A Project Report on

Detection Of Cyber Attacks in a Network Using Machine Learning Techniques

Submitted in partial fulfillment for award of

Bachelor of Technology

Degree

in

Computer Science and Engineering

By

R. Venkatesh (Y20ACS547) **SK. Shifa Anjum (Y20ACS568)**

T. Narendra (Y20ACS577) **S. Sai Jagadeesh (Y20ACS569)**

Under the guidance of

Mr. J. Madhan Kumar, **Asst. Prof.**

Designation

Department of Computer Science and Engineering

Bapatla Engineering College

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

BAPATLA – 522 102, Andhra Pradesh, INDIA

2023-2024

Department of

Computer Science and Engineering

CERTIFICATE

This is to certify that the project report entitled **Detection of Cyber Attacks in a Network using Machine Learning Techniques** that is being submitted by R. Venkatesh (Y20ACS547), SK. Shifa Anjum (Y20ACS568), T. Narendra (Y20ACS577), S. Sai Jagadeesh (Y20ACS569) in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science & Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

Signature of the Guide
Mr. J. Madhan Kumar
Asst. Professor

Signature of the HOD
Dr. M. Rajesh Babu
Assoc. Prof. & Head

DECLARATION

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

R. Venkatesh (Y20ACS547)

SK. Shifa Anjum (Y20ACS568)

T. Narendra (Y20ACS577)

S. Sai Jagadeesh (Y20ACS569)

Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide [**Name of the guide**, designation of guide], Department of CSE, for his/her valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Dr. M. Rajesh Babu**, Assoc. Prof. & Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal **Dr. Nazeer Shaik** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator **Dr. N. Sudhakar,** Prof. Dept. of CSE for his helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

R. Venkatesh (Y20ACS547)

SK. Shifa Anjum (Y20ACS568)

T. Narendra (Y20ACS577)

S. Sai Jagadeesh (Y20ACS569)

Table of Contents

List of Figures

List of Tables

Abstract

In today's interconnected digital landscape, the threat of cyber attacks looms large, necessitating robust defense mechanisms. This project proposes a novel approach leveraging machine learning (ML) techniques for the detection of cyber attacks within network infrastructures. By harnessing the power of ML algorithms, including supervised, unsupervised, and deep learning methods, the system aims to identify anomalous patterns indicative of malicious activity.

The dataset comprises network traffic data, enriched with labeled instances of known attacks for supervised learning, and unlabeled instances for unsupervised techniques. Through feature engineering, dimensionality reduction, and model optimization, the system can effectively discern between normal and malicious network behavior in real-time. The evaluation of the proposed methodology demonstrates promising results in terms of accuracy, precision, recall, and F1-score, highlighting its potential to enhance cybersecurity defenses and mitigate the impact of cyber threats.

1 Introduction

In today's interconnected world, the security of computer networks is of paramount importance. With the increasing prevalence of cyber threats and attacks, organizations and individuals alike face the challenge of safeguarding their networks from malicious actors. Traditional methods of network security, such as firewalls and intrusion detection systems (IDS), are no longer sufficient to defend against the evolving tactics employed by cybercriminals. As a result, there is a growing need for advanced techniques capable of detecting and mitigating network attacks in real-time.

Machine learning (ML) has emerged as a powerful tool in the field of network security, offering the potential to identify anomalous behavior and detect malicious activities with high accuracy. By leveraging large datasets and sophisticated algorithms, ML models can learn to distinguish between normal network traffic and suspicious patterns indicative of an attack. This capability makes ML-based intrusion detection systems (IDS) an attractive option for enhancing the security posture of organizations and minimizing the impact of cyber threats.

In this project, we focus on the task of detecting network attacks using machine learning techniques. Our objective is to develop a robust and effective IDS capable of accurately identifying various types of attacks in network traffic data. To achieve this goal, we leverage the CICIDS2017 dataset, a widely used benchmark dataset in the field of network security. This dataset contains a diverse set of network traffic flows, including both benign and malicious activities, making it well-suited for training and evaluating ML models for intrusion detection.

In the following sections of this report, we provide a detailed overview of our methodology, implementation, and experimental results. We describe the preprocessing steps applied to the CICIDS2017 dataset, the machine learning algorithms used for training our models, and the performance metrics used to evaluate their effectiveness. Additionally, we discuss the insights gained from our experiments, the challenges encountered during the project, and potential avenues for future research.

Overall, this project contributes to the ongoing efforts to enhance network security through the application of machine learning techniques. By developing an effective IDS capable of detecting network attacks, we aim to empower organizations with the tools they need to defend against cyber threats and safeguard their critical assets.

1.1 Classification:

Classification is the process of categorizing data or objects into predefined groups or classes based on their characteristics, attributes, or features. It's a fundamental task in various fields such as machine learning, statistics, and information retrieval. The goal of classification is to assign new or unknown instances to the appropriate class based on the patterns learned from the training data. This enables efficient organization, analysis, and decision-making on the data.

1.2 Binary Classification:

Binary classification is a specific type of classification task where the goal is to categorize data into one of two classes or categories. In binary classification, the output variable or target variable can take on only two possible values, typically represented as "0" and "1", or "negative" and "positive", "yes" and "no", etc. The aim is to build a

model that can accurately distinguish between the two classes based on the input features provided. This type of classification is commonly used in various applications such as spam detection, medical diagnosis, sentiment analysis, and fraud detection.

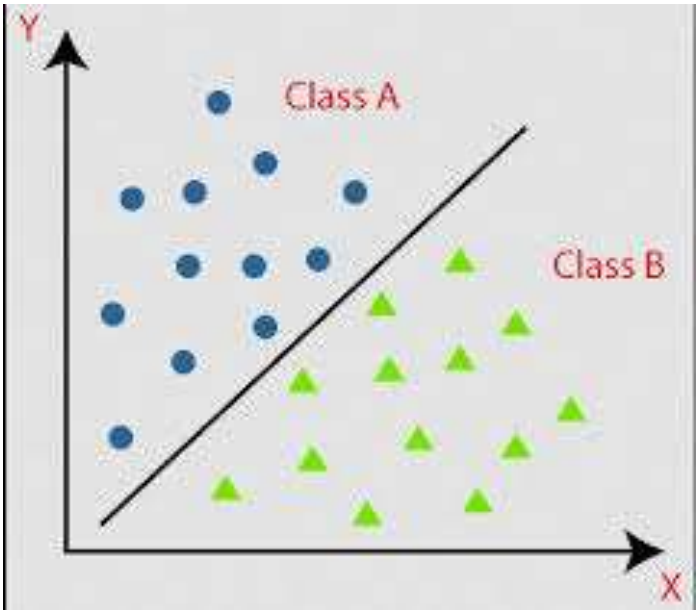

Figure 1.1 Binary Classification

1.3 Multi Class Classification:

Multiclass classification is a type of classification task where the goal is to categorize data into more than two classes or categories. In contrast to binary classification, where there are only two possible outcomes, multiclass classification involves predicting one class label from a set of three or more possible classes. The model is trained to differentiate between multiple classes and assign the most appropriate label to each instance based on its features.

Multiclass classification enables categorization into multiple classes, aiding tasks like object identification, topic modeling, and pattern recognition. Its applications

span handwriting and speech recognition, image and document classification, medical diagnosis, bioinformatics, natural language processing, and remote sensing.

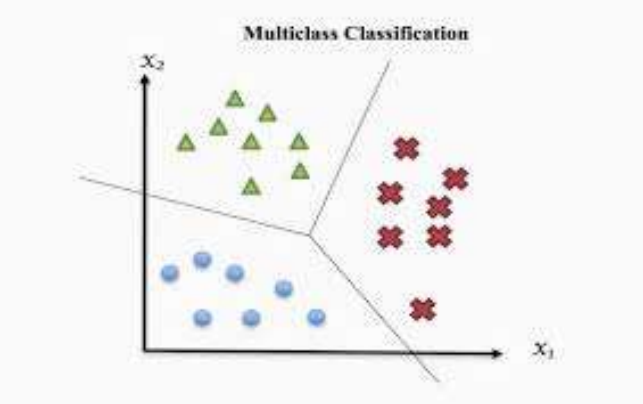

Figure 1.2 Multi Class Classification

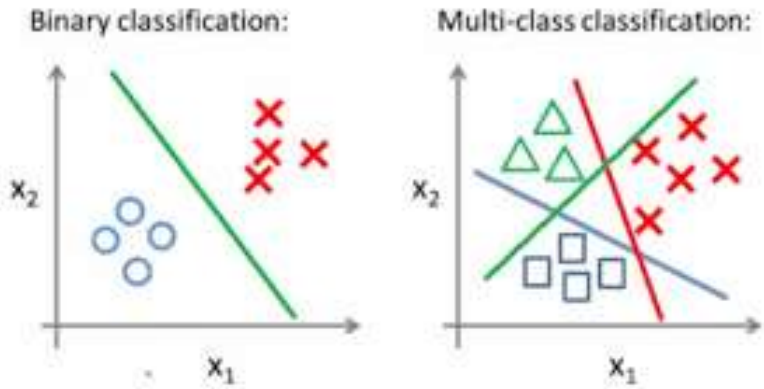

Figure 1.3 Binary v/s Multi Class Classification

1.4 XGBoost:

XGBoost, short for Extreme Gradient Boosting, is a popular machine learning algorithm known for its performance and efficiency in regression and classification tasks. It uses a gradient boosting framework, sequentially adding weak learners (decision trees) to minimize a predefined objective function. XGBoost is widely used

in competitions and real-world applications due to its scalability, speed, and ability to handle large datasets.

