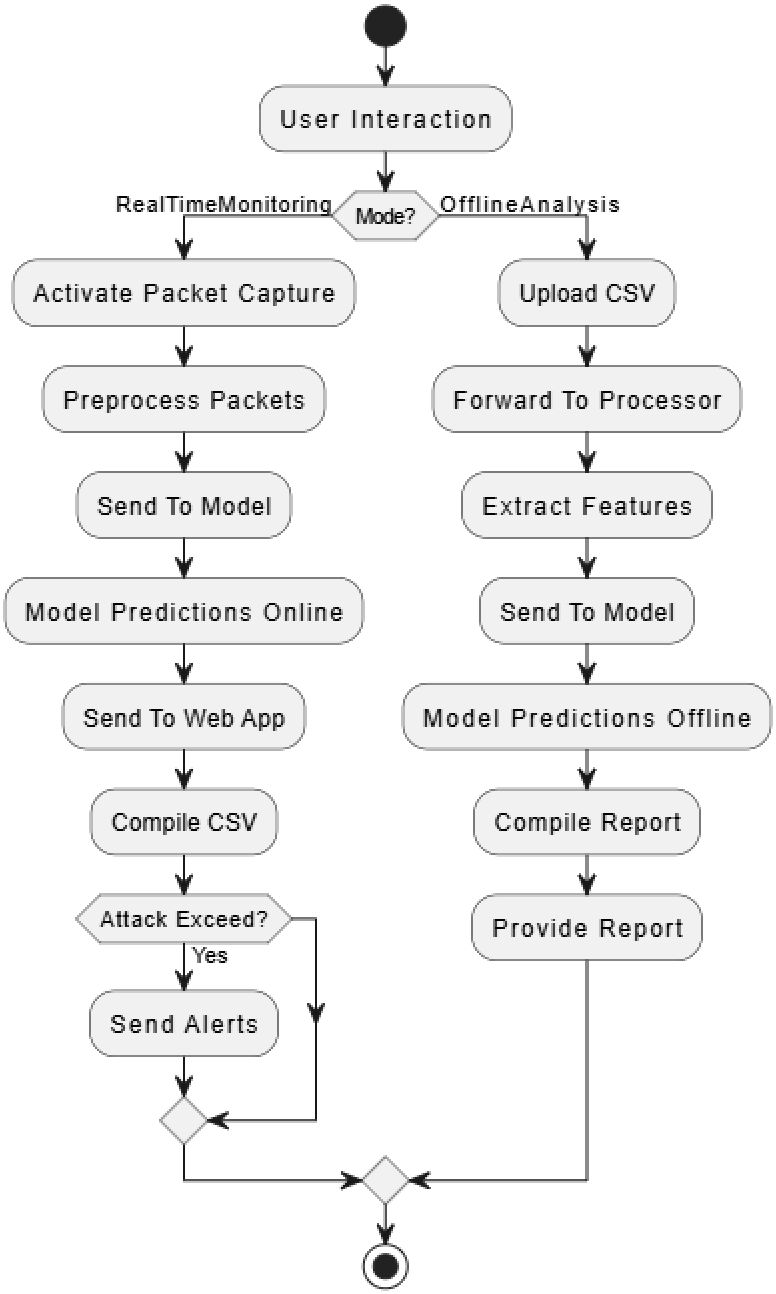


Figure 4. 3: Refined State Diagram

iii. Refinement of Sequence Diagram

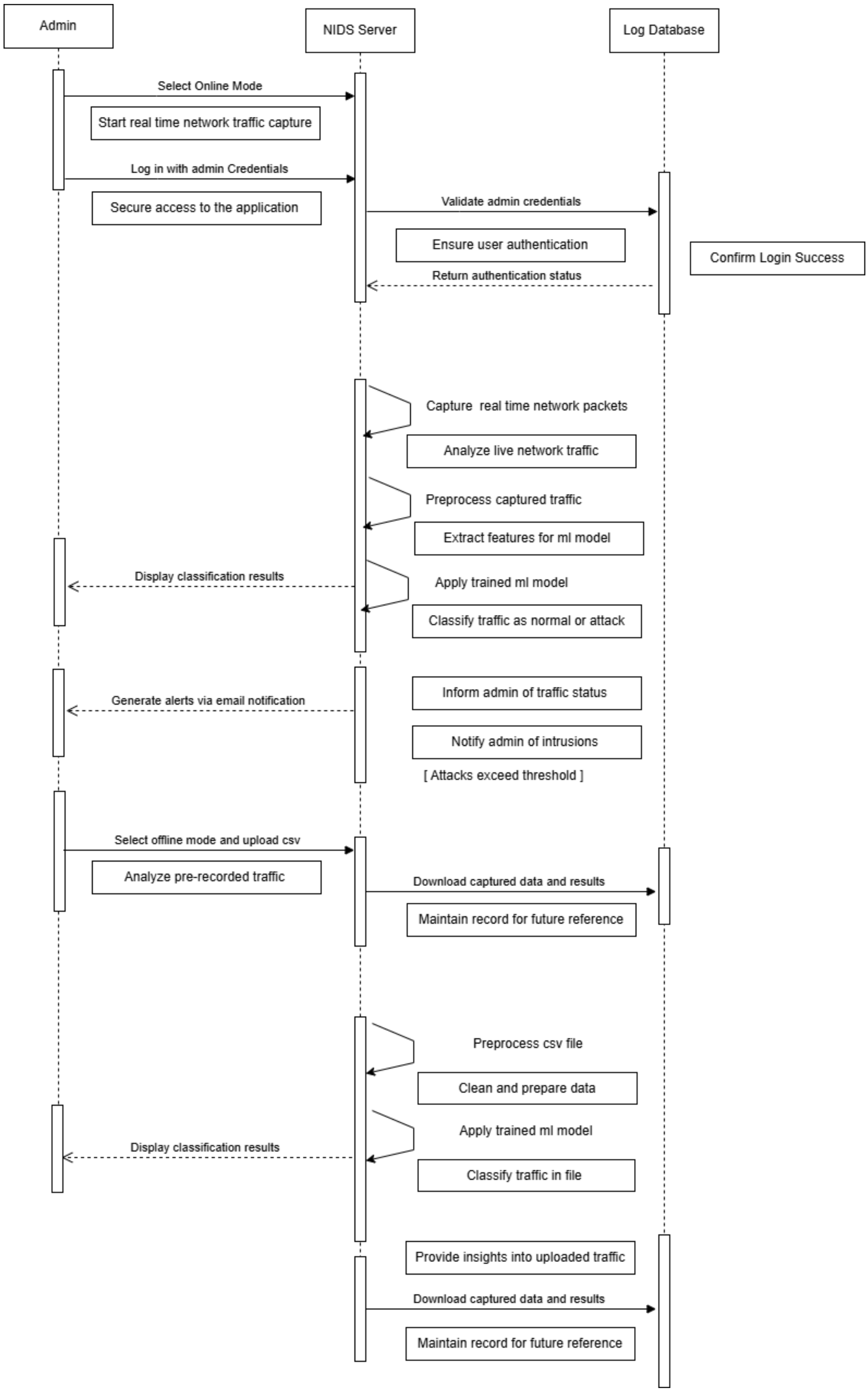


Figure 4. 4: Refined Sequence Diagram

iv. Refinement of Activity Diagram

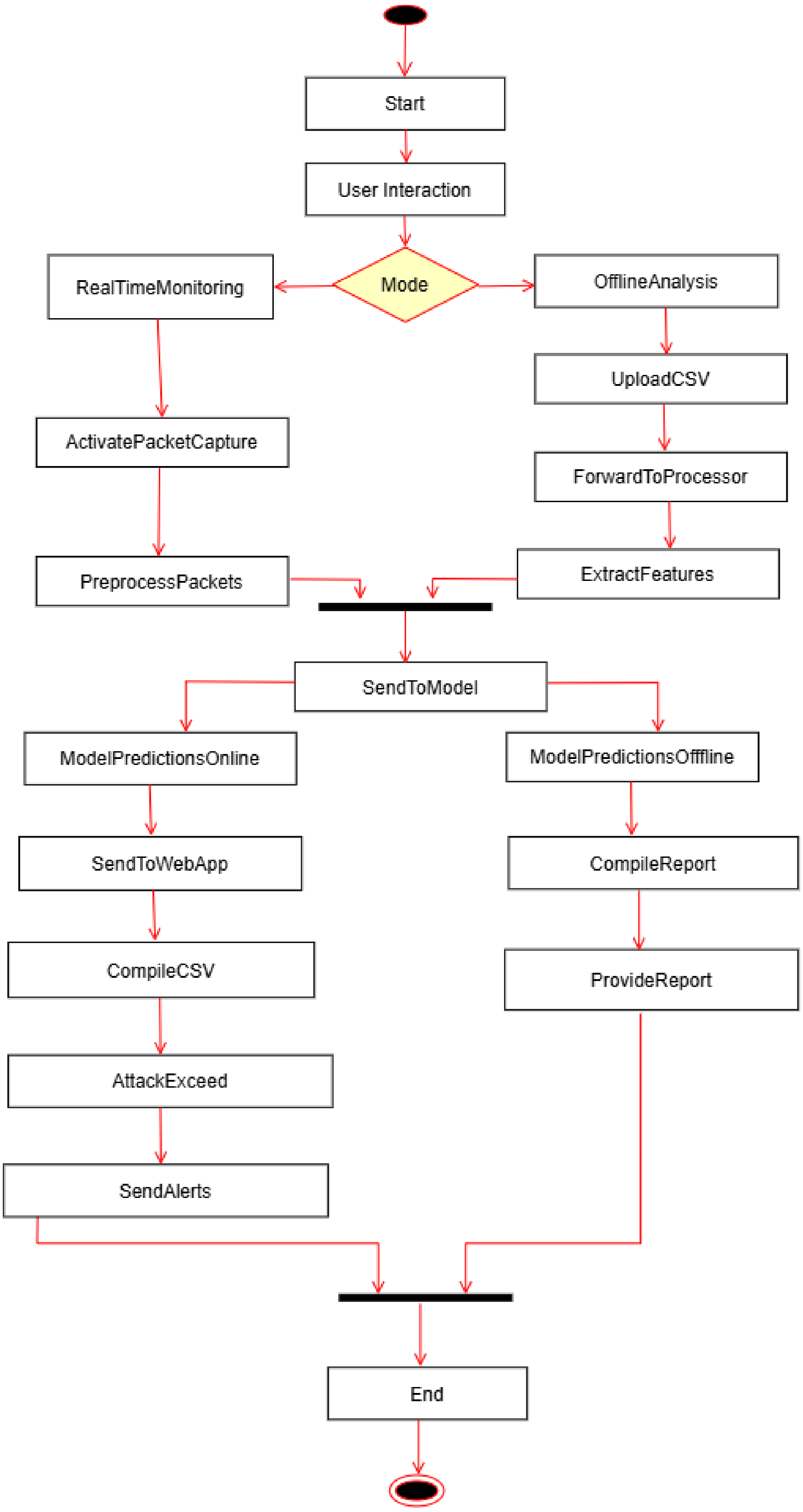


Figure 4. 5: Refined Activity Diagram

v. Component Diagram

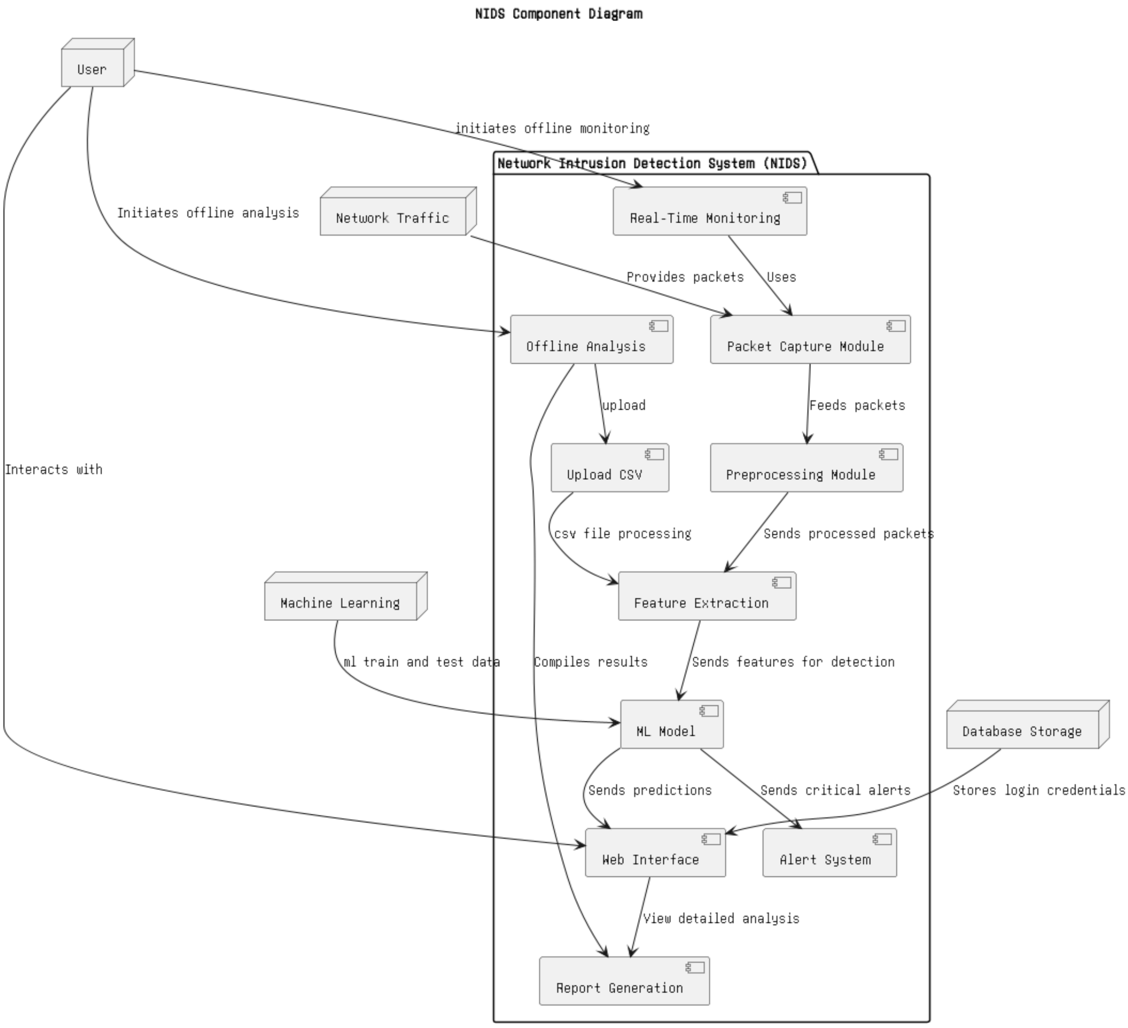


Figure 4. 6: Component Diagram

vi. Deployment Diagram

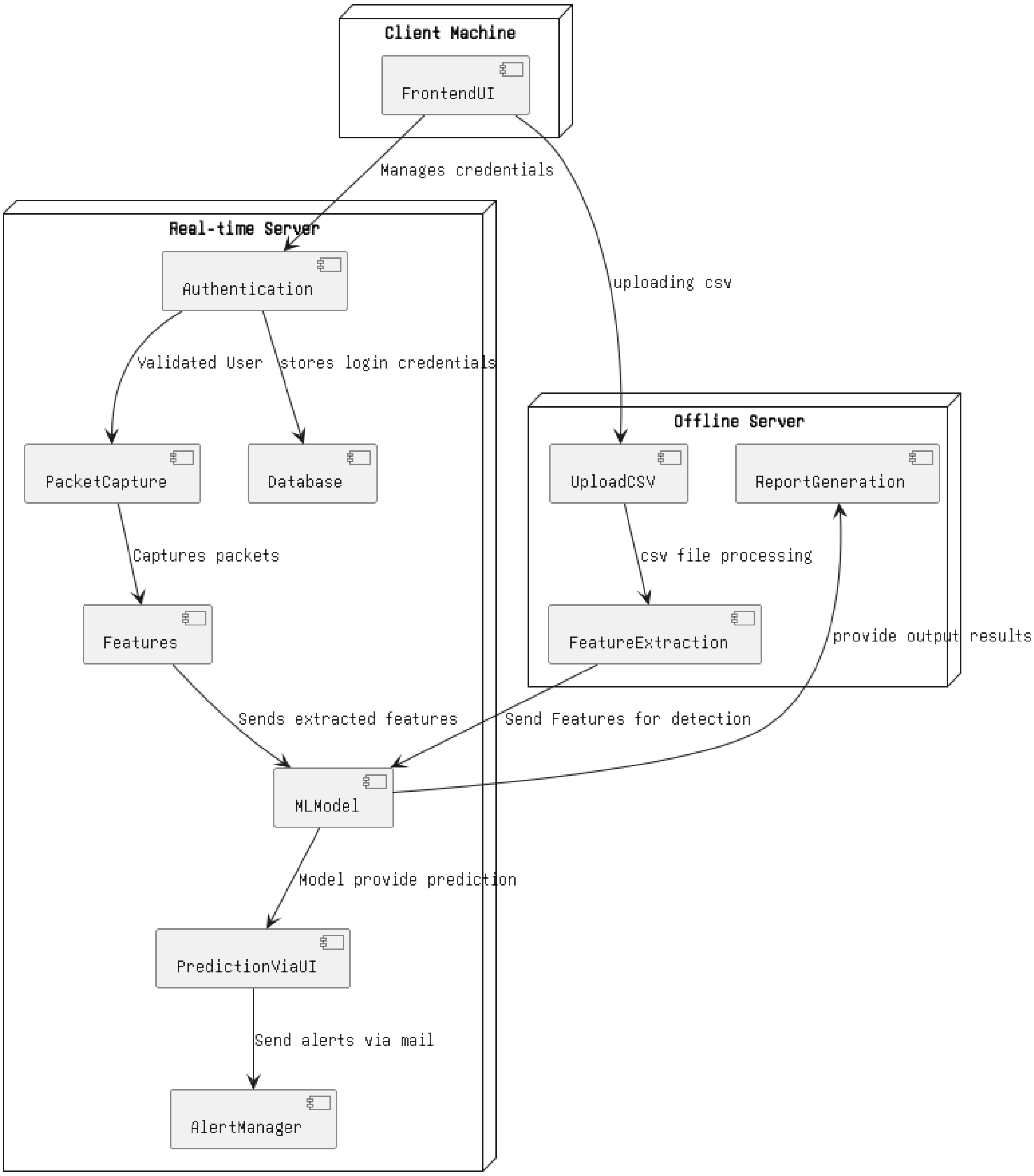


Figure 4. 7: Deployment Diagram

4.2 Algorithm Details

Logistic regression is a popular supervised machine learning algorithm used for binary classification problems, where the goal is to predict one of two possible outcomes. It estimates the probability that a given input belongs to a particular category using the sigmoid function. The output is a probability value between 0 and 1, which is then mapped to a class label (e.g., "yes" or "no," "normal" or "malicious").

Features of Logistic Regression

- **Probabilistic Approach:** Logistic regression calculates the probability of an event occurring, with values ranging between 0 and 1. These probabilities are then used to classify the input data into one of the predefined categories.
- **Sigmoid Function:** The algorithm employs a sigmoid, function to transform the output of the linear model into a probability. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Here, z represents the linear combination of the model's parameters and input features:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

- **Interpretability:** One of the major strengths of logistic regression is its simplicity and interpretability. The model’s coefficients (βi) reflect the influence of each feature on the likelihood of a specific outcome, making it easy to understand and explain.

