PROJECT REPORT
ON

AI FRAMEWORK FOR IDENTIFYING ANOMALOUS NETWORK TRAFFIC IN MIRAI AND BASHLITE IOT BOTNET ATTACKS

ABSTRACT

In recent years, the proliferation of Internet of Things (IoT) devices has significantly increased the volume of network traffic and the complexity of network environments. According to the Global IoT Security Market Report, the number of IoT devices surged from approximately 8.4 billion in 2017 to an estimated 30.9 billion by 2025. This exponential growth has led to a corresponding rise in sophisticated network attacks, with Mirai and BASHLITE botnets being prominent examples. The Mirai botnet, which first emerged in 2016, exploited IoT vulnerabilities to launch large-scale Distributed Denial of Service (DDoS) attacks, while BASHLITE, identified in 2014, has been known for its effective use of compromised devices in various cyber-attacks. Traditional methods for detecting anomalies in network traffic typically involve manual inspection and rule-based systems. These approaches face several challenges, including the high volume of data, the dynamic nature of network traffic, and the evolving tactics of attackers. Manual methods are often Labor-intensive, prone to human error, and struggle to keep pace with sophisticated attack techniques. Rule-based systems, while useful, are limited by their inability to adapt to new and previously unseen attack patterns. Machine Learning (ML) offers a promising alternative for addressing these limitations. By leveraging algorithms capable of learning from historical data and adapting to new patterns, ML models can identify anomalous traffic indicative of Mirai and BASHLITE attacks more effectively. These models can analyze vast amounts of network data in real time, detect subtle deviations from normal behavior, and improve detection accuracy over time. This approach not only enhances the ability to identify emerging threats but also reduces the dependency on manual intervention, leading to more efficient and scalable network security solutions..

INDEX

CHAPTER 1 INTRODUCTION

1.1 History

In the contemporary digital landscape, the proliferation of Internet of Things (IoT) devices has revolutionized various industries, leading to unprecedented growth in network traffic and complexity. According to the Global IoT Security Market Report, the number of IoT devices surged from approximately 8.4 billion in 2017 to an estimated 30.9 billion by 2025. This exponential increase has simultaneously escalated the risk and frequency of sophisticated network attacks, with the Mirai and BASHLITE botnets being particularly notorious. The Mirai botnet, first identified in 2016, exploited vulnerabilities in IoT devices to orchestrate large-scale Distributed Denial of Service (DDoS) attacks, while the BASHLITE botnet, discovered in 2014, has been associated with various cyber-attacks leveraging compromised devices.

Traditional anomaly detection methods in network traffic, which often rely on manual inspection and rulebased systems, are increasingly inadequate in this evolving threat landscape. These conventional approaches face significant challenges due to the vast volume of data, the dynamic nature of network environments, and the continuously evolving tactics of attackers. Manual methods are labour-intensive, prone to human error, and struggle to keep pace with the sophistication of modern cyber threats. Rulebased systems, although useful, lack the flexibility to adapt to new and previously unseen attack patterns.

In response to these limitations, Machine Learning (ML) presents a promising solution. ML algorithms, with their capacity to learn from historical data and adapt to new patterns, offer enhanced capabilities for identifying anomalous traffic indicative of Mirai and BASHLITE botnet activities. By analysing extensive network data in real time, ML models can detect subtle deviations from normal behaviour, thereby improving detection accuracy and response times. This approach not only bolsters the ability to identify emerging threats but also minimizes reliance on manual intervention, paving the way for more efficient and scalable network security solutions.

1.2 Research Motivation

The motivation behind this research stems from the increasing frequency and sophistication of cyberattacks targeting IoT networks. Traditional security measures, such as rule-based systems and manual inspections, are often inadequate in identifying and responding to these evolving threats. The limitations of these methods, coupled with the potential damage that botnet attacks can cause, highlight the necessity for more advanced detection techniques. Machine learning, with its ability to learn from data and identify patterns, offers a promising approach to enhancing network security. This research is driven by the desire to develop an AI framework capable of detecting anomalous network traffic associated with IoT botnets, thereby providing a proactive defense against cyber threats.

1.3 Problem Statement

The proliferation of IoT devices has led to a significant increase in network traffic, making it challenging to distinguish between legitimate and malicious activity. Botnets like Mirai and BASHLITE exploit IoT vulnerabilities to orchestrate large-scale cyber-attacks, often going undetected by traditional security measures. The problem is exacerbated by the dynamic nature of network environments and the evolving tactics of attackers. The primary problem addressed by this research is the development of an AI-driven framework that can accurately identify anomalous network traffic associated with these botnet attacks, providing real-time detection and response capabilities to mitigate potential threats.

1.4 Applications

The proposed AI framework has broad applications across various industries and sectors that rely on IoT networks:

Smart Cities: Ensuring the security of IoT devices in smart city infrastructure, including traffic management systems, surveillance, and public services.

Healthcare: Protecting sensitive medical devices and patient data from cyber-attacks in connected healthcare environments.

Industrial IoT: Securing industrial control systems and manufacturing processes from disruptions caused by botnet attacks.

Energy Sector: Safeguarding smart grids and energy distribution networks from cyber threats.

Telecommunications: Enhancing the security of communication networks that support IoT devices and services.

Home Automation: Providing security for smart home devices, including thermostats, cameras, and voice assistants, against potential intrusions.

CHAPTER 2

LITERATURE SURVEY

Abomhara and Køien examine the various cybersecurity challenges posed by IoT devices. They highlight the inherent vulnerabilities in IoT ecosystems, which stem from inadequate security measures in device design and deployment. Their study categorizes threats into physical attacks, network attacks, and software attacks, detailing how each can compromise IoT systems. The authors emphasize the need for comprehensive security frameworks that incorporate device authentication, secure communication protocols, and regular updates to mitigate these risks. Their findings underscore the importance of adopting a holistic approach to IoT security to protect against increasingly sophisticated cyber threats.

Andrea et al. discuss the security vulnerabilities and challenges associated with IoT in their conference paper. They outline key security concerns, including data privacy, device authentication, and secure communication. The authors propose a layered security model to address these issues, recommending encryption, secure bootstrapping, and continuous monitoring as essential components. Their work also highlights the challenges of implementing such measures due to the resource constraints of many IoT devices. The study calls for collaboration between device manufacturers, network providers, and security experts to develop robust IoT security solutions.