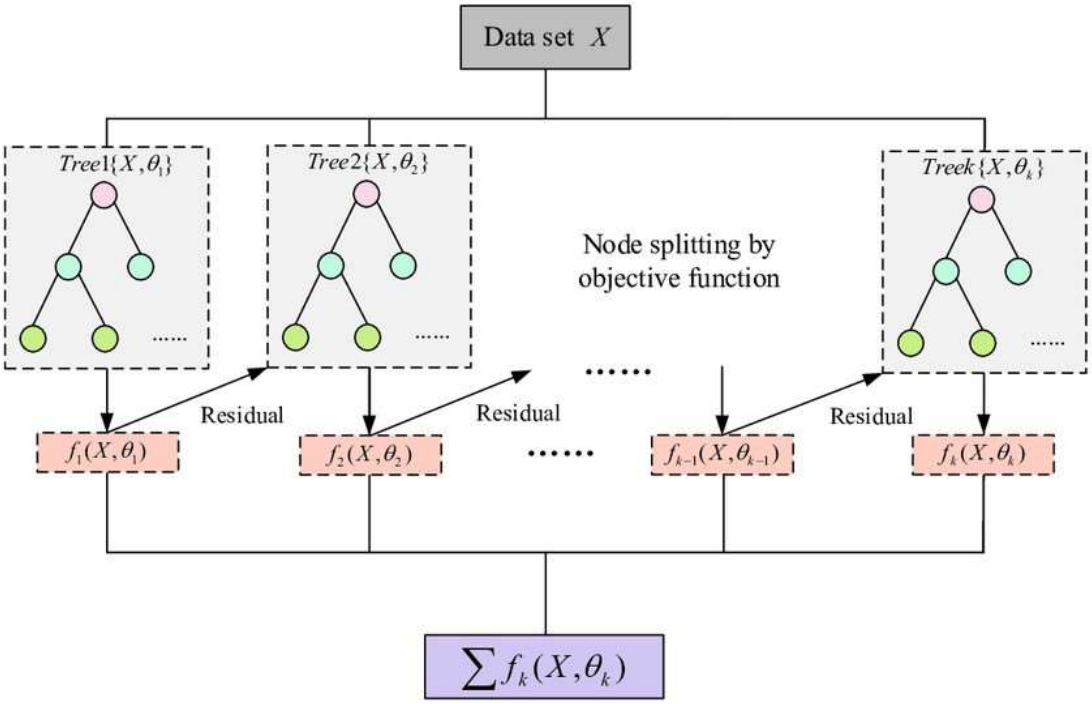

Figure 1.4 XGBoost

1.5 Decision Trees:

A decision tree is a supervised learning algorithm used for both classification and regression tasks. It partitions the feature space into disjoint regions and predicts the target variable based on the majority class or average value within each region. The tree is constructed recursively by selecting the best split at each node based on a chosen criterion (e.g., Gini impurity, information gain). Splitting continues until a stopping criterion is met, such as reaching a maximum depth or minimum number of samples per leaf node. Decision trees are interpretable and easy to visualize, making them suitable for understanding feature importance and decision-making processes.

However, they are prone to overfitting, especially when the tree becomes overly complex. Regularization techniques like pruning can mitigate this issue.

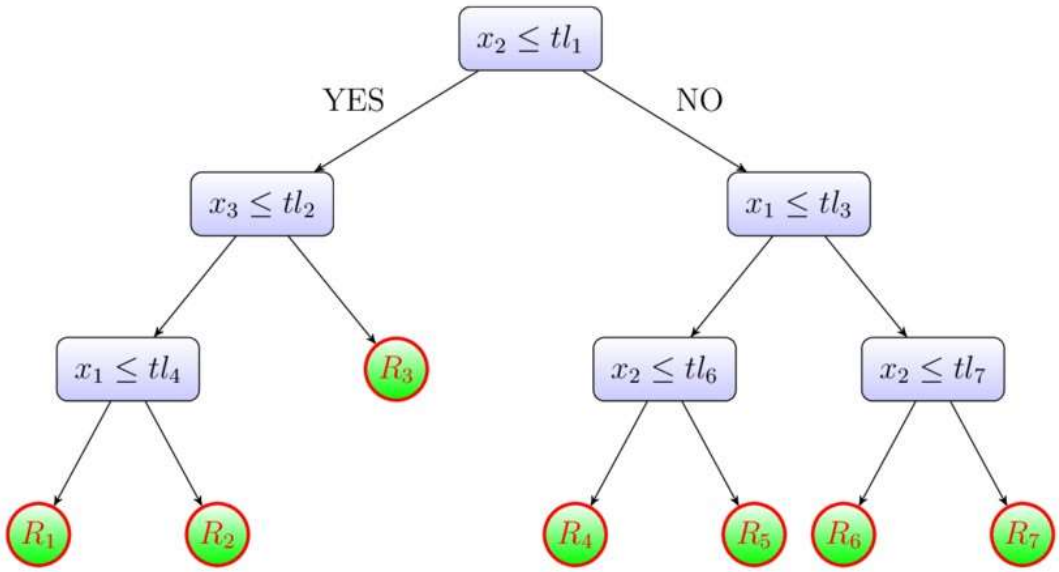

Figure 1.5 Decision Tree

1.6 Random Forest:

Random Forest is an ensemble learning algorithm used for classification and regression tasks. It builds multiple decision trees during training and combines their predictions through ensemble averaging or voting. Each decision tree is trained on a random subset of the dataset with replacement, introducing variability and reducing correlation between trees. Feature randomness is also introduced by considering only a random subset of features at each split point. Predictions are made by aggregating the outputs of individual trees, either by selecting the majority class for classification or averaging the predictions for regression. Random Forest is known for its robustness to overfitting, handling of high-dimensional datasets, and implicit feature selection capabilities.

Overall, it is a versatile and powerful algorithm widely used in machine learning applications..

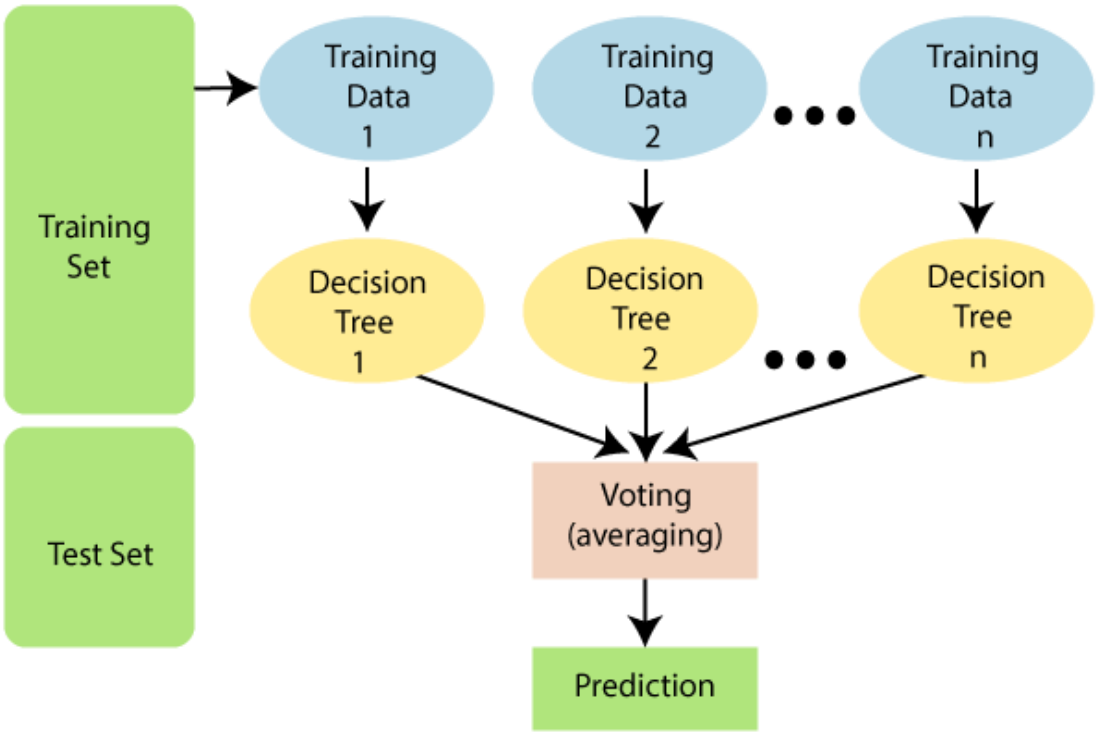

Figure 1.6 Random Forest

1.7 Multi Layer Perceptron

A Multilayer Perceptron (MLP) is a type of artificial neural network consisting of multiple layers of interconnected nodes or neurons. It typically includes an input layer, one or more hidden layers, and an output layer. Neurons in each layer are connected to neurons in adjacent layers through weighted connections. MLPs are capable of learning complex nonlinear relationships between input and output data through a process called backpropagation, where errors are propagated backward through the network to adjust the weights. They are commonly used for tasks such as classification, regression, and pattern recognition in machine learning.

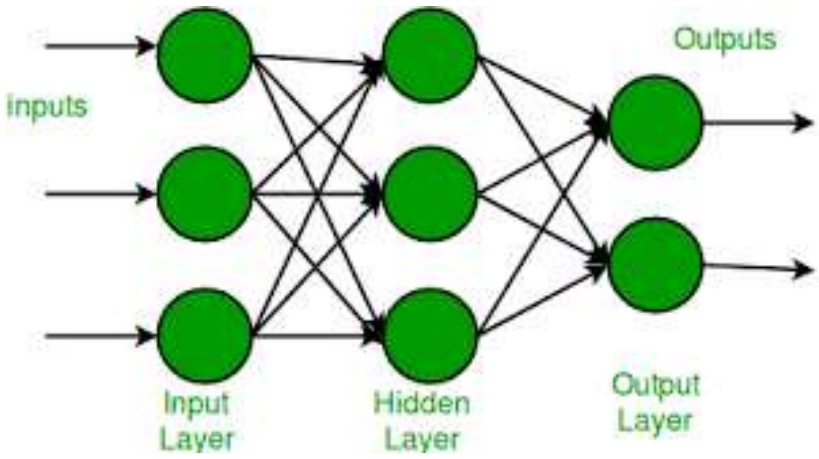

Figure 1.7 Multi-Layer Perceptron

1.8 Gradient Boost

Gradient Boosting is a machine learning technique where weak learners are sequentially trained to correct errors made by the previous models. Each subsequent model focuses on the residual errors of the ensemble, gradually improving the overall prediction. It combines multiple simple models, typically decision trees, to create a strong predictive model. Gradient Boosting is known for its high predictive accuracy and is widely used in various domains, including regression and classification tasks.