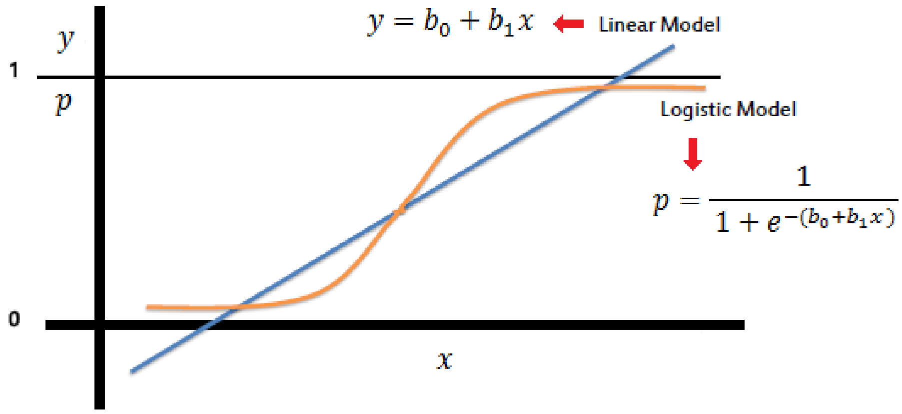


Figure 4. 8: Logistic Regression Model with Sigmoid Function

https://www.saedsayad.com/logistic_regression.htm

Implementation steps of Logistic Regression for Real-Time Network Intrusion Detection

- **Dataset Loading and Preprocessing**

Dataset: Network traffic data is used.

Preprocessing: Irrelevant columns (e.g., identifiers and IP addresses) are removed. Missing values are dropped. Labels are encoded:

Normal traffic → 0

Attack traffic → 1

Features are standardized to ensure uniform scaling.

- **Data Splitting:** The data is divided into training (75%) and testing (25%) subsets using stratified sampling to maintain class balance.
- **Model Training Algorithm:** Logistic Regression with a one-vs-rest strategy and a high iteration limit for convergence.
- **Hyperparameter Tuning:** Grid search with cross validation is performed to optimize the regularization parameter.
- **Saving Artifacts:** The scaler and trained model are saved for deployment in real-time **systems.**
- **Model Evaluation Performance Metrics:** Accuracy, confusion matrix, and classification report (precision, recall, F1-score) are used to evaluate the

model's performance for both Normal and Attack classes. Real Time Prediction Incoming data is preprocessed using the saved scaler. Predictions are made using the saved trained model.

Logistic Regression for Real-Time Network Intrusion Detection

Input:

Network packet features: Flow Duration, Tot Fwd Pkts, Fwd Pkt Len Max, Flow IAT Mean, SYN Flag Count, etc.

Real-time traffic flow data.

Output:

Traffic classification:

Benign (Normal)

Attack (Malicious)

Steps:

- Capture Traffic: Use a packet-capturing library (Scapy) to capture live network traffic.
- Feature Extraction: Extract features such as flow duration, packet length, inter-arrival time, and flag counts.
- Load Model & Scaler: Load the pre-trained Logistic Regression model and scaler.
- Preprocess Features: Normalize the extracted features using the scaler.
- Prediction: Use the Logistic Regression model to classify traffic:
  Class 0: Benign.
  Class 1: Attack.
- Display Results: Show the classification results in a web-based interface.
- Repeat: Continuously process and classify new traffic in real time.

This algorithm uses a pre-trained Logistic Regression model to detect real-time network intrusions. Features from live traffic are captured, preprocessed, and classified into benign or attack categories, with results displayed in a web interface.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Implementation

The Network Intrusion Detection System (NIDS) was implemented using Scapy, a Python library for packet manipulation, to capture and analyze network traffic data, followed by preprocessing techniques like cleaning and normalization to prepare the data for machine learning tasks. Instead of applying feature selection, the model was trained using all available dataset features to fully leverage the data. A Logistic Regression model from Python's Scikit-learn library was used to classify network traffic as normal or malicious, with training, testing, and evaluation conducted in Jupyter Notebook for efficient experimentation and visualization. A user-friendly interface was developed using the Flask web framework, enabling real time monitoring, analysis, and visualization of intrusion detection results. The Agile development methodology was adopted throughout the project to ensure iterative development, regular feedback, and continuous improvement, resulting in a flexible, efficient, and user centric NIDS implementation.

5.1.1 Tools Used

i. **Development Tools/CASE Tools:**

Jupyter Notebook: A web based interactive environment for writing and running Python code, mainly used for data analysis and visualization.

Visual Studio Code (VS Code): A lightweight and powerful code editor with built-in support for Python and extensions for Flask and Scapy development.

Virtualization Software VMware Workstation: Used to set up a Linux virtual machine to simulate network attacks and test the system in a controlled environment.

ii. **Programming Languages:**

Python: Used for machine learning model development, data preprocessing, and backend application logic.

HTML/CSS/JavaScript: Used for frontend development to visualize intrusion detection results.

iii. **Libraries and Frameworks:**

Pandas, NumPy: Essential for data manipulation and preprocessing.

Scikit-learn: Used to implement machine learning models like Logistic Regression, Random Forest, and Autoencoder.

Flask: Provides backend web API functionality and communication between modules.

Scapy: Captures and processes live network traffic for intrusion detection.

iv. **Database Platform:**

MySQL (via XAMPP): Used for storing and managing user credentials for the login functionality of the web application. Although it is not directly related to the network traffic analysis, it ensures secure access to the application through authentication.

v. **Hardware Tools:**

Personal Computer (PC)/Laptop Processor: Intel Core i5/i7 or AMD Ryzen 5/7 (or higher) for efficient processing of machine learning tasks and real-time packet analysis.

RAM: Minimum 8GB (Recommended 16GB or more) to handle large datasets efficiently.

Storage: SSD (Solid State Drive) with at least 256GB to ensure fast data access and processing speeds.

Network Interface Card (NIC): A Gigabit Ethernet Adapter or Wi-Fi Adapter for real-time network traffic monitoring and capturing packets accurately.

Router/Switch: A standard network router or switch used to create a network topology for traffic monitoring and attack simulations.

vi. **Drawing Tools:**

Draw.io: An online diagramming tool used to design network architecture diagrams and process workflows related to intrusion detection.

Canva: Used for creating visually appealing report graphics and illustrations to enhance report presentation.

vii. **Project Management and Documentation Tools:**

Git/GitHub: Version control system used to manage source code changes, track project progress, and collaborate with team members. Enables secure backup and version tracking of code.

Trello: A task management tool used to organize project milestones, tasks, and timelines visually. Helps in tracking the progress of various modules such as data preprocessing, model training, and deployment.

Microsoft Word/Google Docs: Used for writing project documentation, reports, and maintaining records of implementation details.

5.1.2 Implementation Details of Modules

Machine Learning Module

Logistic Regression is a widely used algorithm in supervised machine learning, mainly used for binary classification. This can be done by using the sigmoid function on a linear equation, ensuring that the output lies in a range between 0 and 1 that is, the probability of a data point to belong to one class among two classes. The probabilities are converted to binary outputs according to a threshold value for example, 0 for normal and 1 for malicious traffic. The project scope includes preprocessing labeled network traffic data, training the model, hyperparameter tuning, performance evaluation, and deploying the model in real time. This model is used to identify potential intrusions, enhance network security, and reduce the risks of cybersecurity.

Data Collection

The dataset contains network traffic data enriched with flow metrics, packet sizes, and binary labels indicating normal or malicious traffic. The features in the dataset include flow duration, packet sizes, packet rates, flags, segment sizes, and inter-arrival times, which are essential for binary classification tasks in network intrusion detection. The dataset includes 33,088 unique data entries, each describing various aspects of network flows and attack patterns.

fl_dur: The duration of the flow, indicating the total time the network flow lasts.

tot_fw_pk: The total number of packets transmitted in the forward direction.

tot_bw_pk: The total number of packets transmitted in the backward direction.

tot_l_fw_pkt: The total size of packets sent in the forward direction.

fw_pkt_l_max: The maximum packet size observed in the forward direction.

fw_pkt_l_min: The minimum packet size observed in the forward direction.

fw_pkt_l_avg: The average size of packets sent in the forward direction.

fw_pkt_l_std: The standard deviation of packet sizes in the forward direction.

Bw_pkt_l_max: The maximum packet size observed in the backward direction.

Bw_pkt_l_min: The minimum packet size observed in the backward direction.

Bw_pkt_l_avg: The average packet size in the backward direction.

Bw_pkt_l_std: The standard deviation of packet sizes in the backward direction.

fl_byt_s: The flow byte rate, indicating the number of bytes transferred per second.

fl_pkt_s: The flow packet rate, indicating the number of packets transferred per second.

fl_iat_avg: The average inter-arrival time between two flows.

fl_iat_std: The standard deviation of the time between two flows.

fl_iat_max: The maximum inter-arrival time between two flows.

fl_iat_min: The minimum inter-arrival time between two flows.

fw_iat_tot: The total time between two packets sent in the forward direction.

fw_iat_avg: The average time between two packets sent in the forward direction.

fw_iat_std: The standard deviation of time between two packets sent in the forward direction.

fw_iat_max: The maximum time between two packets sent in the forward direction.

fw_iat_min: The minimum time between two packets sent in the forward direction.

bw_iat_tot: The total time between two packets sent in the backward direction.

bw_iat_avg: The average time between two packets sent in the backward direction.

bw_iat_std: The standard deviation of time between two packets sent in the backward direction.

bw_iat_max: The maximum time between two packets sent in the backward direction.

bw_iat_min: The minimum time between two packets sent in the backward direction.

fw_psh_flag: The number of times the PSH flag is set in packets traveling in the forward direction.

bw_psh_flag: The number of times the PSH flag is set in packets traveling in the backward direction.

fw_urg_flag: The number of times the URG flag is set in packets traveling in the forward direction.

bw_urg_flag: The number of times the URG flag is set in packets traveling in the backward direction.

fw_hdr_len: The total number of bytes used for headers in the forward direction.

bw_hdr_len: The total number of bytes used for headers in the backward direction.

fw_pkt_s: The number of packets transmitted per second in the forward direction.

bw_pkt_s: The number of packets transmitted per second in the backward direction.

pkt_len_min: The minimum length of a flow.

pkt_len_max: The maximum length of a flow.

pkt_len_avg: The average length of a flow.

pkt_len_std: The standard deviation of the length of a flow.

pkt_len_va: The minimum inter-arrival time of packets.

fin_cnt: The number of packets with the FIN flag set.

syn_cnt: The number of packets with the SYN flag set.

rst_cnt: The number of packets with the RST flag set.

pst_cnt: The number of packets with the PUSH flag set.

ack_cnt: The number of packets with the ACK flag set.

urg_cnt: The number of packets with the URG flag set.

cwe_cnt: The number of packets with the CWE flag set.

ece_cnt: The number of packets with the ECE flag set.

down_up_ratio: The ratio of download to upload traffic.

pkt_size_avg: The average size of packets in the flow.

fw_seg_avg: The average size of segments observed in the forward direction.

bw_seg_avg: The average size of segments observed in the backward direction.

fw_byt_blk_avg: The average number of bytes in the bulk rate in the forward direction.

fw_pkt_blk_avg: The average number of packets in the bulk rate in the forward direction.

fw_blk_rate_avg: The average bulk rate in the forward direction.

bw_byt_blk_avg: The average number of bytes in the bulk rate in the backward direction.

bw_pkt_blk_avg: The average number of packets in the bulk rate in the backward direction.

bw_blk_rate_avg: The average bulk rate in the backward direction.

subfl_fw_pk: The average number of packets in a sub-flow in the forward direction.

subfl_fw_byt: The average number of bytes in a sub-flow in the forward direction.

subfl_bw_pkt: The average number of packets in a sub-flow in the backward direction.

subfl_bw_byt: The average number of bytes in a sub-flow in the backward direction.

fw_win_byt: The number of bytes sent in the initial window in the forward direction.

bw_win_byt: The number of bytes sent in the initial window in the backward direction.