Deogirikar and Vidhate's survey paper explores various security attacks targeting IoT systems. They categorize these attacks into categories such as physical attacks, network attacks, and software attacks, providing examples and discussing their potential impact. The authors emphasize the need for a multilayered defense strategy, incorporating physical security measures, secure communication protocols, and regular software updates. They also discuss the role of machine learning and anomaly detection in identifying and mitigating such attacks. Their comprehensive overview provides a foundational understanding of the security challenges facing IoT systems and potential mitigation strategies.

Neshenko et al. provide a detailed examination of IoT security vulnerabilities and exploitation techniques observed on an internet scale. They analyze various attack vectors, including device exploitation,

network-based attacks, and data breaches. The study presents empirical data on the prevalence of IoT exploits, highlighting common vulnerabilities such as weak authentication mechanisms and lack of encryption. The authors propose several mitigation strategies, including enhanced security protocols, automated patch management, and the use of machine learning for anomaly detection. Their research underscores the urgent need for improved IoT security measures to protect against widespread exploitation.

Ronen and Shamir investigate extended functionality attacks on IoT devices, focusing on the case of smart lights. Their research demonstrates how attackers can exploit vulnerabilities in IoT devices to manipulate their functionalities beyond their intended use. They highlight the potential risks of such attacks, including unauthorized access and control over connected devices. The authors suggest implementing stronger authentication mechanisms and secure firmware updates to mitigate these threats. Their study provides valuable insights into the novel attack vectors that can emerge in IoT environments and the importance of proactive security measures.

Alaba et al. provide a comprehensive survey on IoT security, addressing various threats and vulnerabilities. They discuss common attack vectors such as Denial of Service (DoS), data breaches, and device tampering. The authors propose a multi-layered security framework that includes device authentication, secure communication protocols, and real-time monitoring. Their work also highlights the importance of regulatory frameworks and industry standards in enhancing IoT security. The survey offers a thorough overview of the current state of IoT security and potential avenues for future research and development.

McDermott et al. investigate the use of deep learning approaches for botnet detection in IoT environments. Their research focuses on developing models that can identify botnet activities by analyzing network traffic patterns. They demonstrate the effectiveness of deep learning techniques in detecting anomalies and differentiating between normal and malicious traffic. The study suggests that deep learning models can significantly enhance the accuracy and speed of botnet detection, providing a valuable tool for improving IoT security. Their findings highlight the potential of advanced machine learning methods in addressing the challenges of IoT botnet detection.

Brun et al. explore the application of dense random neural networks for detecting attacks against IoTconnected home environments. Their study shows how deep learning models can be trained to identify

malicious activities by analyzing data from IoT devices. The authors highlight the advantages of using dense random neural networks, including their ability to handle large and complex datasets. Their research demonstrates that these models can achieve high detection rates and reduce false positives, making them suitable for real-time security monitoring in smart home environments. The study underscores the potential of neural network-based approaches for enhancing IoT security.

Meidan et al. present a study on detecting unauthorized IoT devices using machine learning techniques. Their research focuses on developing models that can identify devices that do not belong to the authorized network. They employ various machine learning algorithms to analyze device behavior and network traffic patterns. The study demonstrates that machine learning models can accurately detect unauthorized devices, reducing the risk of unauthorized access and potential attacks. The authors suggest that incorporating these models into IoT security frameworks can enhance the overall security of IoT ecosystems.

Doshi et al. examine the use of machine learning for detecting DDoS attacks targeting consumer IoT devices. Their study involves training models on network traffic data to identify patterns indicative of DDoS attacks. The authors demonstrate that machine learning techniques can effectively distinguish between normal and attack traffic, enabling timely detection and mitigation of DDoS attacks. They emphasize the importance of real-time monitoring and anomaly detection in protecting IoT devices from large-scale cyber-attacks. The study provides valuable insights into the application of machine learning for enhancing IoT security.

Sivaraman et al. investigate network-level security and privacy controls for smart-home IoT devices. Their research focuses on developing mechanisms to monitor and control network traffic to and from IoT devices. They propose a framework that includes device profiling, traffic analysis, and policy enforcement to enhance security and privacy. The study shows that these measures can effectively identify and block malicious traffic, protecting smart-home environments from various cyber threats. The authors highlight the need for user-friendly tools and interfaces to enable homeowners to manage their IoT security settings effectively. Their work contributes to the development of practical solutions for securing smart-home IoT ecosystems.

CHAPTER 3 EXISTING

SYSTEM

3.1 Overview:

The existing system for detecting anomalous network traffic associated with Mirai and BASHLITE IoT botnet attacks predominantly relies on traditional intrusion detection systems (IDS) that utilize signaturebased and rule-based techniques. These systems are integrated into network infrastructures and aim to identify and mitigate malicious activities by matching incoming traffic patterns against known attack signatures or predefined rules. However, these methods face significant limitations when dealing with the complex and evolving nature of modern IoT botnets like Mirai and BASHLITE.

Key Components of the Existing System:

Signature-Based Detection:

Signature-based IDS is a common approach where known attack signatures are stored in a database, and incoming network traffic is compared against these signatures to identify potential threats. In the context of Mirai and BASHLITE, these signatures include specific patterns of packets, payloads, and known IP addresses associated with these botnets.

Limitations:

Static Signatures: Since these systems rely on static signatures, they can only detect known attacks. Any new variant or previously unseen attack pattern will bypass detection.

High False Positives/Negatives: Minor variations in attack patterns can lead to a high rate of false positives (benign traffic identified as malicious) or false negatives (malicious traffic not detected).

Rule-Based Detection:

Rule-based IDS relies on predefined rules that are configured to trigger alerts when certain conditions are met. For instance, a rule might be set to flag traffic that exceeds a specific packet threshold or originates from a known malicious IP range. In IoT networks, these rules might be tailored to detect typical botnet behavior, such as abnormal bursts of outbound traffic or repeated connection attempts to a command and control (C&C) server.

Limitations:

Lack of Adaptability: Rule-based systems are rigid and unable to adapt to evolving threats. As attackers modify their tactics, techniques, and procedures (TTPs), existing rules may become obsolete.

Complex Configuration: Setting up and maintaining an extensive rule set is complex and requires continuous updates to stay effective, making it resource-intensive.