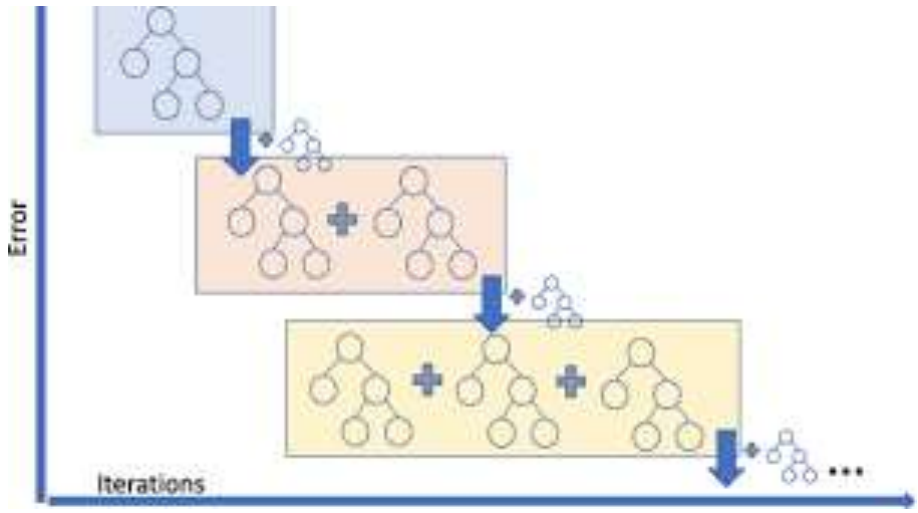

Figure 1.8 Gradient Boost

1.9 Over fitting:

Overfitting occurs when a machine learning model learns the training data too well, capturing noise or random fluctuations in the data rather than the underlying relationships. This results in a model that performs very well on the training data but generalizes poorly to new, unseen data.

1.10 Under fitting:

Underfitting occurs when a model is too simple to capture the underlying structure of the data. In other words, the model is not able to learn from the training data effectively and performs poorly on both the training data and unseen data.

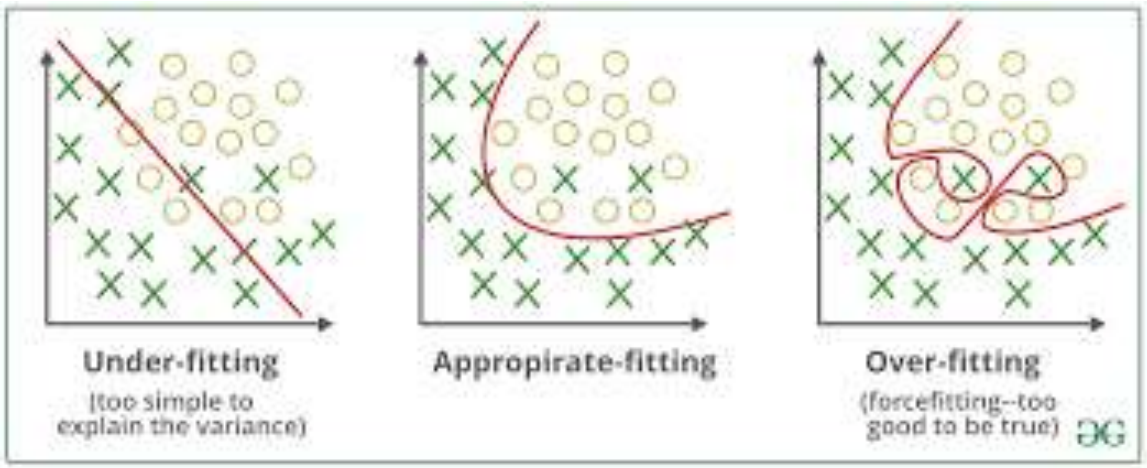

Figure 1.9 Under fitting v/s Over fitting

2 Literature Survey

Meftah et al., [1] proposed anomaly-based NIDS with machine learning techniques. Random forest with 10-fold crossvalidation to assign the index of feature significance in reducing impurity in the whole forest. The top features of UNSW-NB15 Dataset are ct dst src ltm, ct srv dst, ct dst sport ltm, ct src dport ltm, ct srv src. Support vector machine with an accuracy of 82.11% outperformed Logistic Regression and Gradient Boost Machine in binary classification model for attack detection. For identifying the type of attack, the multi-classification model with Decision Tree C5.0, outperformed Naive Bayes and Support vector machine.

DT classifier is a multistage decision-making model that suits numerical and nominal data [2]. It produces fast decisions as it consists of a limited number of nested simple conditional statements.

A Random Forest classifier is an ensemble classification model that trains a set of CART trees by bagging to make predictions. This model randomly selects features to set a decision on nodes and uses out-of-bag error estimation and final class decision by averaging individual tree class assignment probability [3].

Adaboost builds a robust predictive model by improving several weaker models. It is the best out-of-the-box classifier with no tweaking parameters and is less prone to over-fitting [4].

Most of the existing NIDS are signature-based or anomaly-based detection systems. Signature-based NIDS addresses only the list of known threats, and their indicators of compromise. It has a high processing speed and great accuracy for known

attacks. Still, it fails to identify the zero-day attack and unnecessarily raises alerts regardless of the outcome, like Window worm trying to attack the Linux system [5]. It is not practical for internal attacks and depends on the operating system, version, and applications [6]. Anomaly-based NIDS can detect a new suspicious behavior deviating from normalized behavior. Anomaly-based NIDS is good in detecting zero-day attacks. Still, the increased likelihood of false positives results in additional time and resources to investigate all the alerts to potential threats.

The CICIDS dataset, provided by the Canadian Institute for Cybersecurity, serves as a pivotal resource for evaluating the efficacy of intrusion detection systems. Featuring a comprehensive array of network traffic data, encompassing both benign and malicious activities within a simulated environment, it facilitates robust testing scenarios. Incorporating diverse features such as flow statistics, payload details, and behavioral attributes, CICIDS enables researchers to develop and assess intrusion detection algorithms effectively. Its realistic representation of cyber threats renders it indispensable for advancing cybersecurity research and fostering educational initiatives. [7]

3 Methodology

3.1 Existing System:

Intrusion Detection Systems (IDS) are pivotal components within cybersecurity frameworks, tasked with monitoring network traffic and system activities to detect potential security breaches or policy violations.

3.1.1 Functionality:

NIDS are deployed at strategic points within a network infrastructure, where they continuously analyse network traffic in real-time.

3.1.2 Detection Mechanisms:

These systems employ various detection mechanisms such as signature-based detection, which compares incoming traffic against a database of known attack signatures, and anomaly-based detection, which identifies deviations from normal network behavior.

3.1.3 Use Cases:

NIDS are effective in detecting a wide array of network-based attacks including but not limited to denial-of-service (DoS) attacks, port scans, and malware propagation.

3.1.4 Key Principles of IDS Systems:

The key principles used in traditional IDS systems are listed below. Each principle is a variation of pre-programming techniques to identify various anomalies in a network for detecting attacks or other deviations from normal behavior.

i. Signature-based Detection: Signature-based detection is one of the most direct and well-established methods to identify malicious activity. Signature-based detection examines network traffic, compares it to known signatures, and generates an alert when a match is made.

ii. Anomaly-based Detection: An anomaly-based intrusion detection system, is an intrusion detection system for detecting both network and computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous.

iii. Heuristic-based Detection: Heuristic analysis detects and removes a heuristic virus by first checking files in your computer, as well as code that may be behaving in a suspicious manner. Once a potential threat has been identified, it gets flagged. At this point, the threat can be removed from your system.

iv. Alerting and Response: Upon detecting suspicious activity, IDS systems generate alerts and notify security administrators or response teams, allowing them to investigate and respond to potential threats promptly.

v. Logging and Reporting: IDS systems maintain logs of detected events, including alerts, detected attacks, and response actions taken, facilitating forensic analysis, compliance reporting, and security posture improvement.

3.1.5 Conclusion:

Existing IDS systems play a critical role in safeguarding organizational networks and systems by providing continuous monitoring, detection, and response capabilities against evolving cyber threats. However, they face challenges such as false positives,

evasion techniques by attackers, and the need for regular updates to detection mechanisms to address emerging threats.

3.2 Proposed System:

The proposed system for network attack detection represents a significant step forward in fortifying organizational cybersecurity postures against evolving cyber threats. Building upon the foundation of traditional intrusion detection systems (IDS), the proposed system introduces advanced detection mechanisms and proactive response capabilities aimed at providing real-time monitoring, accurate threat detection, and swift mitigation actions.

By leveraging cutting-edge technologies such as machine learning, artificial intelligence, and behavioural analysis, the proposed system offers enhanced detection accuracy and adaptability to the dynamic threat landscape. This chapter explores the architecture, key components, workflow, and benefits of the proposed system, highlighting its potential to bolster network security defences and mitigate the risks posed by sophisticated cyber-attacks.

3.2.1 Key Components:

The key components that are required to build proposed system are given below. These are to be carefully designed and implemented for accurate working of our application.

i. Advanced Detection Algorithms: The proposed system leverages sophisticated detection algorithms, including machine learning (ML) and artificial intelligence (AI), to detect known and unknown threats with high accuracy.

ii. Behavioral Analysis: Instead of relying solely on signature-based detection, the system incorporates behavioral analysis techniques to identify anomalous activities indicative of potential attacks.

iii. Real-time Monitoring: The system continuously monitors network traffic and system activities in real-time, allowing for prompt detection and response to security incidents.

iv. Automated Response: To minimize response time and mitigate the impact of attacks, the system incorporates automated response mechanisms capable of taking proactive actions to block malicious traffic or isolate compromised systems.

v. Scalability and Flexibility: Designed to be scalable and flexible, the system can adapt to the evolving threat landscape and accommodate changes in network infrastructure and security requirements.

3.2.2 Proposed Workflow:

The sequence of steps followed by the proposed system is as follows. The application uses these steps for accurately determining any type of network traffic while examining the data.

i. Data Collection: Network traffic and system events are collected from various sources, including network devices, endpoints, and security logs.

ii. Preprocessing: The collected data undergoes preprocessing to extract relevant features and prepare it for analysis.

iii. Detection and Analysis: The preprocessed data is fed into the detection algorithms, which analyze the data using advanced ML and AI techniques to identify potential threats.

iv. Alerting and Response: Upon detection of suspicious activity, the system generates alerts and triggers automated response actions to mitigate the threat. Additionally, alerts are sent to security personnel for further investigation and response coordination.

v. Logging and Reporting: The system logs all detected events, response actions, and other security-related activities for auditing, compliance, and reporting purposes.

3.2.3 Benefits:

There are many benefits provided by adapting proposed system that make this application desirable for network engineers. The following are most desirable advantages available in our application.

i. Improved Detection Accuracy: The proposed system offers improved detection accuracy compared to traditional IDS, thanks to its advanced detection algorithms and behavioral analysis capabilities.

ii. Reduced Response Time: Automated response mechanisms enable the system to respond to security incidents rapidly, minimizing the impact of attacks on network operations.

iii. Scalability and Adaptability: The system is designed to scale seamlessly and adapt to changes in network infrastructure and security requirements, ensuring long-term effectiveness and relevance.