Fw_act_pkt: The number of packets with at least one byte of TCP data in the forward direction.

fw_seg_min: The minimum segment size observed in the forward direction.

atv_avg: The average time a flow was active before becoming idle.

atv_std: The standard deviation of the time a flow was active before becoming idle.

atv_max: The maximum time a flow was active before becoming idle.

atv_min: The minimum time a flow was active before becoming idle.

idl_avg: The average time a flow was idle before becoming active.

idl_std: The standard deviation of the time a flow was idle before becoming active.

idl_max: The maximum time a flow was idle before becoming active.

idl_min: The minimum time a flow was idle before becoming active.

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts
1	10.3.141.93-17.57.12.11-53692-443-6	10.3.141.93	53692	17.57.12.11	443	6	21/05/2020 19:12:01	2923744.0	13	9
2	17.57.12.11-10.3.141.93-443-53692-6	17.57.12.11	443	10.3.141.93	53692	6	21/05/2020 19:12:04	82408.0	2	1
3	10.3.141.167-13.107.4.52-64558-80-6	10.3.141.167	64558	13.107.4.52	80	6	21/05/2020 19:12:08	12726.0	3	4
4	10.3.141.167-217.116.4.152-64557-443-6	10.3.141.167	64557	217.116.4.152	443	6	21/05/2020 19:12:03	8429639.0	63	149
5	10.3.141.93-17.253.109.203-54737-80-6	10.3.141.93	54737	17.253.109.203	80	6	21/05/2020 19:12:13	67900.0	3	4
6	17.253.109.203-10.3.141.93-80-54737-6	17.253.109.203	80	10.3.141.93	54737	6	21/05/2020 19:12:13	30658.0	1	3
7	10.3.141.93-17.57.146.52-54736-5223-6	10.3.141.93	54736	17.57.146.52	5223	6	21/05/2020 19:11:56	60680144.0	24	16
8	17.57.146.52-10.3.141.93-5223-54736-6	17.57.146.52	5223	10.3.141.93	54736	6	21/05/2020 19:12:56	40.0	2	0
9	172.217.17.14-10.3.141.203-80-33998-6	172.217.17.14	80	10.3.141.203	33998	6	21/05/2020 19:13:06	217329.0	2	0
10	10.3.141.98-172.217.19.138-37654-443-6	10.3.141.98	37654	172.217.19.138	443	6	21/05/2020 19:12:30	52989409.0	4	1
11	172.217.19.138-10.3.141.98-443-37654-6	172.217.19.138	443	10.3.141.98	37654	6	21/05/2020 19:13:23	49.0	2	0
12	10.3.141.98-216.58.211.46-48686-443-6	10.3.141.98	48686	216.58.211.46	443	6	21/05/2020 19:12:51	52988661.0	4	1
13	216.58.211.46-10.3.141.98-443-48686-6	216.58.211.46	443	10.3.141.98	48686	6	21/05/2020 19:13:44	44.0	2	0
14	10.3.141.1-10.3.141.98-22-51172-6	10.3.141.1	22	10.3.141.98	51172	6	21/05/2020 19:11:56	119622400.0	238	238
15	10.3.141.167-192.168.1.38-64540-8009-6	10.3.141.167	64540	192.168.1.38	8009	6	21/05/2020 19:11:56	116886472.0	51	27
16	8.6.0.1-8.0.6.4-0-0-0	8.6.0.1	0	8.0.6.4	0	0	21/05/2020 19:11:56	114176776.0	25	0
17	172.217.168.174-10.3.141.98-443-33196-6	172.217.168.174	443	10.3.141.98	33196	6	21/05/2020 19:11:58	113858194.0	46	41
18	10.3.141.93-224.0.0.251-5353-5353-17	10.3.141.93	5353	224.0.0.251	5353	17	21/05/2020 19:11:59	75748568.0	6	0
19	10.3.141.167-217.116.4.152-64559-443-6	10.3.141.167	64559	217.116.4.152	443	6	21/05/2020 19:12:14	91301774.0	12	9
20	10.3.141.167-10.3.141.255-137-137-17	10.3.141.167	137	10.3.141.255	137	17	21/05/2020 19:12:13	10521867.0	27	0
21	10.3.141.167-74.125.140.188-64560-5228-6	10.3.141.167	64560	74.125.140.188	5228	6	21/05/2020 19:12:14	90501300.0	10	11
22	216.58.211.42-10.3.141.203-443-43090-6	216.58.211.42	443	10.3.141.203	43090	6	21/05/2020 19:14:43	49.0	2	0
23	216.58.211.42-10.3.141.203-443-43090-6	216.58.211.42	443	10.3.141.203	43090	6	21/05/2020 19:14:43	218211.0	2	0

Figure 5. 1: Dataset for Network Intrusion Detection System

Data Preprocessing

a. Remove Unnecessary Columns

```
# Drop unnecessary columns
df = df.drop(columns=['Flow ID', 'Src IP', 'Src Port', 'Dst IP', 'Dst Port', 'Protocol', 'Timestamp'], axis=1)
```

Irrelevant features, such as IP addresses and timestamps, are dropped to focus on meaningful attributes for classification.

b. Handle Missing Values

```
# Drop NaN values
df = df.dropna(axis=0)
```

Rows with missing values are removed to ensure data integrity.

c. Visualize Data Distribution

```
# Original distribution of attack types
attack_counts = df['Label'].value_counts()

# Pie chart before label encoding
plt.figure(figsize=(8, 8))
plt.pie(attack_counts, labels=attack_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Original Attack Types')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

A pie chart shows the distribution of the original attack types in the dataset before label encoding.

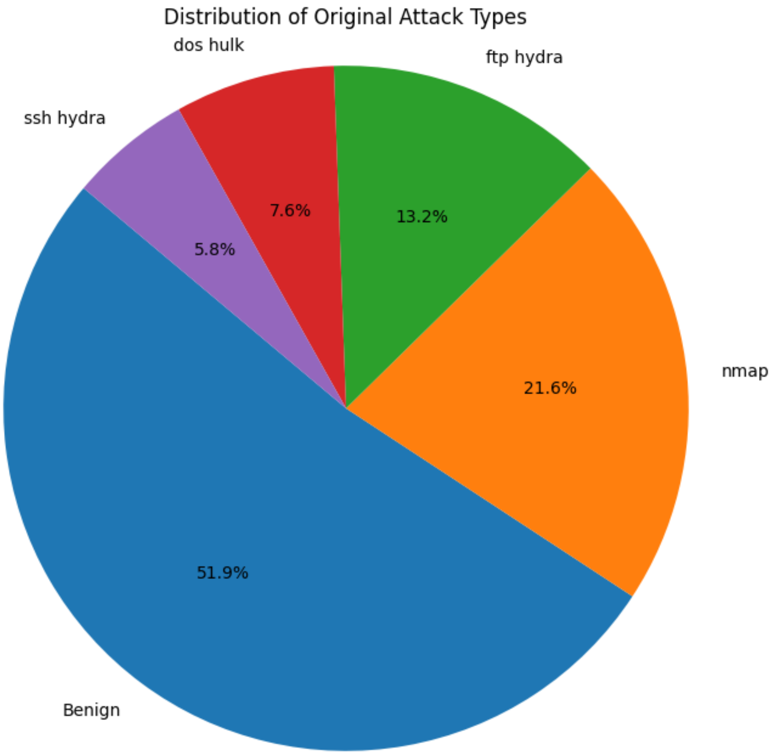


Figure 5. 2: Distribution of Original Attack Types

d. Label Encoding

```
# Transform labels: Attack = 1 / Benign = 0
df['Label'] = df['Label'].apply(lambda x: 1 if x != 'Benign' else 0)
```

Converts the Label column to binary format, where 1 represents malicious traffic, and 0 represents normal traffic.

e. **Visualize Data Distribution**

```
# New distribution after label encoding
binary_counts = df['Label'].value_counts()

# Pie chart after label encoding
plt.figure(figsize=(8, 8))
plt.pie(binary_counts, labels=['Normal (0)', 'Attack (1)'], autopct='%1.1f%%', startangle=140, colors=['#66b3ff', '#ff9999'])
plt.title('Distribution of Labels: Normal vs Attack')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

A pie chart shows the proportion of normal and malicious traffic in the dataset.