Network Flow Analysis:

Some existing systems incorporate network flow analysis, which examines metadata from network traffic (e.g., source/destination IPs, port numbers, packet counts) to detect anomalies. This method can identify traffic that deviates from normal patterns, such as unusually high volumes of traffic or unexpected communication between devices.

Limitations:

Scalability Issues: With the massive influx of IoT devices, network flow analysis becomes more challenging due to the increased volume and diversity of traffic.

Limited Granularity: Network flow data may lack the granularity needed to pinpoint specific malicious activities, especially when encrypted traffic is involved.

Manual Inspection and Forensic Analysis:

In scenarios where automated detection fails or is inconclusive, security teams often resort to manual inspection of traffic logs and forensic analysis. This involves deep packet inspection (DPI) and correlating network events to identify the presence of botnet activity.

Limitations:

Time-Consuming and Error-Prone: Manual methods are labor-intensive and can be prone to human error, especially when dealing with large volumes of data.

Reactive Rather Than Proactive: Manual inspection is often reactive, meaning that it occurs after a potential breach has been identified, rather than proactively preventing it.

CHAPTER 4

PROPOSED SYSTEM

4.1 Overview

1. **Library Imports:**
    - Essential libraries are imported for data handling (numpy, pandas), visualization (matplotlib, seaborn), machine learning (sklearn), and model persistence (joblib).
2. **Data Loading and Exploration:**
    - The dataset is loaded from a CSV file. ○ Basic exploratory data analysis (EDA) is conducted, including displaying the first and last few rows, summary statistics, and checking for unique values in the 'Attack' column.
3. **Data Preprocessing:**
    - Missing values are identified.

- A heatmap is generated to visualize correlations among features. ○ Categorical columns like 'Device_Name' and 'Attack_subType' are dropped, and the 'Attack' labels are encoded using LabelEncoder.

4. Data Visualization:

- A count plot visualizes the distribution of different attack categories in the dataset.

5. **Feature and Target Variable Separation:**
    - The feature set (x) is separated from the target variable (y), which represents attack classes.
6. **Data Splitting:**
    - The dataset is split into training and testing sets using an 70/30 ratio for model evaluation.
7. **Performance Metrics Function:**

- A function (performance_metrics) is defined to calculate and print accuracy, precision, recall, F1 score, and display a confusion matrix.

8. Model Training:

- Bernoulli Naive Bayes Classifier: The model is trained on the training data, and predictions are made on the test set. If a saved model exists, it is loaded instead of retraining.
- Random Forest Classifier: Similar logic applies. The model is either trained from scratch or loaded from a saved state.

9. Model Evaluation:

- Predictions from both classifiers are evaluated using the previously defined performance metrics function.

10. Prediction on New Data:

- A new test dataset is loaded and preprocessed similarly. Predictions are made using the Random Forest model, and results are printed, labeling each entry with the corresponding attack type.

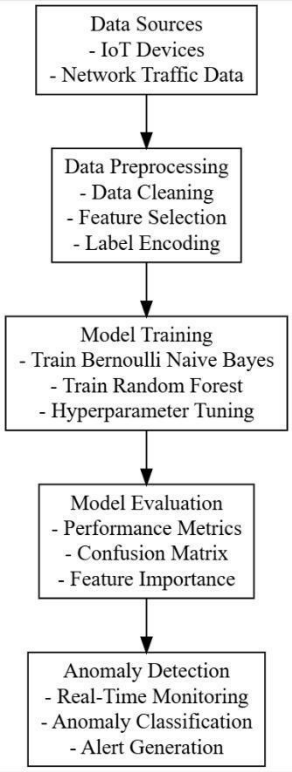


Figure 4.1: Architecture diagram of proposed system

4.2 Data Preprocessing and Splitting

The process begins with the collection and loading of the dataset into the environment, specifically targeting traffic data related to IoT devices under attack by Mirai and BASHLITE botnets. This data is typically rich with various network parameters, including packet size, transmission time, source and destination IPs, etc.

Data Preprocessing:

Null Value Removal:

The dataset undergoes initial preprocessing where null or missing values are identified and removed. Missing data can skew the results or lead to inaccuracies in model predictions. Techniques like removing rows with missing data or imputing values based on statistical measures may be used.

Label Encoding:

After handling null values, categorical variables are converted into numerical formats using label encoding. In this specific context, attack labels (e.g., Normal, BASHLITE, Mirai) are encoded into numerical values to make them interpretable by machine learning models.

Heatmap Visualization:

A correlation heatmap is generated to visualize the relationships between different features in the dataset. This step helps in identifying which features contribute most significantly to the classification task, guiding the selection of important variables for the model.

Data Splitting: 4. Training and Testing Set Creation:

The preprocessed data is split into training and testing sets, usually in a 70-30 ratio. The training set is used to build the model, while the testing set is reserved for evaluating its performance. This step is critical to ensure the model can generalize well to new, unseen data.

4.3 ML Model Building

4.3.1Naive Bayes Classifier

Overview:

- Naive Bayes is a family of probabilistic algorithms based on Bayes' theorem, which assumes independence between the features. It calculates the probability of a data point belonging to a class based on the likelihood of each feature given that class.

Key Characteristics:

- **Independence Assumption**: Assumes that features are independent, which may not hold true in real-world scenarios.
- **Fast and Efficient**: Generally computationally efficient, making it suitable for large datasets.
- **Good with Small Data**: Performs well with small datasets and is particularly effective for text classification (e.g., spam detection).

Performance:

- Often performs well for problems where the independence assumption holds, but its accuracy can drop when features are correlated.

4.3.2 Random Forest Classifier

Overview:

- Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of their predictions for classification (or average for regression). It improves accuracy and controls overfitting by aggregating predictions from several trees.

Key Characteristics:

- **Ensemble Method**: Combines the predictions of multiple trees, which helps reduce overfitting compared to individual decision trees.
- **Feature Importance**: Can identify the importance of different features in making predictions, aiding in model interpretability.
- **Robustness**: Handles non-linear relationships and interactions between features effectively.

Performance:

- Generally provides higher accuracy and robustness than Naive Bayes, especially with complex datasets and when features are correlated.

4.3.3 Comparison and When Random Forest Performs Better

1. **Feature Relationships**:
    - **Random Forest**: Excels when features are correlated or when there are complex interactions. It captures these relationships by constructing multiple trees.
    - **Naive Bayes**: Struggles in these scenarios due to its independence assumption.
2. **Data Size**:

- **Random Forest**: Works well with large datasets and can handle high dimensionality effectively.
- **Naive Bayes**: While efficient for smaller datasets, it may not generalize well with complex patterns in larger datasets.