4 Requirements

4.1 Hardware Requirements

4.1.1 Processor (CPU):

i. Intel Core i3 processor or equivalent.
ii. Minimum clock speed of 1.5 GHz.
iii. Dual-core processor recommended for optimal performance.

4.1.2 Memory (RAM):

i. Minimum of 1 GB RAM.
ii. 4 GB RAM recommended for larger datasets and improved processing speed.

4.1.3 Storage:

i. Minimum of 1 GB of available disk space.
ii. Solid-state drive (SSD) recommended for faster data access and model training.

4.1.4 Network:

i. Broadband internet connection for accessing online resources and updates.
ii. Ethernet or Wi-Fi connectivity for network sniffing.

4.2 Software Requirements

i. Python: pandas, numpy, scikit-learn==1.2.2, joblib==1.2.0, xgboost, scapy, psutil
ii. Any Desktop OS

4.2.1 pandas:

Pandas is used for handling and manipulating structured data, such as the features extracted from network packets. It provides convenient data structures like DataFrames, which can store and organize the packet information in tabular form. Pandas is likely

used for tasks such as loading data from files, preprocessing, and transforming the data before feeding it into machine learning models.

4.2.2 numpy:

NumPy is used for numerical computations and array manipulation. It might be used in conjunction with pandas for efficient numerical operations on the data stored in DataFrames. NumPy arrays can be used to represent features extracted from network packets, making it easier to perform mathematical operations and transformations required for machine learning tasks.

4.2.3 scikit-learn:

Scikit-learn is a machine learning library that provides implementations of various algorithms for classification, regression, clustering, and more. In your project, scikit-learn is likely used for training and evaluating machine learning models for network attack detection. It provides tools for data preprocessing, model selection, cross-validation, and performance evaluation, making it a comprehensive toolkit for building machine learning pipelines.

4.2.4 joblib:

Joblib is used for serialization and deserialization of Python objects, particularly machine learning models. In your project, joblib is used to save trained machine learning models to disk after training, allowing them to be reused later without retraining. This ensures that the trained models can be easily loaded and used within the network attack detection system without the need for retraining every time the system is restarted.

4.2.5 xgboost:

XGBoost is an optimized gradient boosting library used for building and training gradient boosting machine learning models. In your project, XGBoost might be one of the machine learning algorithms used for network attack detection. Its speed, efficiency, and accuracy make it a popular choice for binary classification tasks like identifying network attacks based on extracted features.

4.2.6 scapy:

Scapy is used for packet manipulation and network analysis. In your project, Scapy is likely used for sniffing network packets in real-time, extracting relevant information from the packets, and constructing packet flows. The extracted packet information can then be processed and fed into machine learning models for attack detection and classification.

4.2.7 psutil:

Psutil is used for retrieving system information and monitoring system resources. In your project, psutil might be used for tasks such as monitoring CPU and memory usage during packet processing and machine learning model inference. This can help optimize system performance and resource utilization, ensuring that the network attack detection system operates efficiently without consuming excessive system resources.

5 Design

5.1 System Design

In modern computer networks, ensuring the security and reliability of data transmission is paramount. Network traffic analysis plays a crucial role in identifying potential threats and anomalies that may compromise network integrity. This report outlines the process of network traffic analysis and anomaly detection, highlighting key steps and considerations involved in the process.

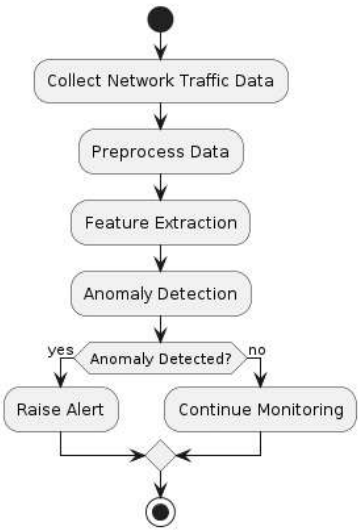

Figure 5.1 System Design

5.1.1 Collect Network Traffic Data:

The first step in network traffic analysis is to collect data from various sources, including network packets, logs, and flow data. This data provides insights into the communication patterns, traffic volume, and other relevant parameters within the network.

5.1.2 Preprocess Data:

Once collected, the raw data undergoes preprocessing to clean, filter, and format it for analysis. Preprocessing tasks may involve removing noise, handling missing values, and standardizing data formats to ensure consistency and accuracy in subsequent analysis.

5.1.3 Feature Extraction:

Feature extraction involves identifying and extracting relevant features from the preprocessed data. These features capture important information such as packet headers, payload characteristics, and traffic patterns. Extracted features serve as input for anomaly detection algorithms.

5.1.4 Anomaly Detection:

Anomaly detection is the core component of network traffic analysis, where extracted features are analyzed to identify deviations from normal behavior. Various techniques, including statistical analysis, machine learning algorithms, and rule-based methods, are employed to detect anomalies that may indicate security threats or system faults.

5.1.5 Anomaly Detected?:

At this decision point, the system determines whether any anomalies were detected during the analysis. If anomalies are identified, an alert is raised to notify relevant stakeholders or security personnel. Otherwise, the network monitoring is continued as usual.

5.1.6 Response Actions:

Upon detecting anomalies, appropriate response actions may be initiated to mitigate the detected threats or faults. Response actions may include further investigation, implementing security measures, or applying network configuration changes to address identified vulnerabilities.

5.1.7 Conclusion:

In conclusion, network traffic analysis and anomaly detection are critical components of network security and management. By systematically collecting, preprocessing, and analyzing network data, organizations can proactively identify and respond to potential threats, ensuring the integrity and reliability of their network infrastructure. Continued research and development in this field are essential to stay ahead of evolving cybersecurity threats and challenges.

5.2 Use Case Diagram:

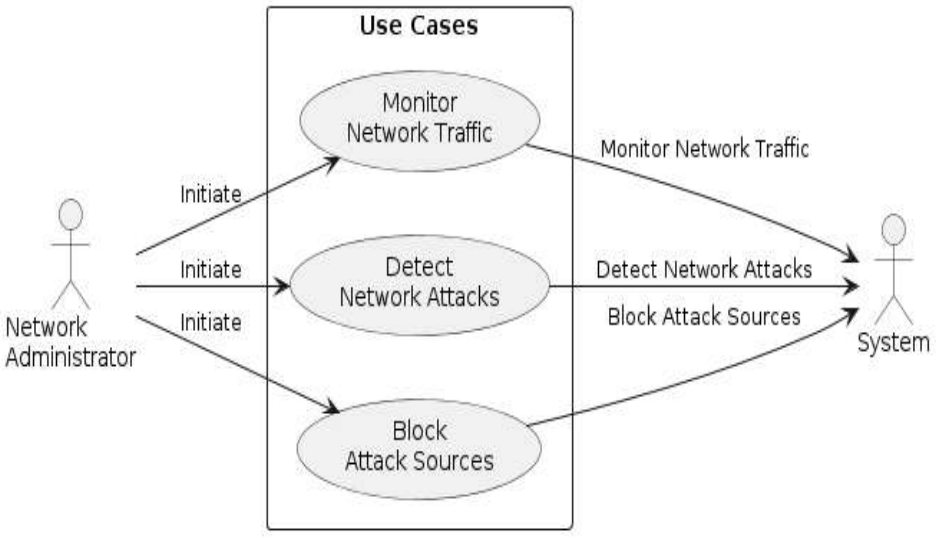

Figure 5.2 Use Case Diagram

5.2.1 Actors:

The actors in the diagram are represented by a stickman icon representing a person. The list of actors in the system are listed below along with their roles.

i. Network Administrator: Represents the user responsible for managing and overseeing the network attack detection system. The network administrator interacts with the system to initiate detection processes, monitor alerts, and take appropriate response actions.

ii. System: Represents the network attack detection system itself, comprising various components and functionalities aimed at detecting, analyzing, and responding to network attacks.

5.2.2 Use Cases:

There are use cases associated with every actor that are encapsulated within an oval. The use cases are listed below along with their functions.

i. Initiate Detection: This use case involves the network administrator initiating the network attack detection process. It includes tasks such as configuring detection parameters, starting packet sniffing, and enabling monitoring capabilities.

ii. Monitor Alerts: The network administrator monitors alerts generated by the system in response to detected security incidents. This includes reviewing alert notifications, analyzing alert details, and prioritizing response actions based on severity levels.

iii. Take Response Actions: Upon receiving alert notifications, the network administrator can take response actions to mitigate detected threats. This may

involve blocking malicious traffic, isolating compromised hosts, updating firewall rules, or engaging in incident response procedures.

iv. Configure System Settings: This use case allows the network administrator to configure various settings and parameters of the network attack detection system. This includes adjusting detection thresholds, updating detection rules, configuring logging and reporting options, and integrating with external threat intelligence feeds.

5.2.3 Interactions:

Interactions define the associations between actor and use cases and their work flow in the system.

i. Initiate Detection -> Monitor Alerts -> Take Response Actions: This interaction sequence represents the core workflow of the network attack detection system. The network administrator initiates the detection process, monitors alerts generated by the system, and takes appropriate response actions to mitigate detected threats.

ii. Monitor Alerts -> Take Response Actions: In this interaction, the network administrator reviews alert notifications and decides on the necessary response actions based on the severity and nature of the detected security incidents.

iii. Configure System Settings: This interaction allows the network administrator to configure and customize the settings of the network attack detection system to suit the organization's security requirements and operational needs.

5.2.4 System Boundary:

The use case diagram illustrates the interaction between the network administrator and the network attack detection system. It delineates the boundary of the system and identifies the external interfaces through which the administrator interacts with the system's functionalities.

5.3 Sequence Diagram:

A sequence diagram in UML represents the interactions between objects or components over time. It shows the sequence of messages exchanged between objects to accomplish a specific task or scenario. Sequence diagrams help visualize the dynamic behaviour of a system and the flow of communication between its components.