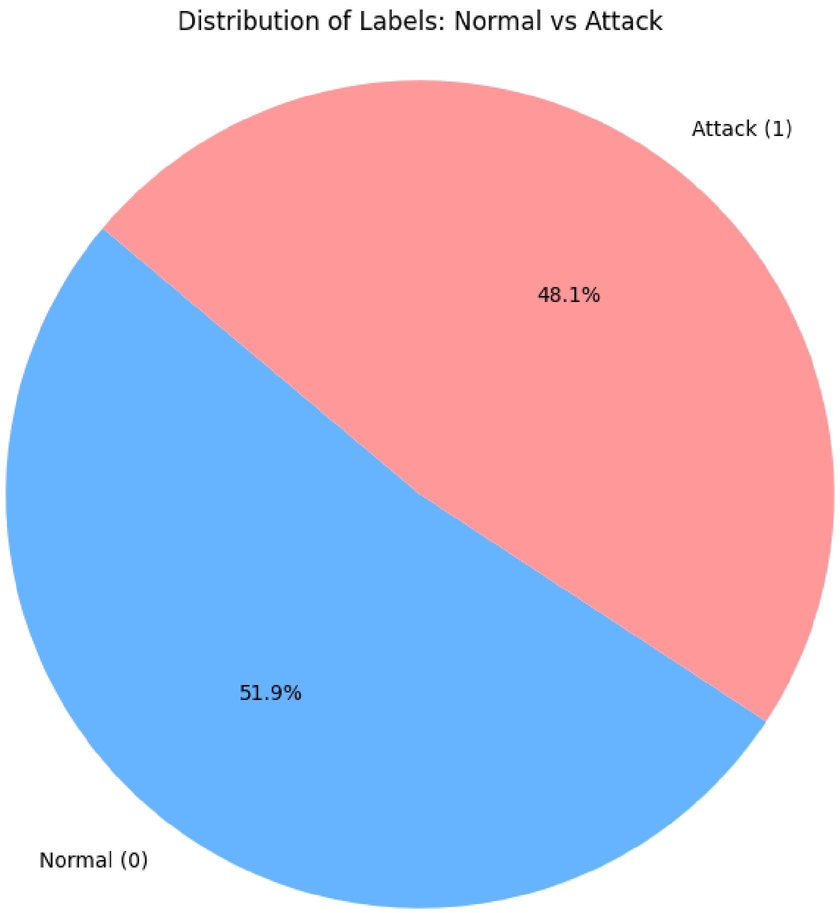


Figure 5. 3: Distribution of Labels

Data Splitting

```
# Separate features and labels
X = df.drop('Label', axis=1)
y = df['Label']

# Split the dataset into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y)
```

Data is split into training (75%) and testing (25%) sets to evaluate the model's performance.

Feature Scaling

```
# Initialize StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both train and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Save the scaler for real-time use
with open('EScaler.pkl', 'wb') as scaler_file:
    pickle.dump(scaler, scaler_file)
    print("Scaler saved.")
```

```
Scaler saved.
```

Standardizes features to have zero mean and unit variance for improved Logistic Regression performance. The scaler is saved for real-time use during predictions.

Model Training

```
# Initialize Logistic Regression
logistic_model = LogisticRegression(max_iter=20000, multi_class='ovr')

# Define hyperparameters for Logistic Regression
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

# Perform Grid Search with cross-validation
start_time = time.time()
gridsearch = GridSearchCV(logistic_model, param_grid, cv=5, verbose=3)
gridsearch.fit(X_train_scaled, y_train)
elapsed_time = time.time() - start_time

print(f"Finished. Elapsed time: {int(elapsed_time // 60)} min {int(elapsed_time % 60)} sec")
```

A Logistic Regression model is employed for binary classification, and Grid Search CV is used to optimize the regularization parameter 'C' by testing values within a specified range (0.001 to 1000) through 5-fold cross-validation. This process helps identify the best 'C' value for improved model performance, and the time taken for model training is measured to evaluate efficiency.

Model Evaluation

a. Prediction and Accuracy

```
# Make predictions
y_pred = gridsearch.predict(X_test_scaled)

# Accuracy score
accuracy = gridsearch.score(X_test_scaled, y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 86.06%
```

Evaluates the model's accuracy on the testing set.

b. Classification Report

```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.85      0.86      4290
           1       0.85      0.87      0.86      3982

    accuracy                           0.86      8272
   macro avg       0.86      0.86      0.86      8272
weighted avg       0.86      0.86      0.86      8272
```

Displays precision, recall, F1-score, and support for each class.

Model Deployment Preparation

```
# Save the trained model
with open('EModel.pkl', 'wb') as model_file:
    pickle.dump(gridsearch, model_file)
    print("Model saved.")
```

```
Model saved.
```

The trained Logistic Regression model is saved as a .pkl file for integration into the web application.

5.2 Testing

We tested our Network Intrusion Detection System (NIDS) for reliability, accuracy, and efficiency. Unit tests validated components like the machine learning model and data preprocessing, while system tests ensured real time packet capture, accurate predictions, and smooth user interactions. Results confirmed the system's robustness in effectively detecting network attacks.

5.2.1 Test Case for Unit Testing

Unit testing is a software testing method that focuses on validating the functionality of individual components or modules of a program in isolation. Each "unit" is the smallest testable part of an application, such as a function, method, or class. The goal of unit testing is to ensure that these components work as expected independently of other parts of the system.

Test ID	Test Cases	Expected Output	Actual Output	Result
01	Initialize FlowPredictor with valid model and scaler files	FlowPredictor object created successfully with loaded model and scaler	FlowPredictor instance initialized with EModel.pkl and EScaler.pkl	PASS
02	Test packet capture with valid network interface	Capture starts successfully and begins collecting packets	Packet capture initiated on specified interface	PASS
03	Extract flow features from captured packets	Dictionary containing all 72 features with calculated values	Complete feature set extracted from packet data	PASS
04	Calculate packet length	Dictionary with min, max, mean,	Correct statistical calculations returned	PASS

	statistics for flow	std, and total values		
05	Calculate Inter-Arrival Time (IAT) statistics	Dictionary with min, max, mean, std values for packet timing	Accurate IAT statistics calculated	PASS
06	Extract TCP protocol flags from packet	Dictionary with all 8 flag values (FIN, SYN, RST, etc.)	Correct flag values extracted from TCP packets	PASS
07	Calculate time-based features (active/idle times)	Tuple containing active and idle time statistics	Accurate time-based metrics calculated	PASS
08	Save flow data to CSV file	CSV file created with all flow records	Flow data successfully saved to file	PASS
09	Process packets with prediction model	Flow features extracted and prediction made	Packets processed and prediction returned	PASS
10	Login with valid credentials	User successfully authenticated and session created	Login successful, session created	PASS
11	Register new user with strong password	User account created in database	Account created with hashed password	PASS
12	Calculate network statistics from flow data	JSON with complete statistics including attack rates	Accurate statistics generated	PASS

13	Validate password complexity requirements	Boolean result indicating if password meets requirements	Correct validation of password strength	PASS
14	Process multiple concurrent packet flows	All flows processed without data loss	Concurrent flows handled correctly	PASS
15	Send admin alert email for repeated attacks	Email sent with attack details	Alert email delivered successfully	PASS
16	Get detected attacks list	JSON with attack flows and blocked IPs	Accurate attack detection list	PASS
17	Clear detected attacks	Flow data cleared of attack entries	Attack data successfully cleared	PASS
18	Verify dashboard UI elements and responsiveness	Dashboard elements render properly across devices.	Dashboard UI responsive and functioning correctly.	PASS
19	Test proper rendering of all web app pages	All pages render correctly with their intended content and UI elements.	All web pages render as expected with no missing elements or layout issues.	PASS

Table 5. 1: Test Case for Unit Testing

5.2.2 Test Case for System Testing

System testing is a type of software testing that evaluates the entire system to ensure it meets the specified requirements. It verifies the end-to-end functionality of the integrated application, including its interaction with external systems, under realistic conditions. The goal is to identify any defects or issues in the complete system before deployment.

Test ID	Test Case	Expected Output	Actual Output	Result
01	Registration Form Validation	The registration form should submit successfully. A new user account should be created in the system. The user should be redirected to the login page. A success message should be displayed confirming registration.	The registration form submitted successfully. A new user account was created in the system. The user was redirected to the login page. A success message was displayed.	Pass
02	Login Authentication	The user should be able to log in successfully using valid credentials. Upon login, the user should be redirected to their dashboard. A session should be created for the user.	The user logged in successfully using valid credentials. The system redirected the user to the dashboard. A session was	Pass

			created for the user.	
03	Responsive Design	The user interface should adapt seamlessly to different screen sizes (e.g., mobile, tablet, desktop). All features should remain accessible on various devices. There should be no horizontal scrolling required on smaller screens.	The user interface adapted perfectly to different screen sizes. All features were accessible on mobile, tablet, and desktop. No horizontal scrolling was observed.	Pass
04	Session Management	User sessions should automatically expire after 30 minutes of inactivity. The user should be redirected to the login page upon session expiration. A message should be displayed informing the user that their session has expired.	User sessions expired after 30 minutes of inactivity as expected. The user was redirected to the login page. A session expiration message was displayed.	Pass
05	Start Packet Capture	The system should start capturing packets successfully when initiated.	The system successfully started capturing	Pass

		Real-time data should be displayed during the capture. Network statistics should update dynamically throughout the process.	packets. Real-time data was displayed during the capture. Network statistics were dynamically updated.	
06	Attack Detection	The system should identify network attacks in real-time. Alerts should be generated for any detected attacks. Email notifications should be sent to the relevant recipients when attacks are detected.	The system successfully identified network attacks in real-time. Alerts were generated for the detected attacks. Email notifications were sent as expected.	Pass
07	CSV File Upload	The system should allow CSV files to be uploaded successfully. Once uploaded, the system should process the file and complete the operation. The results of the	CSV files were uploaded successfully. The system processed the files as expected. Results were displayed to the user.	Pass

		processing should be displayed to the user.		
08	Large File Processing	The system should handle large file uploads without any timeout or crashes. The results should be accurate even for large datasets. The overall processing should remain stable.	The system handled large file uploads without any issues. Results were accurate even for large datasets. Processing was stable throughout.	Pass
09	Result Export	An export option should be available to the user. The user should be able to download the exported file successfully. Data integrity should be maintained in the exported file.	The export option was available as expected. The user successfully downloaded the exported file. Data integrity was maintained in the exported file.	Pass

10	High Traffic Load	The system should remain stable even under high traffic loads. Accurate detection and functionality should be maintained during the load test.	The system remained stable under high traffic loads. Accurate detection and functionality were maintained throughout the load test.	Pass
11	Database Performance	The system should provide quick responses to database queries. Data retrieval should be efficient. There should be no timeout or performance issues during database interactions.	The system provided quick query responses. Data retrieval was efficient. There were no timeout or performance issues.	Pass
12	Password Security	User passwords should be properly hashed and stored securely. No plaintext passwords should be stored in the database.	Passwords were hashed and stored securely as expected. No plaintext passwords were stored in the database.	Pass

Table 5. 2: Test Case for System Testing

5.2.3 Test Case for Integration Testing

During the integration testing phase of the Network Intrusion Detection System (NIDS), various modules were combined to ensure their seamless operation as a unified system. For instance, the data preprocessing module was integrated with the feature extraction module to ensure accurate and relevant feature selection, while the machine learning model module was combined with the real-time intrusion detection engine for effective classification of network traffic. The alert generation module was also tested in conjunction with the dashboard and logging system to ensure that detected intrusions were accurately logged and displayed in real-time for network administrators. Additionally, the interdependencies between the network traffic monitoring, anomaly detection algorithms, and the user interface were verified to guarantee proper synchronization and usability. Throughout the integration phase, tests were conducted to ensure that each combination of modules worked together without errors, and the overall NIDS functioned efficiently and accurately.