3. **Model Interpretability**:
   - **Random Forest**: Provides insights into feature importance, making it easier to understand model decisions.
   - **Naive Bayes**: Less interpretable since it operates purely on probabilities without providing feature importance.
4. **Handling Outliers**:
   - **Random Forest**: More robust to outliers due to averaging across multiple trees. ○ **Naive Bayes**: Can be affected by outliers, as it relies on probability distributions.
5. **Overall Performance**:
   - In many practical scenarios, especially those involving complex relationships, high dimensionality, and larger datasets, **Random Forest tends to outperform Naive Bayes**.

     It provides better accuracy, robustness, and interpretability, making it a preferred choice in many classification tasks.

4.4 Advantages of the Proposed System

1. **Enhanced Detection Accuracy**:
   - Utilizing ensemble methods like Random Forest improves accuracy in identifying complex attack patterns compared to simpler models, leading to more reliable detection of IoT botnet activities.

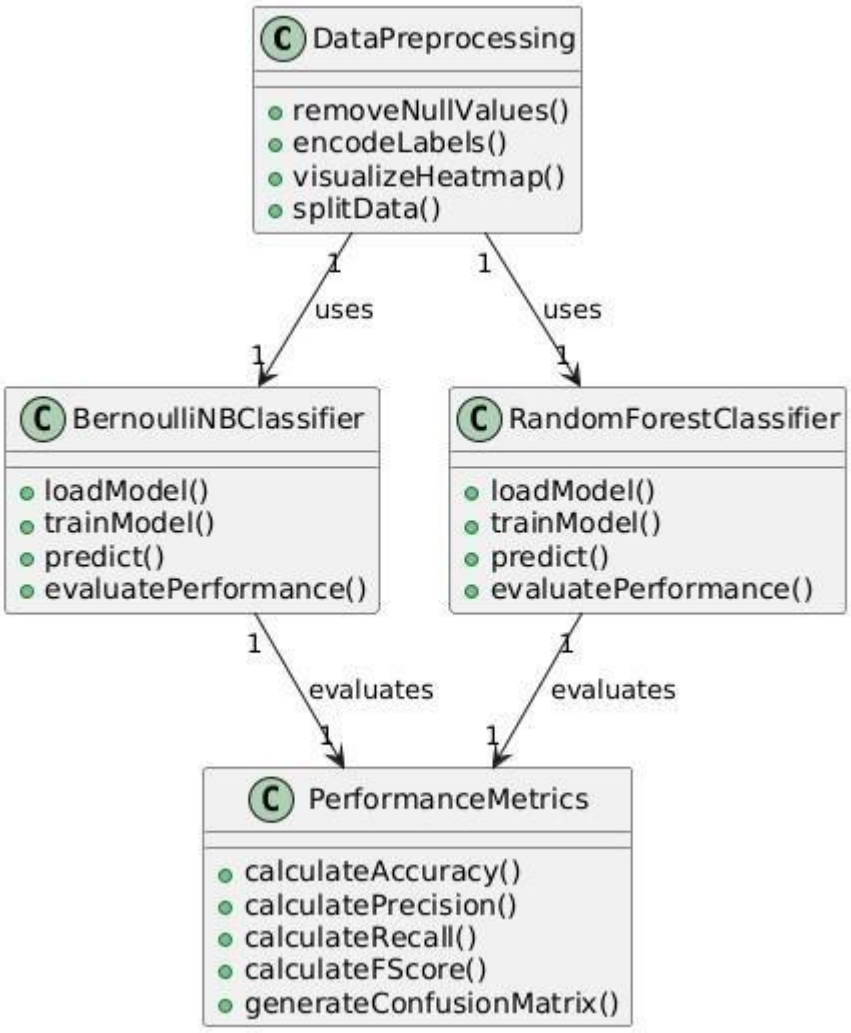
2. **Robustness to Noise and Outliers**:
    - The Random Forest classifier's inherent ability to mitigate the impact of noise and outliers results in more stable predictions, enhancing overall system reliability.
3. **Real-Time Anomaly Detection**:
    - The system can be deployed for real-time monitoring of network traffic, enabling immediate detection and response to potential threats, thereby minimizing damage.
4. **Feature Importance Analysis**:
    - The Random Forest model provides insights into feature importance, allowing stakeholders to understand which attributes contribute most significantly to attack detection, aiding in network security strategy formulation.
5. **Scalability**:
    - The proposed system can scale to accommodate larger datasets as IoT environments expand, ensuring consistent performance and adaptability to increasing data volumes.
6. **Flexibility in Handling Diverse Data Types**:
    - The approach can handle a variety of feature types (categorical, numerical) effectively, making it suitable for diverse IoT network configurations.
7. **Improved Model Generalization**:
    - By using ensemble methods, the system reduces the likelihood of overfitting, leading to better generalization on unseen data, which is crucial for effective anomaly detection.