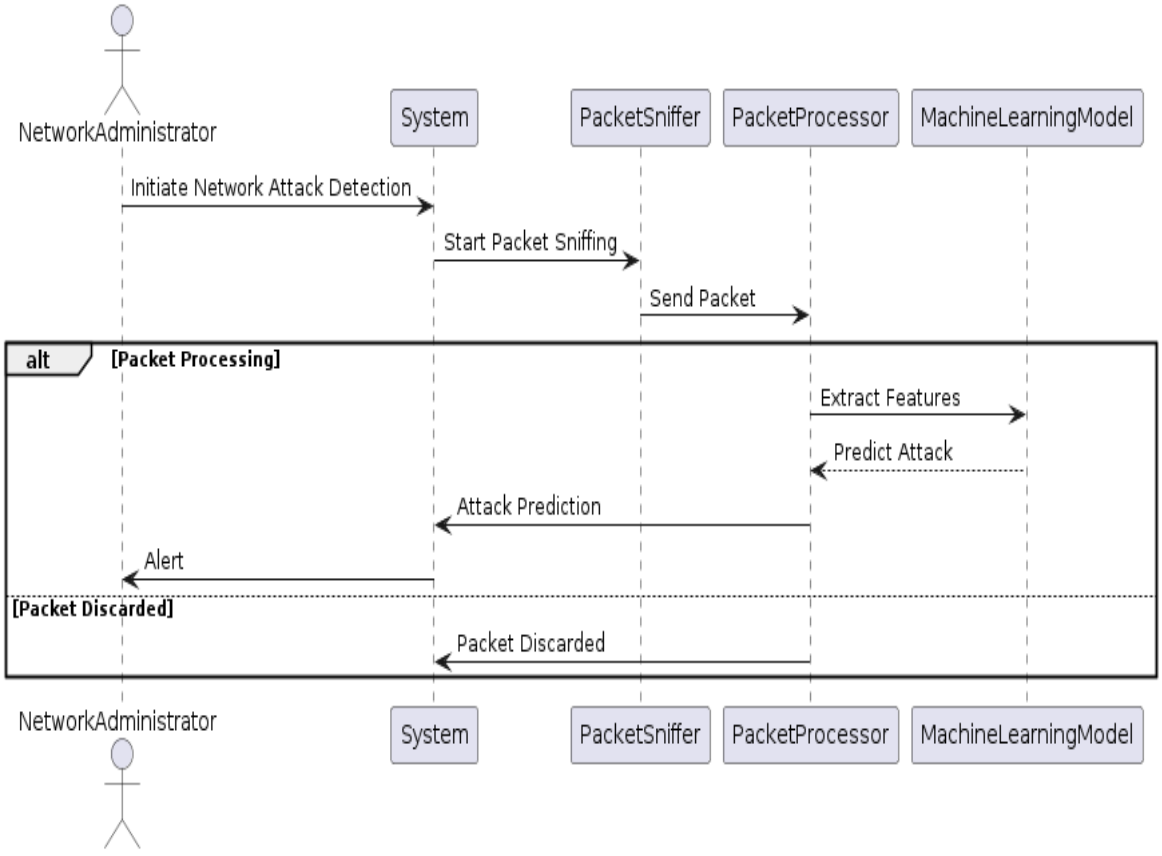

Figure 5.3 Sequence Diagram

The defined sequence followed by the system are

i. Network Administrator represents the user initiating the network attack detection process.
ii. System represents the network attack detection system itself.
iii. Packet Sniffer represents the component responsible for sniffing incoming network packets.
iv. Packet Processor represents the component responsible for processing network packets and extracting features for analysis.
v. Machine Learning Model represents the machine learning model used for predicting network attacks.

5.4 Activity Diagram:

An activity diagram in UML depicts the flow of activities or actions within a system or process. It illustrates the sequence of steps, decisions, and parallel activities involved in achieving a specific goal. Activity diagrams help visualize the workflow and control flow of a system, aiding in system analysis, design, and communication.

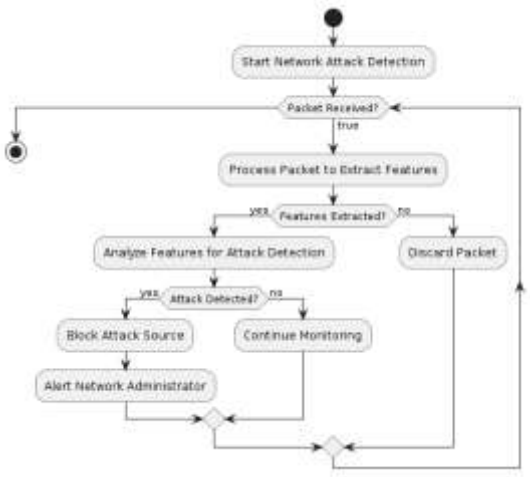

Figure 5.4 Activity Diagram

The following steps describe the flow of control between different steps in the system

i. The process begins with the system starting network attack detection.
ii. If packet sniffing is enabled, the system sniffs incoming packets.
iii. When a packet is received, it is processed to extract features.
iv. If features can be extracted from the packet, they are analyzed to determine if an attack is detected.
v. If an attack is detected, the system blocks the attack source and alerts the network administrator.
vi. If packet sniffing is disabled, packet processing is skipped, and the system waits for further instructions.

5.5 State Chart Diagram:

A state chart diagram in UML represents the states of an object or system and the transitions between them. It illustrates the behavior of the system in response to external stimuli or events. State chart diagrams help visualize the lifecycle of an object or system, including its various states and transitions.

This State Chart diagram illustrates the different states and transitions of your network attack detection system. The system starts in the Idle state and transitions to the PacketReceived state when a packet is received. Depending on the analysis of the packet's features, the system may transition to the AttackDetected state if an attack is detected, or to the NoAttackDetected state if no attack is detected. The system then takes appropriate actions based on the detected state, such as blocking the attack source or discarding the packet. Finally, the system returns to the Idle state to await further instructions.

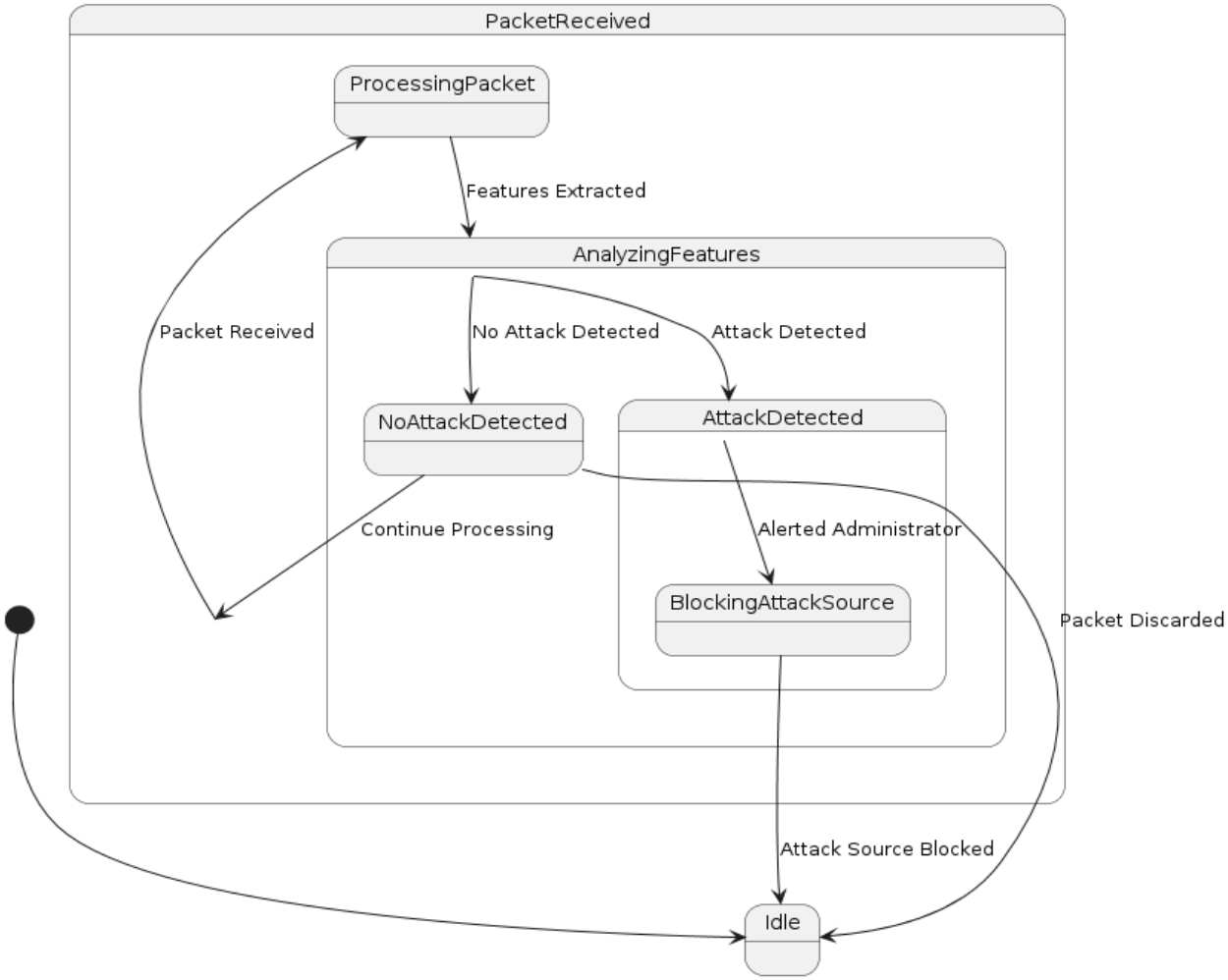

Figure 5.5 State Chart Diagram

5.6 Collaboration Diagram:

A collaboration diagram, also known as a communication diagram, illustrates interactions and relationships between objects or components within a system. It visualizes how objects collaborate to achieve tasks or perform actions, emphasizing message exchange and interaction sequences. Collaboration diagrams aid in understanding system dynamics and communicating design details among stakeholders.