5.3 Result Analysis

The testing demonstrated consistent and reliable results across various testing phases, including unit testing, system testing, and integrated testing. The successful execution of all test cases highlights the system's robustness in detecting and classifying network traffic under a wide range of conditions. With its ability to process real time traffic, detect attacks accurately, and perform under high traffic loads, the system meets the core requirements for an effective intrusion detection solution.

Confusion Matrix Analysis

The confusion matrix provides a detailed evaluation of the classification performance:

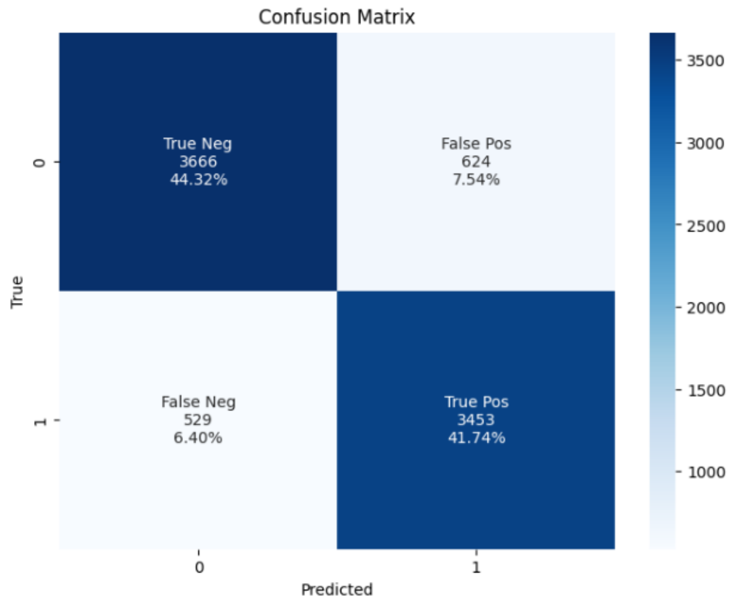


Figure 5. 4: Confusion Matrix Analysis

True Negatives (TN): 3666 instances of benign traffic were correctly identified.

False Positives (FP): 624 benign traffic instances were misclassified as attacks.

False Negatives (FN): 529 attack instances were missed and classified as benign.

True Positives (TP): 3453 attack instances were correctly identified.

Precision

Precision indicates the accuracy of positive predictions by calculating the proportion of true positives out of all predicted positives.

Precision for Class 0 (Benign): 87%

Precision for Class 1 (Attack): 85%

Weighted Average Precision: 86%

Recall

Recall measures the model's ability to identify all actual positive instances by calculating the proportion of true positives out of all actual positives.

Recall for Class 0 (Benign): 85%

Recall for Class 1 (Attack): 87%

Weighted Average Recall: 86%

F1-Score

The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that balances the trade-off between Precision and Recall.

F1-Score for Class 0 (Benign): 86%

F1-Score for Class 1 (Attack): 86%

Weighted Average F1-Score: 86%

Accuracy

Accuracy measures the ratio of correctly classified instances (true positives + true negatives) to the total number of instances.

Overall Accuracy: 86.06%

Summary Table

Metric	**Class 0 (Benign)**	**Class 1 (Attack)**	**Weighted Average**
Precision	87%	85%	86%
Recall	85%	87%	86%
F1-Score	86%	86%	86%
Accuracy	-	-	86.06%

Table 5. 3: Summary Table

The NIDS system performed effectively, achieving an accuracy of 86.06%, which aligns well with industry standards for intrusion detection systems. The confusion matrix shows a reliable balance between correctly classified benign and attack instances. Moving forward, additional fine-tuning of the machine learning model and feature extraction methods could further reduce misclassification rates, enhancing overall detection accuracy.

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATION

6.1 Conclusion

This project successfully developed a machine learning based Network Intrusion Detection System (NIDS) to address the increasing need for robust and adaptive cybersecurity measures in the face of evolving threats. By applying supervised learning algorithms like Logistic Regression, the system efficiently classifies network traffic as either normal or malicious. It features secure admin login, real time traffic preprocessing, CSV file upload for offline analysis, and automated alerts, ensuring comprehensive network monitoring and management. The adoption of agile methodology throughout the development process allowed for iterative improvements in model accuracy, system performance, and overall usability. By evaluating various machine learning algorithms, the project highlights the potential of supervised learning techniques in enhancing intrusion detection capabilities, with a focus on optimizing computational efficiency and detection reliability. While the system successfully detects key attack types such as port scanning, brute force attacks, and denial-of-service (DoS) attacks, it is not without limitations. The need for continuous model retraining, challenges in feature extraction from complex network data, and scalability concerns for large-scale networks underscore areas for future improvement. Nevertheless, the project demonstrates the effectiveness of machine learning driven solutions in modern cybersecurity and lays the groundwork for more advanced, scalable, and adaptive intrusion detection systems. Additionally, the system is currently limited to specific attack types, leaving room for future expansion to cover a broader range of intrusions.

In conclusion, this project highlights the effectiveness of machine learning in cybersecurity by replacing traditional static methods with a dynamic, adaptive, and scalable framework. It provides a real time solution for intrusion detection, contributing to the development of intelligent cybersecurity systems. With further refinements, this system can evolve to address an even wider range of cyber threats, ensuring stronger protection for modern digital infrastructures.

6.2 Future Recommendations

- **Machine Learning Refinement:**

Continuously refine and update machine learning models to improve prediction accuracy and adaptability.

- **Expansion to Additional Attack Types:**

The system can be extended to detect a wider range of attacks, including advanced persistent threats (APTs), ransomware, and zero-day vulnerabilities. This will increase the scope and robustness of the system.

- **Scalability for Large-Scale Networks:**

Make the system faster and better for large networks. This includes deploying the system in distributed architectures using cloud platforms or containerization tools like Docker.

- **Real-Time Threat Prevention:**

Include features to block threats automatically in real time.

- **Incorporation of Feedback Mechanisms:**

Adding a feedback loop where administrators can label detected traffic as false positives or true positives will enable the system to learn and improve over time through reinforcement.

- **Advanced Visualization and Reporting:**

Enhancing the dashboard with advanced visualization tools like heatmaps, real-time graphs, and predictive analytics will provide deeper insights into network behavior and intrusion trends.

- **Mobile and IoT Network Monitoring:**

Extend the system to monitor mobile devices and IoT networks.

REFERENCES

[1] R. Saha, G. Kumar, M. K. Rai, and H.-J. Kim, "Adaptive classifier-based intrusion detection system using logistic regression and Euclidean distance on network probe vectors in resource-constrained networks," Int. J. Inf. Computer Security, vol. 16, no. 3/4, pp. XX–XX, Nov. 2021.

[2] M. University, "Machine Learning Classification of Port Scanning and DDoS Attacks: A Comparative Analysis," *Mehr. Univ. Res. J. Eng. Technol.*, vol. 40, no.1, pp.215-229, Jan2021.

[3] G. Haixiang, Y. Li, J. Shang, G. Mingyun, H. Yuanyue, and B. Gong, "Learning from class-imbalanced data: Review of methods and applications," ResearchGate, Dec.2016.

[4] Z. Zhong, "Improving model robustness through hybrid adversarial training: Integrating FGSM and PGD methods," Applied and Computational Engineering, vol.109, no.1, pp57-62, Nov2024.

[5] M. K. Baklizi, I. Atoum, M. Alkhazaleh, and H. Kanaker, "Web attack intrusion detection system using machine learning techniques," International Journal of Online and Biomedical Engineering (iJOE), vol.20, no. 03, pp. 24-38, Feb. 2024.

[6] R. R. S, R. R. Yadav, M. Moharir, and G. Shobha, "SCAPY - A powerful interactive packet manipulation program," in Proc. 2018 Int. Conf. Netw., Embedded Wireless Syst. (ICNEWS), Dec.2018. doi :10.1109.

[7] M. Bhuyan, D. K. Bhattacharyya, and J. Kalita, "Surveying port scans and their detection methodologies," The Computer Journal, vol. 54, pp. 1565–1581, Oct. 2011.

[8] R. A. Grimes, "Brute-force attacks," in Hacking Multifactor Authentication, pp. 295–306, Oct.2020.

[9] K. B. Adedeji, A. M. Abu-Mahfouz, and A. M. Kurien, "DDoS attack and detection methods in internet-enabled networks: Concept, research perspectives, and challenges," Journal of Sensor and Actuator Networks, vol. 12, no. 4, p. 51, Jul. 2023.

[10] R. Tahri, Y. Balouki, A. Jarrar, and L. Abdellatif, “Intrusion detection system using machine learning algorithms,” ITM Web of Conferences, vol. 46, no. 6, p. 02003,2022.