CHAPTER 5

UML DIAGRAMS

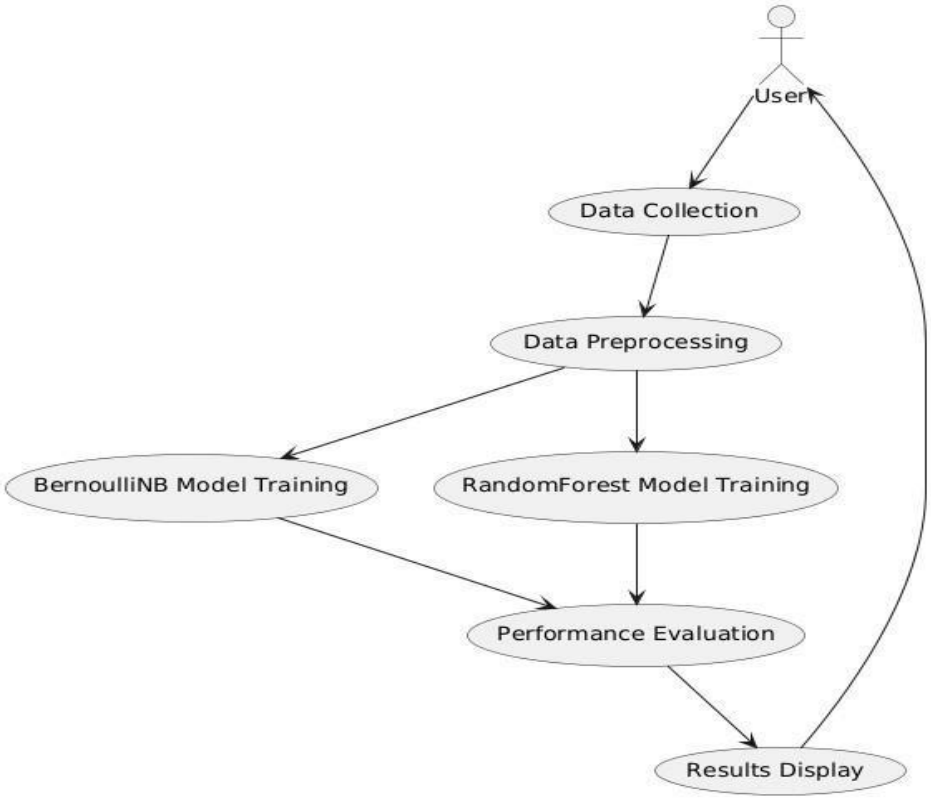
Class Diagram

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an “is-a” or “has-a” relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed “methods” of the class. Apart from this, each class may have certain “attributes” that uniquely identify the class.



Data flow diagram

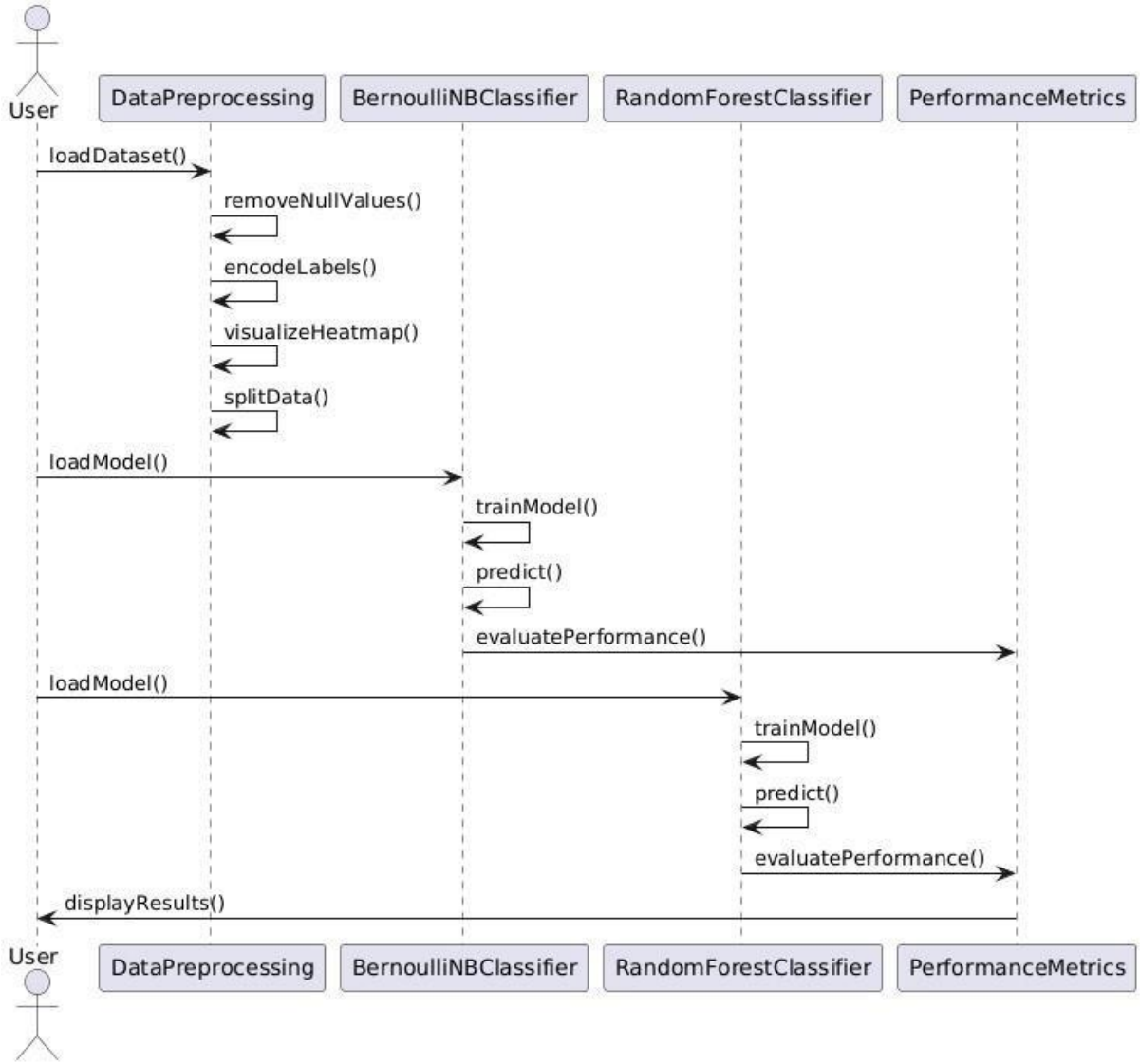
A Data Flow Diagram (DFD) is a visual representation of the flow of data within a system or process. It is a structured technique that focuses on how data moves through different processes and data stores within an organization or a system. DFDs are commonly used in system analysis and design to understand, document, and communicate data flow and processing.