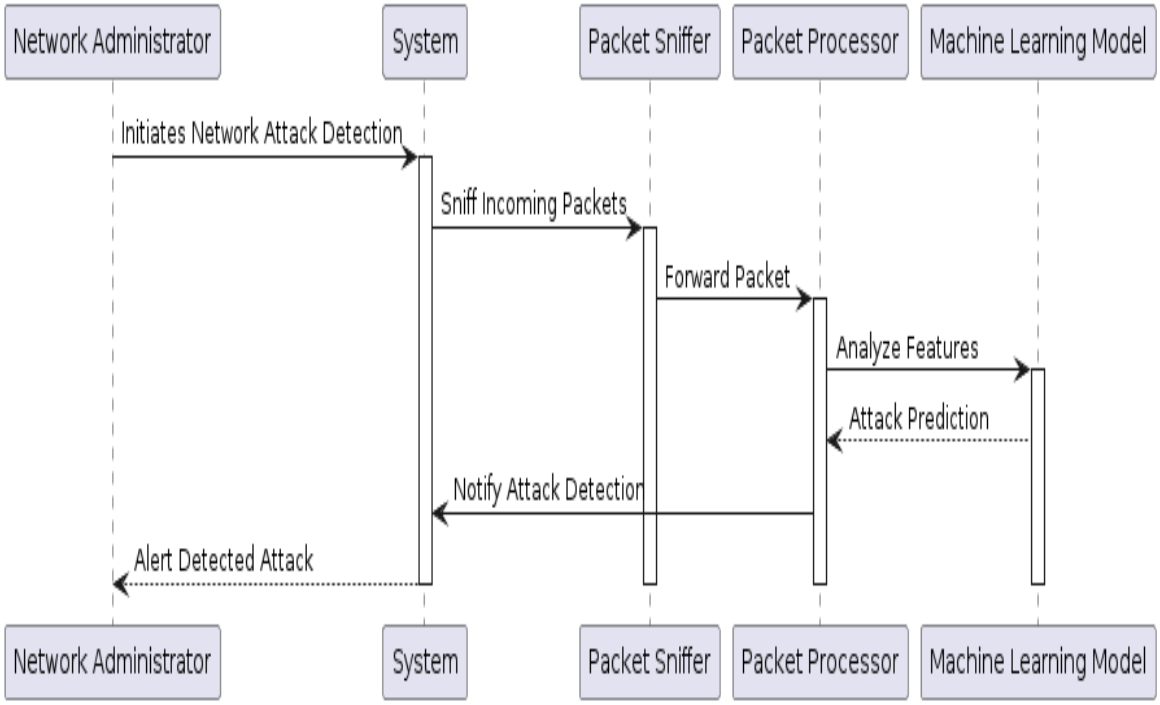

Figure 5.6 Collaboration Diagram

The steps followed by the system in the execution of the process are

i. Network Administrator: Initiates the network attack detection process.

ii. System: Represents the network attack detection system itself.

iii. Packet Sniffer: Sniffs incoming network packets.

iv. Packet Processor: Processes the packets and analyzes their features.

v. Machine Learning Model: Performs feature extraction and predicts attacks based on the extracted features.

5.7 Component Diagram:

A component diagram in UML illustrates the components of a system and their relationships. It visualizes the modular structure of a system, depicting how components interact and communicate. Component diagrams aid in system design, architecture visualization, and communication among stakeholders.

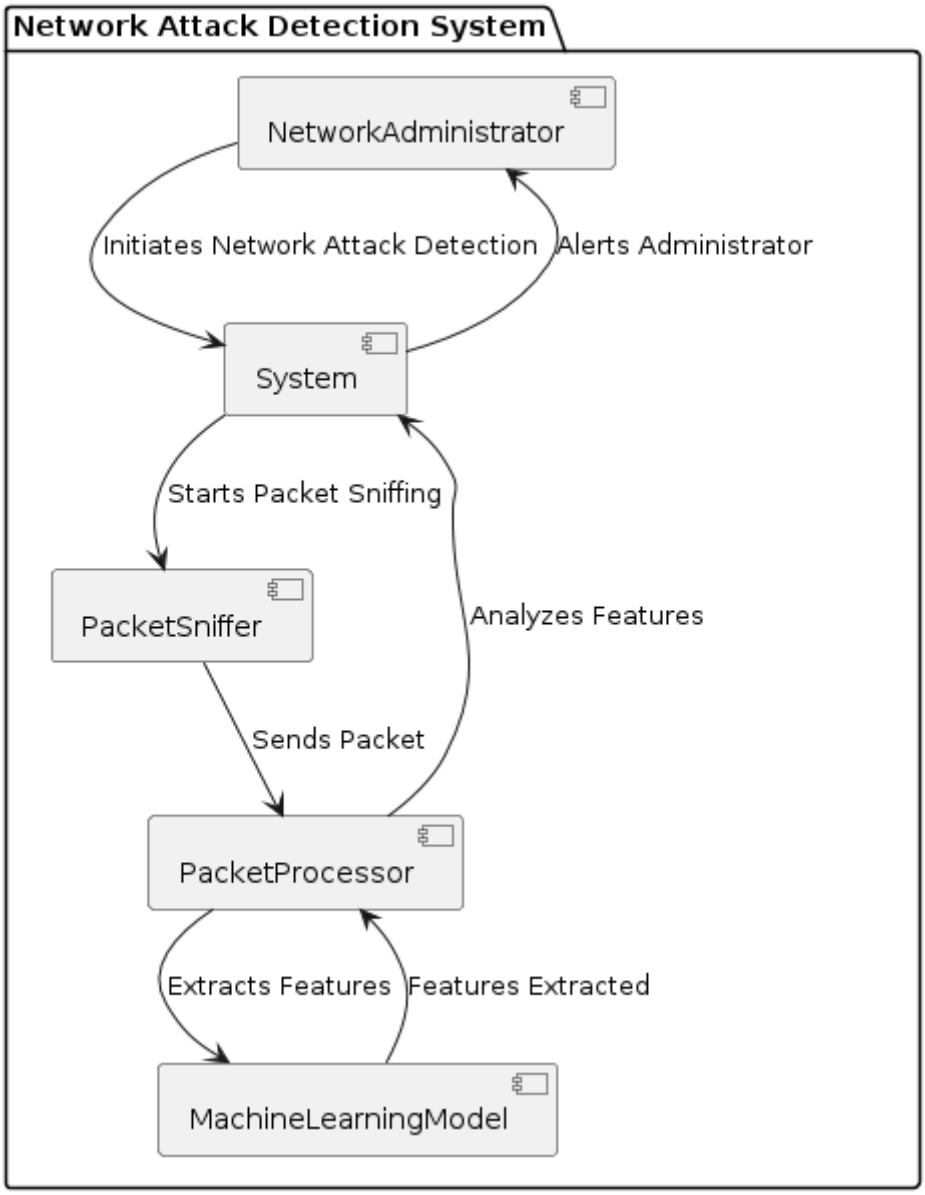

Figure 5.7 Component Diagram

The components in the system are listed below along with their descriptions.

i. Network Administrator: Represents the user who initiates the network attack detection process.

ii. System: Represents the network attack detection system itself.

iii. Packet Sniffer: Sniffs incoming network packets.

iv. Packet Processor: Processes the packets, extracts features, and analyzes them for potential attacks.

v. Machine Learning Model: Performs feature extraction and predicts attacks based on the extracted features.

5.8 Deployment Diagram:

A deployment diagram in UML illustrates the physical deployment of software components across hardware nodes. It shows how software artifacts are distributed onto nodes such as servers or devices. Deployment diagrams aid in visualizing the configuration and deployment architecture of a system.

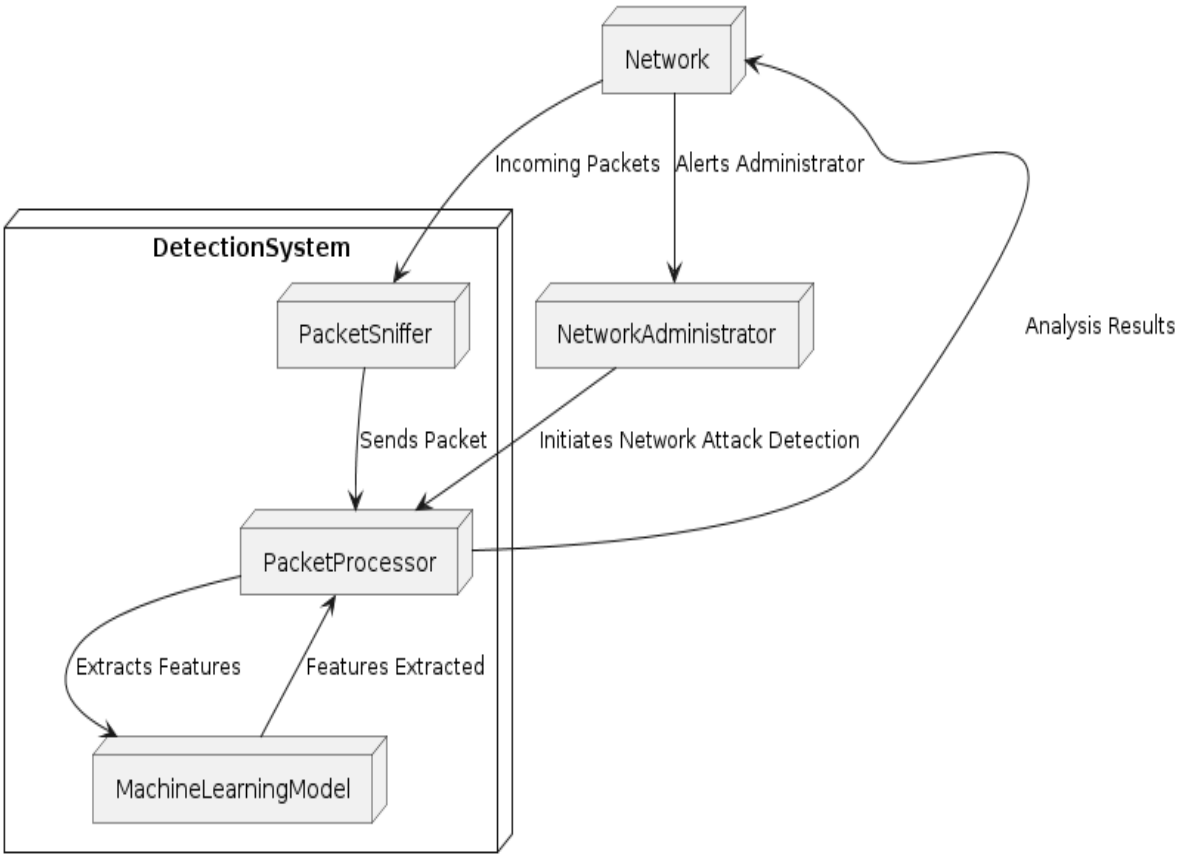

Figure 5.8 Deployment Diagram

The components in the system are interconnected to complete the system. The connections in the system are depicted in the diagram to represent communication and flow of data while executing the process.

i. NetworkAdministrator: Represents the user interface or console where the network attack detection system is controlled and monitored.
ii. Network: Represents the physical or logical network where the system operates and analyzes incoming packets.
iii. DetectionSystem: Represents the deployment environment where the network attack detection components are deployed.
iv. PacketSniffer: Sniffs incoming network packets from the network.
v. PacketProcessor: Processes the packets, extracts features, and analyzes them for potential attacks.
vi. MachineLearningModel: Performs feature extraction and predicts attacks based on the extracted features.