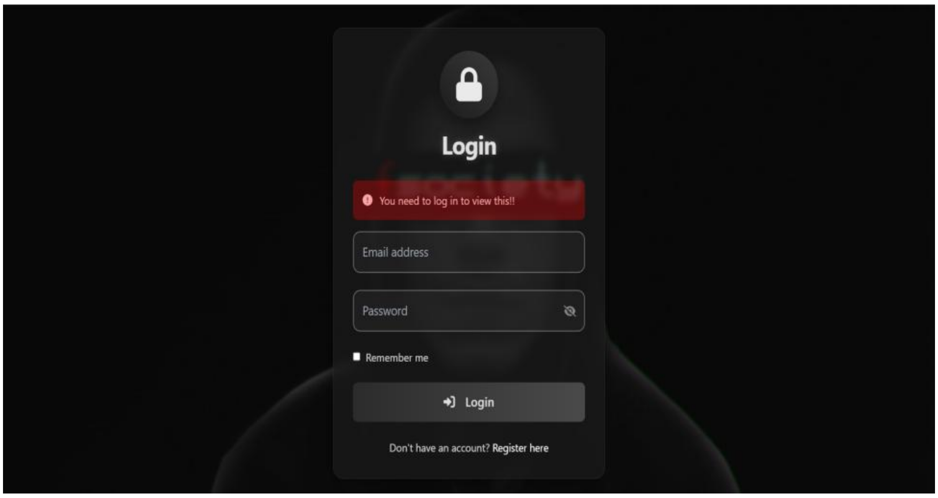
APPENDICES

Snapshots:

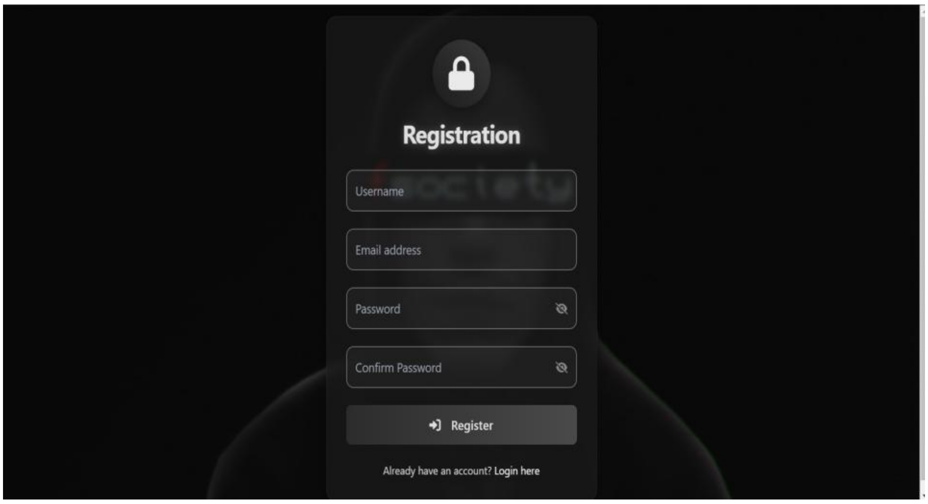
1. Homepage

2. Online Mode

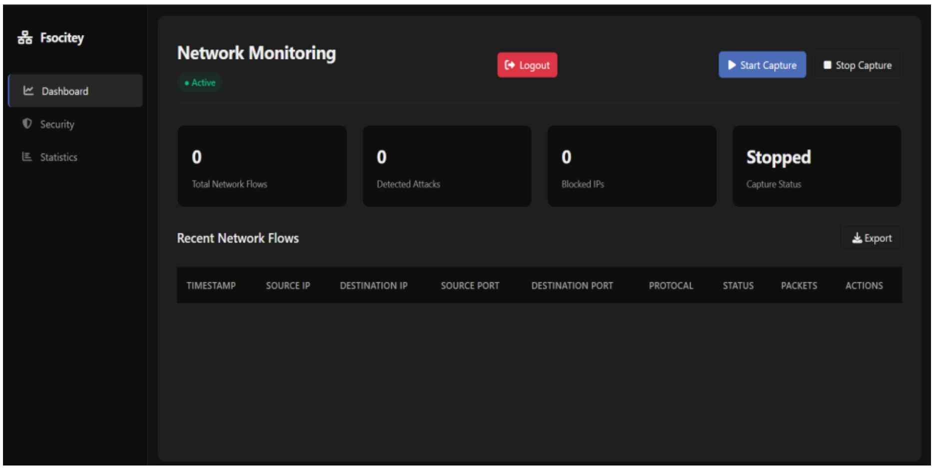
i. Login Page

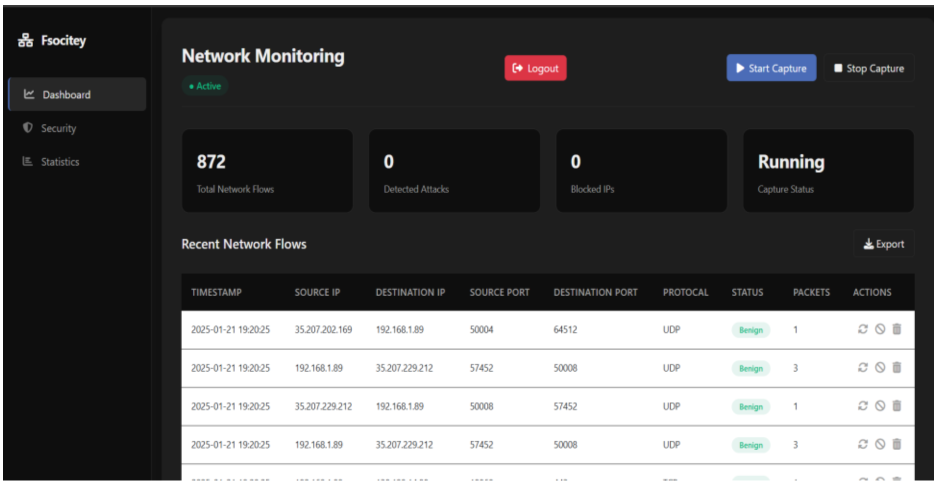


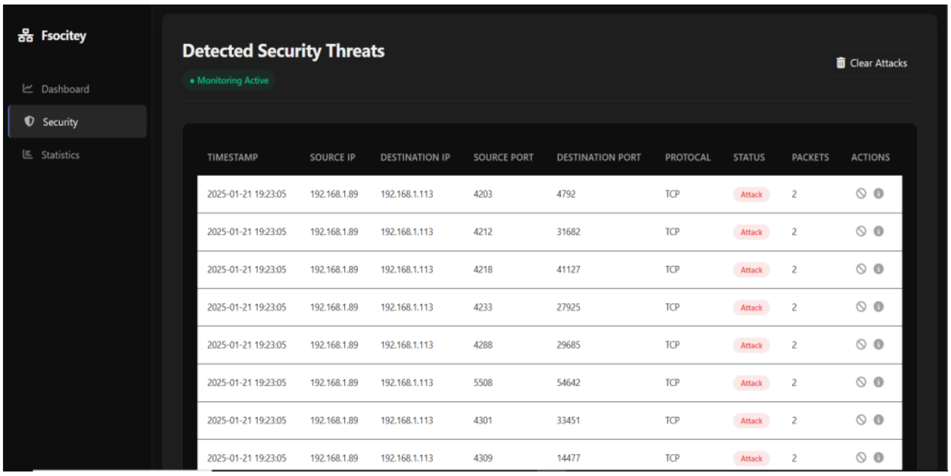
ii. Registration Page



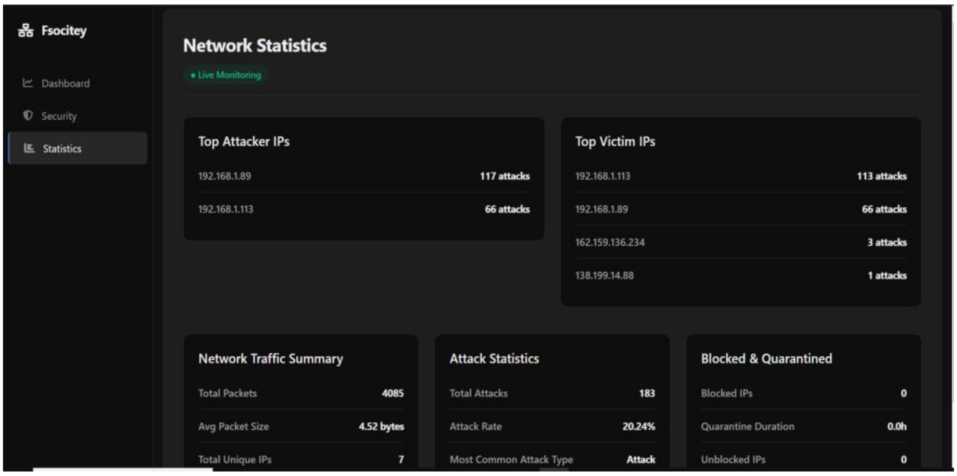
iii. Dashboard

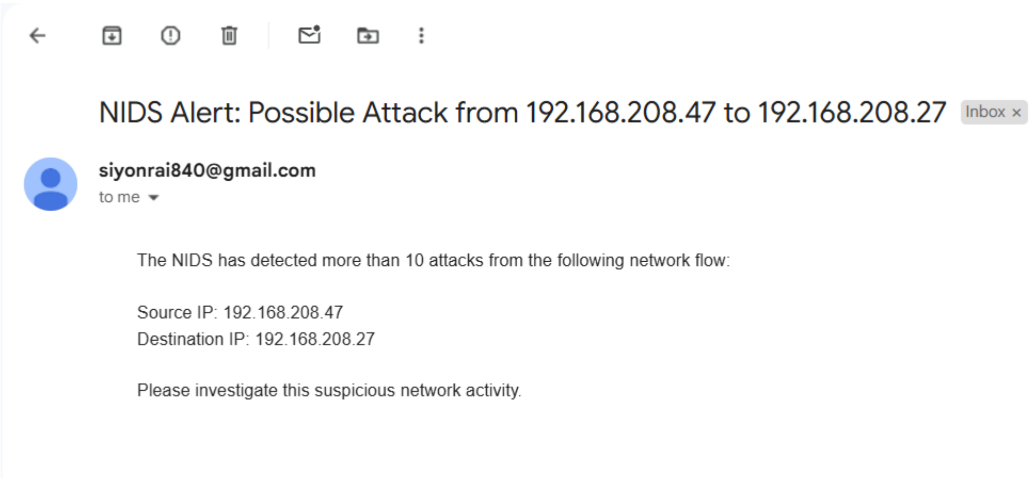






TIMESTAMP	SOURCE IP	DESTINATION IP	SOURCE PORT	DESTINATION PORT	PROTOCAL	STATUS	PACKETS	ACTIONS
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4203	4792	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4212	31682	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4218	41127	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4233	27925	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4288	29685	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	5508	54642	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4301	33451	TCP	Attack	2	
2025-01-21 19:23:05	192.168.1.89	192.168.1.113	4309	14477	TCP	Attack	2	