Sequence Diagram

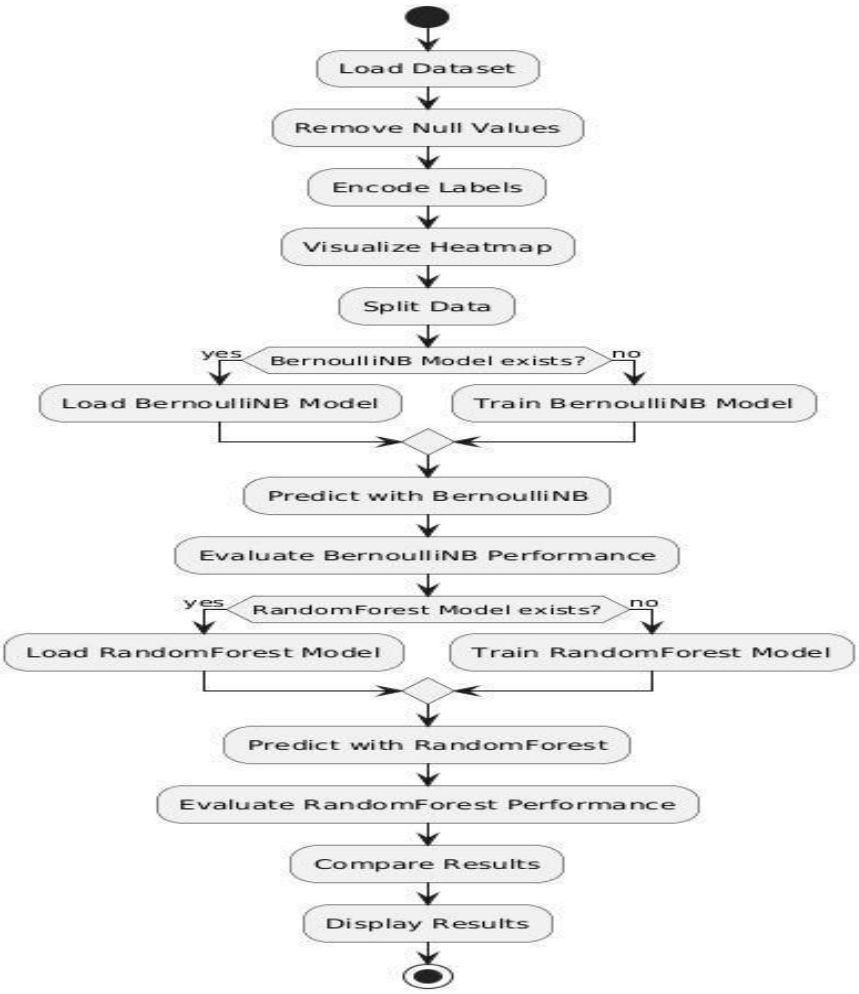
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live

simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

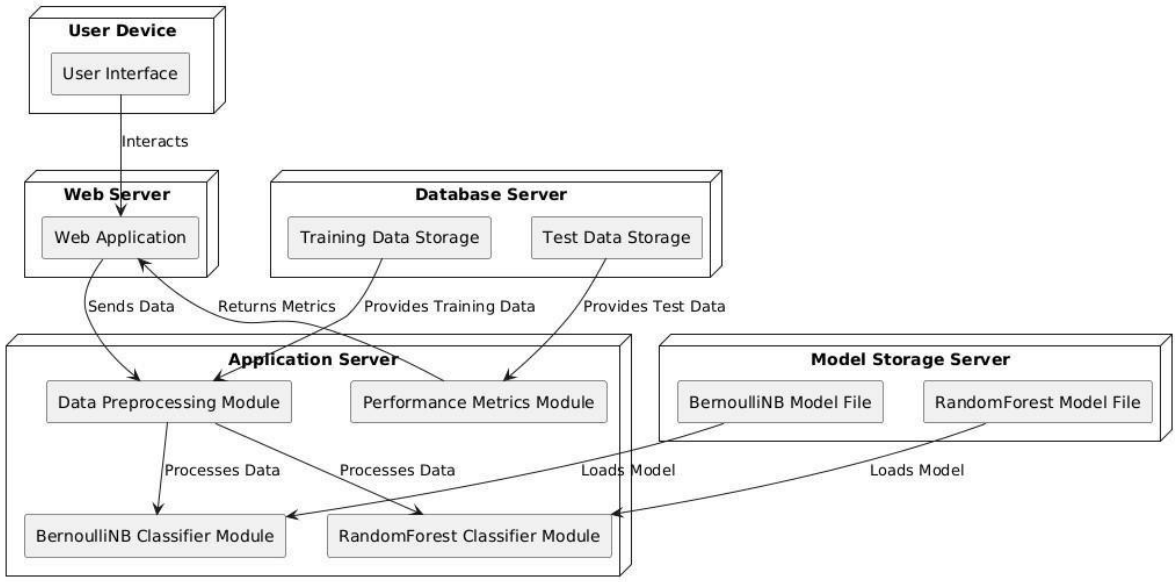


Activity diagram

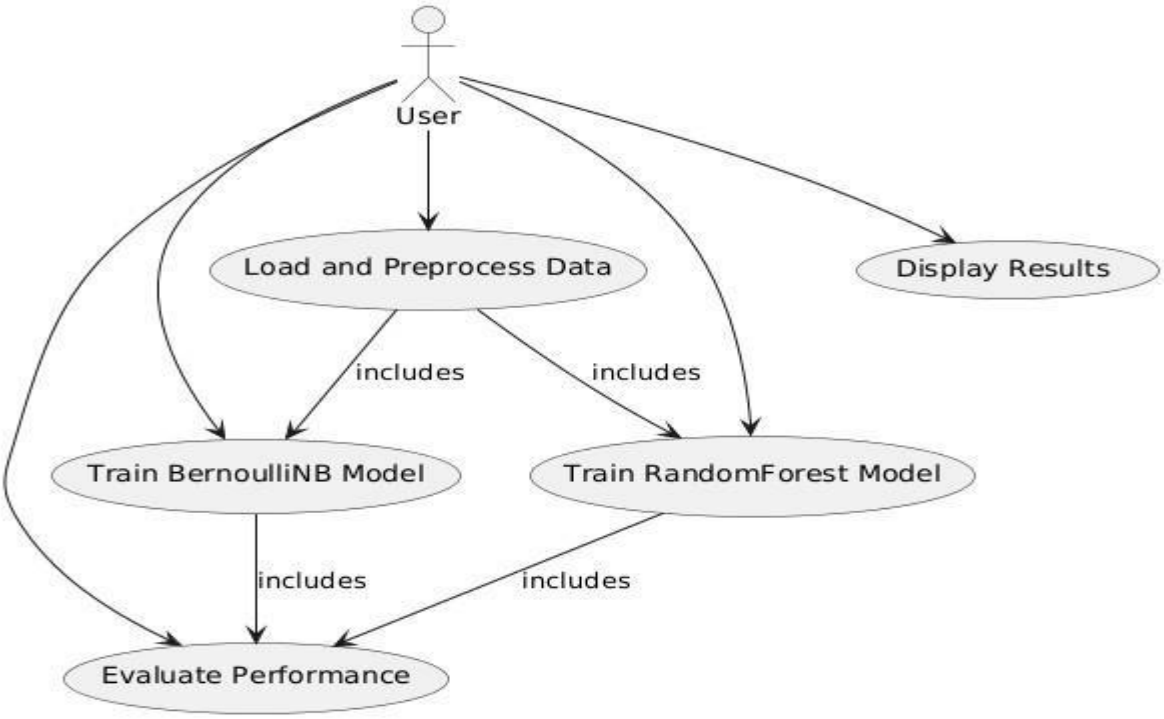
Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.