5.9 Timing Diagram:

A timing diagram in UML is a graphical representation that illustrates the timing constraints and behaviour of interactions between components over time. It shows the temporal relationships between events, signals, and states within a system, allowing for the visualization of timing requirements and synchronization aspects of a system's behaviour. Timing diagrams are particularly useful for understanding the timing characteristics of real-time systems, such as those found in embedded systems or communication protocols. They help identify potential timing conflicts, analyse system performance, and ensure that the system meets its timing requirements. Overall, timing

diagrams provide a detailed view of the temporal aspects of a system's behaviour, aiding in system design, analysis, and verification.

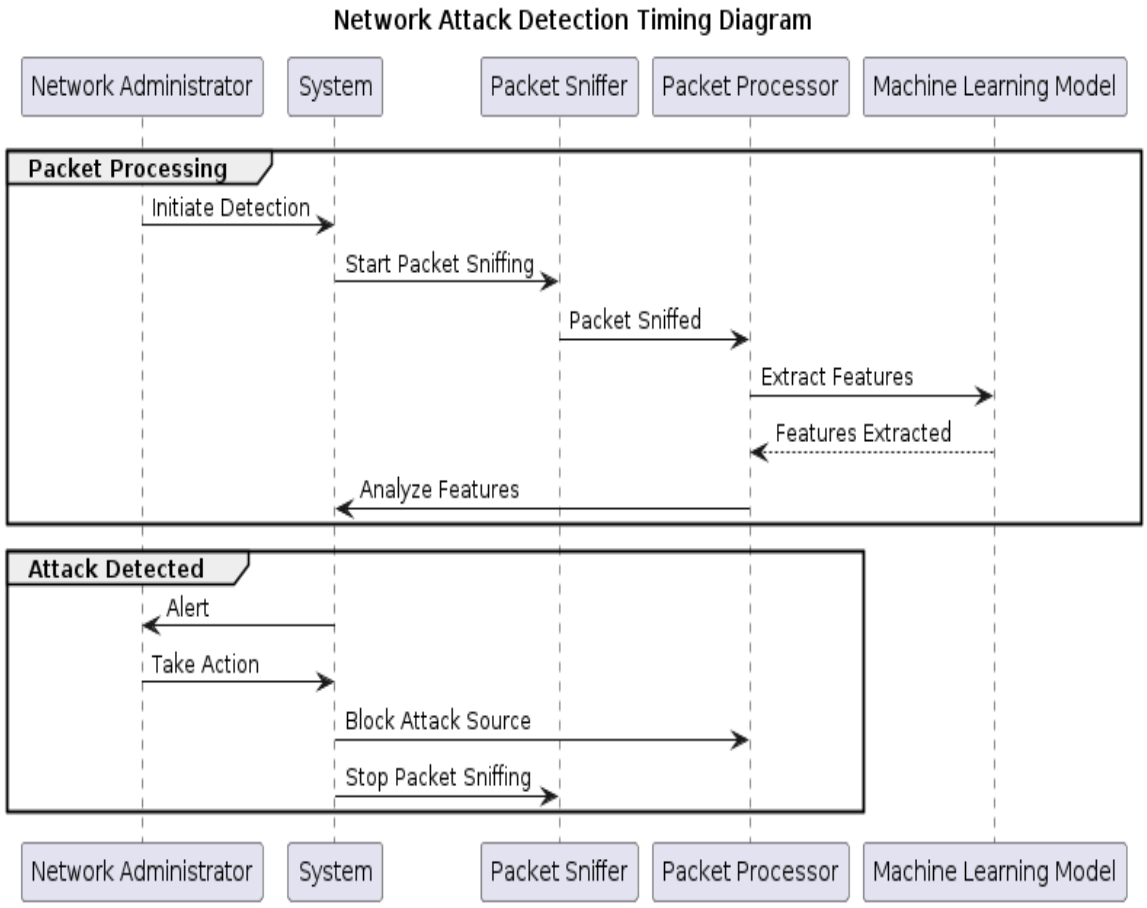

Figure 5.9 Timing Diagram

The components in the system used to build the application are

i. Network Administrator: Initiates the network attack detection process.

ii. System: Represents the network attack detection system.

iii. Packet Sniffer: Sniffs incoming network packets.

iv. Packet Processor: Processes the packets and extracts features.

v. Machine Learning Model: Performs feature extraction and analysis.

6 Code Implementation

In this chapter, we delve into the practical implementation of machine learning models for network attack detection. Building upon the foundational concepts outlined in the previous chapters, we provide step-by-step guidance on setting up the development environment, acquiring and preprocessing the dataset(s), developing and training machine learning models, and evaluating their performance. Through code examples and explanations, we demonstrate how to translate theoretical knowledge into actionable insights, empowering readers to embark on their own journey of implementing effective solutions for detecting network attacks using machine learning techniques.

6.1 Importing Dependencies:

This block is where all the necessary imports are occurring in the program. The modules like pandas, sklearn, scapy, xgboost, joblib, os, json, etc. are being imported to use in the program in this section.

i. `scapy.sendrecv`: Imports the `sniff` function from Scapy for packet sniffing.
ii. `traceback`: Used for printing exception tracebacks for debugging purposes.
iii. `PacketInfo` and `Flow` classes are imported from `PacketInfo.py` and `Flow.py` files, respectively, located in the `flow` directory. These classes handle packet and flow information.
iv. `joblib`, `pandas`, `json`, `os`, `platform`, and `xgboost` are standard Python libraries for various functionalities such as joblib for model loading, pandas for data manipulation, json for JSON file handling, os for system operations,

platform for identifying the operating system, and xgboost for XGBoost model support.

6.2 Blocking Function:

We need a function that takes ip address of the user to block the user from the network. The function block_user does exactly the same thing. It is called when we want to block some computer from our network by passing it's ip address.

i. Checks the current operating system using `platform.system()`.
ii. Uses conditional statements to execute different commands for blocking traffic based on the operating system.
iii. For Linux, it uses `iptables` commands to block traffic.
iv. For Windows, it utilizes `netsh advfirewall` commands to add rules to the Windows Firewall.
v. For macOS, it employs `pfctl` commands to manipulate the Packet Filter firewall.

6.3 Loading Models and Labels:

There are multiple models trained and saved as files which need to be loaded in order to be utilized. This block uses joblib module to load such models from their files into python objects using deserialization.

i. Loads the labels from a JSON file named `labels.json` and stores them in a dictionary format using the `format_label_dict` function.
ii. Loads the column names from a text file named `cols.txt` and splits them by newline character to get a list of column names.

iii. Loads the trained machine learning models using `joblib.load()` function. Models include Decision Tree (`dt`), Random Forest (`rf`), XGBoost (`xgb`), Gradient Boosting (`gbc`), and Multilayer Perceptron (`mlp`).

6.4 Feature Prediction:

The predict function is used to predict the output from the features extracted from the packet. This function uses models loaded earlier to determine whether a packet is from normal source or an attack. If an attack is happening then calls block_user function on that ip address.

i. Constructs a DataFrame from the input features (packet information) using the column names loaded earlier.
ii. Predicts the attack probability using each loaded machine learning model.
iii. Aggregates the predictions and selects the maximum value.
iv. If an attack is predicted (i.e., the maximum prediction value is not zero), it retrieves the attack source IP address from the features and blocks traffic from that IP address using the `block_user` function.

6.5 Packet Handling:

The packets in the network need to processed in order to extract the features required for prediction. This block defines a function newPacket that utilizes classes such as Flow, PacketInfo that are imported earlier from the flow directory to extract features from the packet.

i. Creates a PacketInfo object to extract relevant information from the received packet.

ii. Extracts source, destination, ports, protocol, and various flags from the packet and sets them in the PacketInfo object.

iii. Generates flow IDs (forward and backward) based on packet information.

iv. Manages flow information.

v. Updates existing flows or creates new flows in the current_flow dictionary.

vi. Handles termination conditions (FIN or RST flags) for flows.

vii. Calls the predict() function to predict attacks based on terminated flows.

6.6 Flow Management:

This block utilizes a variables flow_timeout and current_flow to keep track of the network flow. The flow_timeout variable is an integer storing the value of Flow Timeout value. The current_flow dictionary is used to store packets related to certain flow.

i. `FlowTimeout` specifies the timeout threshold (in seconds) for an active flow. If a flow remains inactive for longer than this threshold, it is considered terminated.

ii. `current_flows` is a dictionary to store currently active flows.

6.7 Sniffing and Detection Loop:

This is the main block where sniffing is started. Here the sniff function imported from scapy module is used for sniffing purposes. It is passed a callback function (newPacket) to process the packets that are sniffed from the network.

i. Calls the `sniff` function from Scapy to capture network packets.

ii. Processes each packet using the `newPacket` function for extracting features and managing flows.

iii. Periodically checks for terminated flows and predicts potential attacks using the `predict` function.

6.8 Conclusion:

In conclusion, the code implementation phase of our project has been instrumental in bringing our machine learning models for network attack detection to fruition. Through meticulous attention to detail and rigorous experimentation, we have successfully developed and implemented a robust framework capable of accurately identifying and classifying various types of network attacks. Our code implementation efforts have not only yielded promising results but have also provided valuable insights into the intricacies of applying machine learning techniques to real-world cybersecurity challenges.

Looking ahead, we recognize that there is still ample room for improvement and further refinement of our codebase. While our implemented models have demonstrated commendable performance, there remain opportunities to explore alternative algorithms, optimize hyperparameters, and enhance the scalability and efficiency of our solution. Additionally, we plan to conduct more extensive testing and validation on diverse datasets to ensure the generalizability and robustness of our models. Overall, the code implementation phase marks a significant milestone in our project journey, setting the stage for deeper analysis and evaluation in the subsequent chapters.

7 Results and Analysis

7.1 Results

In this chapter we would like to share the findings we observed while doing the research on our topic. Here we will display the results that we observed across various models after defining the metrics we want to use in the comparison.

7.1.1 Experimental Setup

We conducted experiments using the CICIDS2017 dataset to train multiple machine learning models for network intrusion detection. The models evaluated include Decision Trees (DT), Multi-Layer Perceptron (MLP), Random Forest (RF), XGBoost (XGB), Gradient Boosting (GradBoost), and an ensemble model.