NIDS Alert: Possible Attack from 192.168.208.47 to 192.168.208.27 Inbox ×

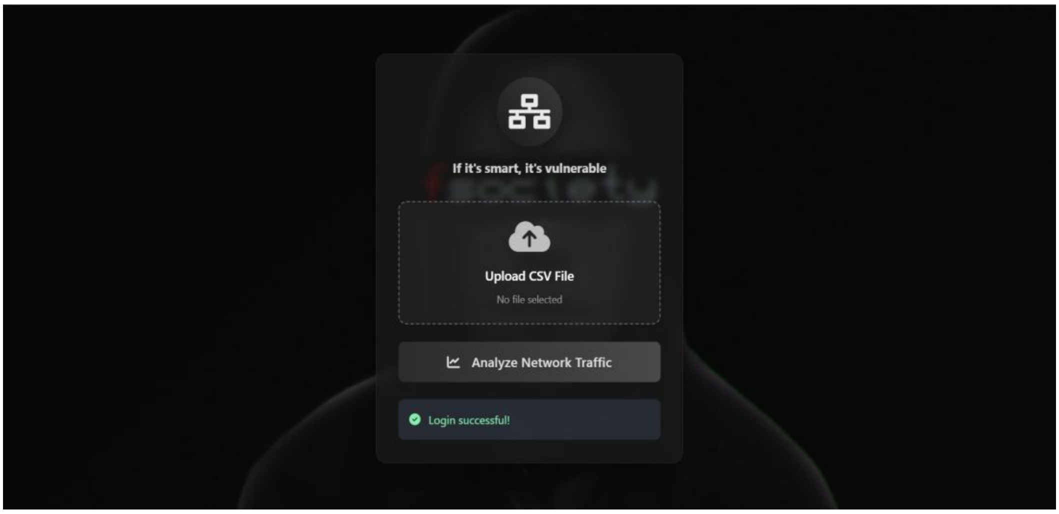
siyonrai840@gmail.com
to me

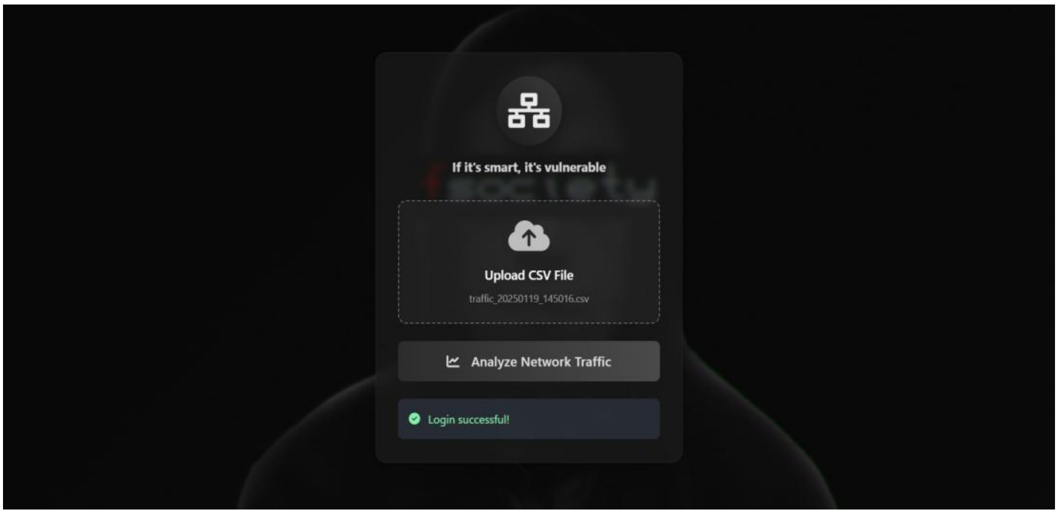
The NIDS has detected more than 10 attacks from the following network flow:

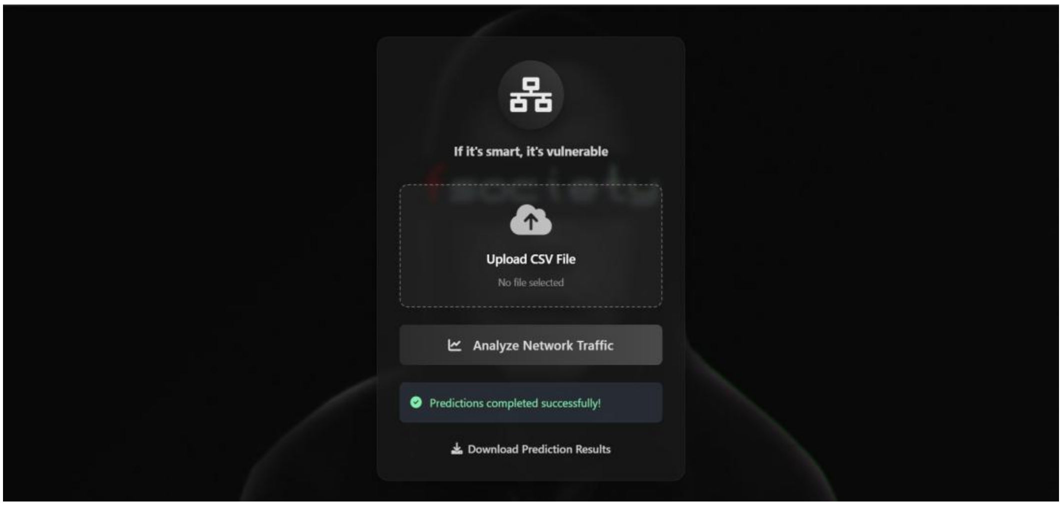
Source IP: 192.168.208.47
Destination IP: 192.168.208.27

Please investigate this suspicious network activity.

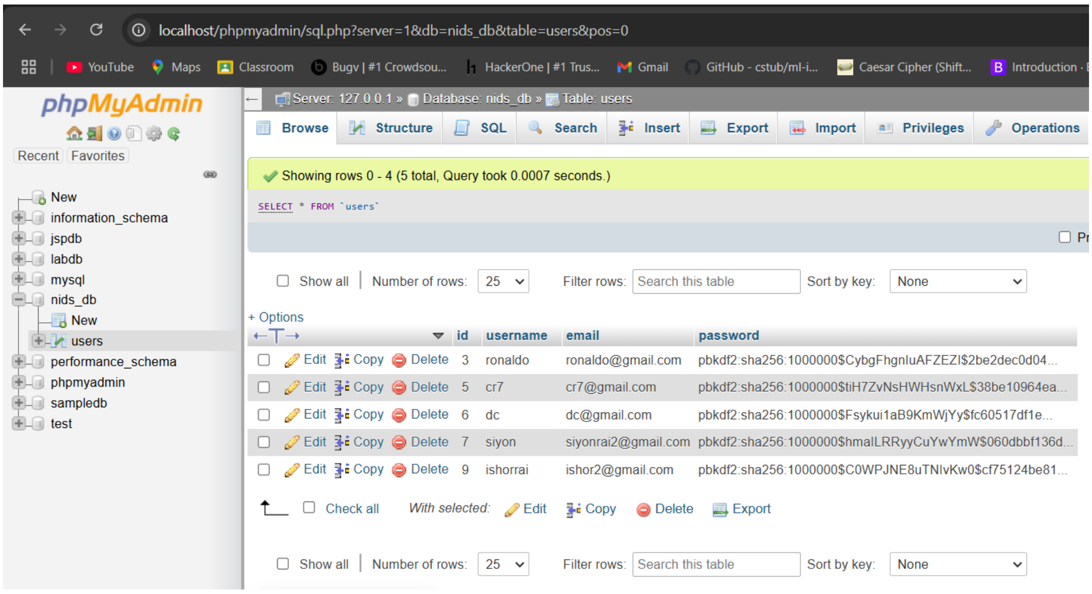
3. Offline Mode







4. Database


localhost/phpmyadmin/sql.php?server=1&db=nids_db&table=users&pos=0
YouTube
Maps
Classroom
Bugv | #1 Crowdsou...
HackerOne | #1 Trus...
Gmail
GitHub - cstub/ml-i...
Caesar Cipher (Shift...
Introduction
phpMyAdmin
Recent
Favorites
New
information_schema
jspdb
labdb
mysql
nids_db
New
users
performance_schema
phpmyadmin
sampledb
test
Server: 127.0.0.1 » Database: nids_db » Table: users
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Showing rows 0 - 4 (5 total, Query took 0.0007 seconds.)
SELECT * FROM `users`
Show all
Number of rows:
25
Filter rows:
Search this table
Sort by key:
None
+ Options
id
username
email
password
Edit Copy Delete 3 ronaldo ronaldo@gmail.com pbkdf2:sha256:1000000$CybgFhgnluAFZEZI$2be2dec0d04...
Edit Copy Delete 5 cr7 cr7@gmail.com pbkdf2:sha256:1000000$tiH7ZvNsHWHsnWxL$38be10964ea...
Edit Copy Delete 6 dc dc@gmail.com pbkdf2:sha256:1000000$Fsykui1aB9KmWjYy$fc60517df1e...
Edit Copy Delete 7 siyon siyonrai2@gmail.com pbkdf2:sha256:1000000$hmalLRRyyCuYwYmW$060dbbf136d...
Edit Copy Delete 9 ishorrai ishor2@gmail.com pbkdf2:sha256:1000000$COWPJNE8uTNlvKw0$cf75124be81...
Check all
With selected:
Edit
Copy
Delete
Export
Show all
Number of rows:
25
Filter rows:
Search this table
Sort by key:
None