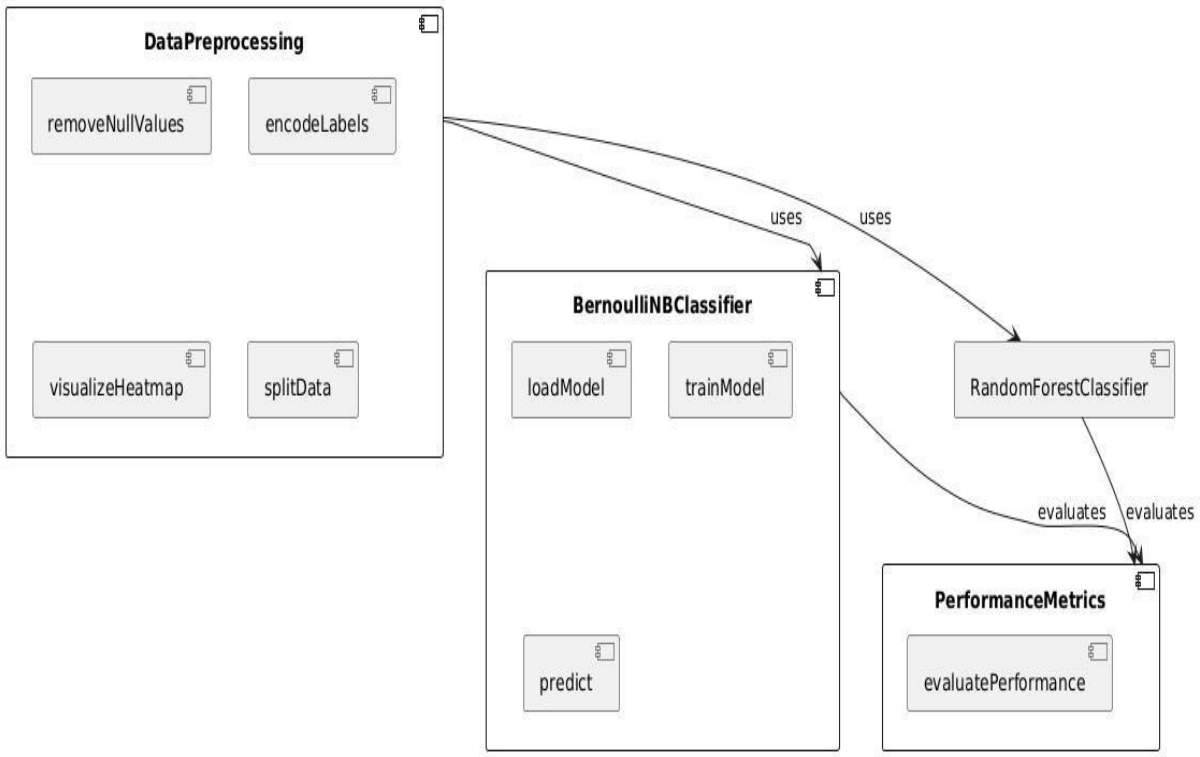
Deployment diagram: The deployment diagram visualizes the physical hardware on which the software will be deployed.



Use case diagram: The purpose of use case diagram is to capture the dynamic aspect of a system.



Component diagram: Component diagram describes the organization and wiring of the physical components in a system.



CHAPTER 6

SYSTEM REQUIREMENTS

Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

- Python IDLE 3.7 version (or)
- Anaconda 3.7 (or)
- Jupiter (or)

- Google colab

Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system	:	Windows, Linux
Processor	:	minimum intel i3
Ram	:	minimum 4 GB
Hard disk	:	minimum 250GB

CHAPTER 7

FUNCTIONAL REQUIREMENTS

Output Design

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

- External Outputs, whose destination is outside the organization
- Internal Outputs whose destination is within organization and they are the
- User's main interface with the computer.

- Operational outputs whose use is purely within the computer department.
- Interface outputs, which involve the user in communicating directly.

Output Definition

The outputs should be defined in terms of the following points:

- Type of the output
- Content of the output
- Format of the output
- Location of the output
- Frequency of the output
- Volume of the output
- Sequence of the output

Input Design

Input design is a part of overall system design. The main objective during the input design is as given below:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable and understood by the user.

Input Types

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

- External inputs, which are prime inputs for the system.
- Internal inputs, which are user communications with the system.
- Operational, which are computer department's communications to the system?
- Interactive, which are inputs entered during a dialogue.

Input Media

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

- Type of input
- Flexibility of format
- Speed
- Accuracy
- Verification methods
- Rejection rates
- Ease of correction
- Storage and handling requirements
- Security
- Easy to use
- Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

Data Validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

Computer-Initiated Interfaces

The following computer – initiated interfaces were used:

- The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
- Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