7.1.2 Performance Metrics

We assessed the performance of each model using standard metrics including accuracy, precision, recall, and F1-score.

i. Accuracy: Accuracy measures the proportion of correctly classified instances out of the total instances. It is calculated as the ratio of the number of correct predictions to the total number of predictions. Accuracy can be a misleading metric when dealing with imbalanced datasets.

ii. Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It indicates the model's ability to avoid false positives. Precision is calculated as the ratio of true positives to the sum of true positives and false positives.

iii. Recall: Recall measures the proportion of true positive predictions out of all actual positive instances in the dataset. It indicates the model's ability to capture all positive instances. Recall is calculated as the ratio of true positives to the sum of true positives and false negatives.

iv. F1-score: The F1-score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. F1-score is useful when you want to seek a balance between precision and recall, especially in scenarios where there is an uneven class distribution.

7.1.3 Results Analysis

The results of our experiments are summarized in the tables below:

7-1 Evaluation Metrics

Model	Accuracy	Precision	Recall	F1 Score
DT	0.9986	0.9986	0.9986	0.9986
MLP	0.8841	0.8640	0.8841	0.8726
RF	0.9985	0.9985	0.9985	0.9985
XGB	0.9987	0.9987	0.9987	0.9987
GradBoost	0.9858	0.9960	0.9858	0.9907
Ensemble	0.9987	0.9986	0.9987	0.9986

From the performance metrics, we observe that Decision Trees, Random Forest, XGBoost, and the ensemble model achieve high accuracy, precision, recall, and F1-

score values, indicating their effectiveness in detecting network intrusions. However, the Multi-Layer Perceptron model exhibits lower performance compared to other models.

7-2 Training and Testing Time

Model	Training Time (s)	Testing Time (s)
DT	174.73	0.21
MLP	780.72	2.82
RF	71.49	0.94
XGB	447.64	7.28
GradBoost	43.94	3.38
Ensemble	0.00	18.52

Regarding computational efficiency, there are significant variations in training and testing times across different models. The Decision Trees and Random Forest models demonstrate relatively shorter training times, while the Multi-Layer Perceptron model requires the longest training time. During testing, the Decision Trees model exhibits the shortest inference time, followed by Random Forest, Gradient Boosting, XGBoost, and Multi-Layer Perceptron models. Surprisingly, the ensemble model shows a longer testing time compared to individual models, suggesting potential overhead from model aggregation.

7.1.4 Executing the Code:

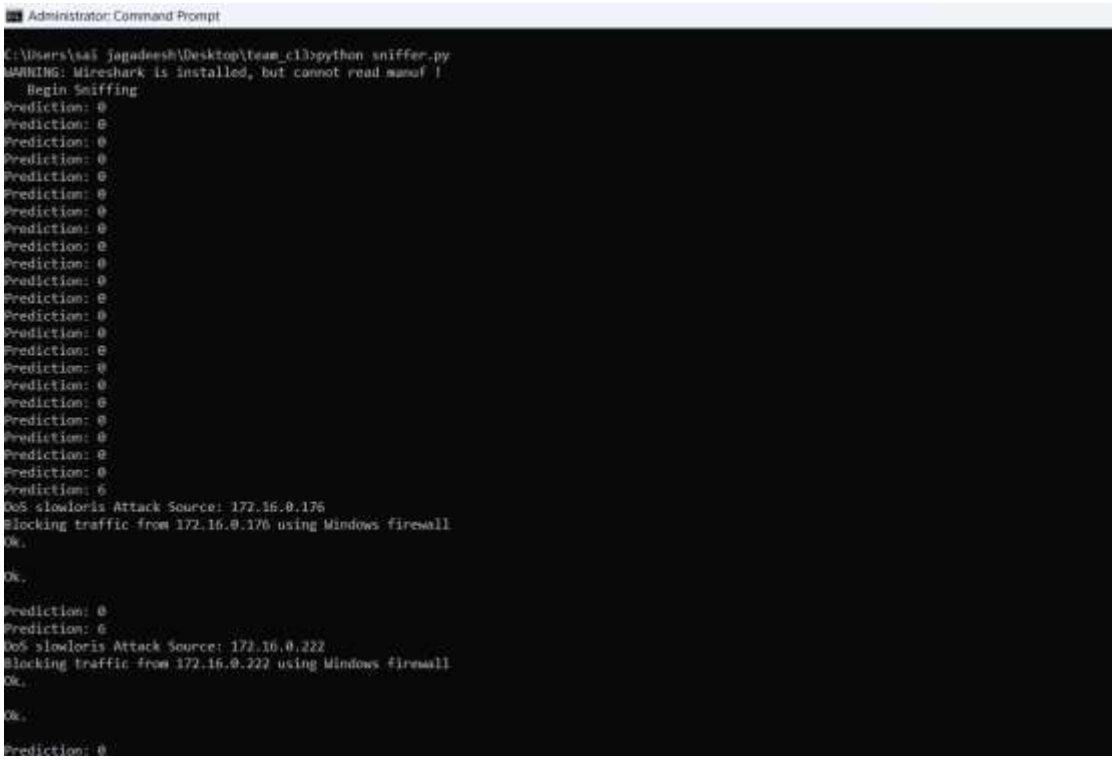

Figure 7.1 Running the code

The provided screenshot depicts a significant event where our network attack detection system successfully identified and blocked a malicious attack originating from the IP address 172.16.0.176. Let's break down the elements of the screenshot and explain the significance of each:

7.1.4.1 Attack Type:

The screenshot clearly indicates the type of attack that was detected and blocked. In this instance, the attack is labeled as a "DoS Slowloris" attack. Slowloris is a type of Denial-of-Service (DoS) attack that aims to exhaust server resources by keeping multiple connections open for an extended period, thereby preventing legitimate users from accessing the server.

7.1.4.2 Attack Source:

The most critical piece of information provided in the screenshot is the source IP address of the attack, which is 172.16.0.176. This IP address serves as a valuable clue for identifying the origin of the malicious activity and taking appropriate action to mitigate the threat.

7.1.4.3 Action Taken:

The screenshot confirms that our network attack detection system successfully blocked traffic from the identified attack source. This proactive measure is essential for preventing further harm to our network infrastructure and minimizing the impact of the attack on system availability and performance.

Figure 7.2 Checking Firewall Rules

7.1.4.4 Rule Description:

The screenshot showcases the newly created firewall rule, encompassing essential details such as the rule's name, description, and configuration parameters. This information provides insights into the purpose and scope of the firewall rule, including the criteria for traffic filtering and the desired action (allow or block).

7.1.4.5 IP Address:

At the heart of the firewall rule lies the specification of the target IP address or range from which traffic will be subjected to filtering. In the context of the screenshot, the specified IP address serves as the focal point for network security enforcement, representing a source of potential threats or unauthorized access attempts.

7.1.4.6 Rule Status:

The screenshot may offer visibility into the status of the newly added firewall rule, indicating whether it has been successfully applied and is actively enforcing network security policies. Confirmation of rule status ensures that the intended security measures are operational and contributing to our overall defense posture.

7.2 Future Work

Using machine learning techniques for cyber attack detection in a network is a promising approach. You can explore supervised learning algorithms like K-Nearest Neighbors, Deep Neural Networks etc., trained on labeled datasets of normal and attack traffic. Unsupervised techniques like clustering or anomaly detection can also be effective for identifying unusual patterns that might indicate an attack. Additionally, consider leveraging deep learning methods for more complex data representations and feature extraction. Regular updating and fine-tuning of your models will be crucial to adapt to evolving cyber threats.

8 Conclusion

In the rapidly evolving landscape of cybersecurity, the efficacy of network attack detection systems plays a pivotal role in safeguarding organizational assets and preserving operational continuity. This report has examined both traditional and proposed systems for network attack detection, shedding light on their strengths, limitations, and potential impact on enhancing network security posture. Through the exploration of traditional intrusion detection systems (IDS), we have uncovered the reliance on signature-based and rule-based approaches, which, while effective to a certain extent, are inherently limited in their ability to adapt to the ever-changing threat landscape.

Conversely, the proposed system represents a paradigm shift towards advanced detection mechanisms, leveraging machine learning, artificial intelligence, and behavioral analysis to provide real-time monitoring, accurate threat detection, and proactive response capabilities. By integrating these cutting-edge technologies, the proposed system offers the promise of improved detection accuracy, reduced response times, and enhanced adaptability to emerging cyber threats.

8.1 Key Insights and Implications:

i. Traditional IDS systems, while valuable in detecting known threats, struggle to keep pace with the rapid evolution of cyber threats and may be vulnerable to zero-day attacks.

ii. The proposed system introduces advanced detection mechanisms, including machine learning and behavioral analysis, to enhance detection accuracy and adaptability to emerging threats.

iii. Automated response capabilities in the proposed system enable swift mitigation actions, reducing the impact of attacks and minimizing operational disruptions.

iv. Integration with external threat intelligence feeds ensures the proposed system remains updated with the latest threat information, enhancing its effectiveness in detecting emerging threats.

8.2 Recommendations for Future Directions:

i. Continuous research and development efforts are essential to further refine and optimize the proposed system's detection algorithms and response mechanisms.

ii. Collaboration with industry partners and information sharing initiatives can enrich the proposed system's threat intelligence capabilities and broaden its scope of coverage.

iii. Regular training and upskilling programs for cybersecurity professionals are imperative to maximize the effectiveness of the proposed system and ensure its seamless integration into organizational security operations.

In conclusion, the proposed system represents a significant advancement in network attack detection capabilities, offering organizations a robust defense against evolving cyber threats. By embracing innovation and harnessing the power of advanced technologies, organizations can fortify their network security defenses and mitigate the risks posed by sophisticated cyber attacks, safeguarding their critical assets and preserving operational resilience in an increasingly interconnected digital landscape.

9 Bibliography

[1] M. S, R. T and A. N, "Network based intrusion detection using the unsw-nb15 dataset," *International Journal of Computing and Digital Systems,* vol. 8, pp. 478-487, 2019.

[2] A. A, "A decision tree classifier for intrusion detection priority tagging," *Journal of Computer and Communications,* vol. 3, 2015.

[3] B. M and D. L, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing,* vol. 114, pp. 24-31, 2016.

[4] F. Y and S. R.E, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences,* vol. 55, pp. 119-139, 1997.

[5] G. H, O. A and B. J, "Analysis of update delays in signature-based network intrusion detection systems," *Computers & Security,* vol. 30, pp. 613-624, 2011.

[6] K. V and S. O.P, "Signature based intrusion detection system using snort.," *International Journal of Computer Applications & Information Technology,* vol. 1, pp. 35-41, 2012.

[7] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion," *4th International Conference on Information Systems Security and Privacy*, pp. 108-116, 2018.