CHAPTER 8

SOURCE CODE

AI Framework for Identifying Anomalous Network Traffic in Mirai and BASHLITE IoT Botnet Attacks

```
## Importing Libraries import numpy as np import pandas as pd import
matplotlib.pyplot as plt import seaborn as sns import warnings
warnings.filterwarnings('ignore') from sklearn.preprocessing import LabelEncoder from
imblearn.over_sampling import SMOTE from sklearn.model_selection import
train_test_split from sklearn.metrics import precision_score from sklearn.metrics import
recall_score from sklearn.metrics import f1_score from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report import os import joblib from
sklearn.naive_bayes import BernoulliNB from sklearn.ensemble import
RandomForestClassifier
## Data Analysis  data = pd.read_csv(r"C:\Users\USER\Desktop\saint martins\Traffic\BoTNeTIoT-
L01-v2.csv") data data.head() data.tail() data.describe() data.info()  data['Attack'].unique()   #it  refers
to a value that appears only once or a distinct value within a specific dataset or column of a dataset ##
Data preprocessing data.isnull().sum() data.shape ### Heatmap  labels = set(data['Attack']) labels
plt.figure(figsize=(15,10)) sns.heatmap(data.corr(),cmap = 'Accent',annot = True) plt.xticks(rotation =
80) plt.yticks(rotation = 45) plt.show() labels = ['Normal', 'BASHLITE', 'Mirai'] labels columns =
['Device_Name','Attack_subType'] data = data.drop(columns = columns)  data
Labels = ['Attack'] for i in Labels:     data[i] =
LabelEncoder().fit_transform(data[i]) data ###
countplot sns.set(style="darkgrid")
plt.figure(figsize=(12, 6))  ax =
sns.countplot(x=data['Attack'], palette="Set2")
```

```
plt.title("Count Plot")   plt.xlabel("Categories")
plt.ylabel("Count")  plt.show() x = data.drop(['Attack'],
axis = 1) x y = data['Attack'] y
## Splitting the Data x_train, x_test, y_train, y_test = train_test_split(x,y, test_size =
0.30,  random_state  =  42)  x_train  y_train  x_test  y_test
x_train.shape y_train.shape ## Performance Evaluation precision
=  []  recall  =  []  fscore  =  []  accuracy  =  []  def
performance_metrics(algorithm,  predict,  testY):          testY  =
testY.astype('int')          predict  =  predict.astype('int')          p  =
precision_score(testY,  predict,average='macro')  *  100          r  =
recall_score(testY,  predict,average='macro')  *  100          f  =
f1_score(testY,  predict,average='macro')  *  100          a  =
accuracy_score(testY,predict)*100                accuracy.append(a)
precision.append(p)          recall.append(r)          fscore.append(f)
print(algorithm+' Accuracy  :  '+str(a))          print(algorithm+'
Precision  :  '+str(p))          print(algorithm+' Recall  :  '+str(r))
print(algorithm+'  FSCORE  :  '+str(f))
report=classification_report(predict,  testY,target_names=labels)
print('\n',algorithm+"  classification  report\n",report)
conf_matrix  =  confusion_matrix(testY,  predict)
plt.figure(figsize  =(5,  5))          ax  =  sns.heatmap(conf_matrix,
```

```
xticklabels = labels, yticklabels = labels, annot = True,
cmap="Blues"
,fmt ="g");     ax.set_ylim([0,len(labels)])
plt.title(algorithm+" Confusion matrix")
plt.ylabel('True class')      plt.xlabel('Predicted class')
plt.show()  ## BernoulliNBClassifier Algorithm if
os.path.exists('BernoulliNBClassifier.pkl'):     # Load the
Bernoulli Naive Bayes Classifier model     bnb_classifier
= joblib.load('BernoulliNBClassifier.pkl')        predict =
bnb_classifier.predict(x_test)
else:
   # Train and save the Bernoulli Naive Bayes Classifier model
bnb_classifier = BernoulliNB()     bnb_classifier.fit(x_train,
y_train)     joblib.dump(bnb_classifier,
'BernoulliNBClassifier.pkl') # Predict using the trained Bernoulli
Naive Bayes Classifier model y_pred_bnb =
bnb_classifier.predict(x_test)
# Evaluate the Bernoulli Naive Bayes Classifier model  performance_metrics('BernoulliNBClassifier',
y_pred_bnb, y_test)
## RandomForestClassifier if
os.path.exists('RandomForest_weights.pkl'):     # Load
```

```
the model from the pkl file      classifier =
joblib.load('RandomForest_weights.pkl') else:
    # Train the classifier on the training data      classifier =
RandomForestClassifier(random_state=42)      classifier.fit(x_train, y_train)
# Save the model weights to a pkl file          joblib.dump(classifier,
'RandomForest_weights.pkl')       print("RandomForest classifier model
trained and model weights saved.") y_pred = classifier.predict(x_test)
performance_metrics("RandomForest Classifier", y_pred, y_test) test =
pd.read_csv(r"test.csv") test test.info() columns =
['Device_Name','Attack_subType'] test = test.drop(columns = columns)
test predict = classifier.predict(test) predict
A='Normal'  B='BASHLITE'
c='mirai'
#test = pd.read_csv((r"test.csv")
predict = classifier.predict(test) for i in
range(len(predict)):      if predict[i] ==
0:          print("{}        :{}
            ".format(test.iloc[i,:],A))      elif
predict[i] == 1:
        print("{}  {}
            ".format(test.iloc[i,:],B))      elif
```

```
predict[i]== 2:          print("{} :{}

".format(test.iloc[i, :],c))
```

CHAPTER 9

OUTPUTS

The results of the analysis and machine learning models applied to the dataset are detailed below. Figures and tables included in this section visually represent key aspects of the dataset, model performance, and predictions.

	MI_dir_L0.1_weight	MI_dir_L0.1_mean	MI_dir_L0.1_variance	H_L0.1_weight	H_L0.1_mean	H_L0.1_variance	HH_L0.1_weight	HH_L0.1_mean	HH_L0.1_
0	1.000000	98.000000	0.000000e+00	1.000000	98.000000	0.000000e+00	1.000000	98.000000	0.000000e-
1	1.931640	98.000000	1.818989e-12	1.931640	98.000000	1.818989e-12	1.931640	98.000000	1.348699e
2	2.904273	86.981750	2.311822e+02	2.904273	86.981750	2.311822e+02	1.000000	66.000000	0.000000e-
3	3.902546	83.655268	2.040614e+02	3.902546	83.655268	2.040614e+02	1.000000	74.000000	0.000000e-
4	4.902545	81.685828	1.775746e+02	4.902545	81.685828	1.775746e+02	2.000000	74.000000	9.536743e
...	...	...	...	...	...	...	...	...	...
7062601	2.937269	217.763487	1.770682e+04	2.937269	217.763487	1.770682e+04	1.220882	60.000000	9.540000e
7062602	1.730254	282.630543	1.054589e+04	1.730254	282.630543	1.054589e+04	1.213342	330.000000	5.390000e
7062603	2.730251	299.980395	7.204117e+03	2.730251	299.980395	7.204117e+03	1.213352	330.000000	6.610000e
7062604	2.882414	216.723647	1.775308e+04	2.882414	216.723647	1.775308e+04	1.209274	60.000000	6.740000e
7062605	2.032574	154.377267	1.303249e+04	2.032574	154.377267	1.303249e+04	1.299681	145.339354	1.010891e-

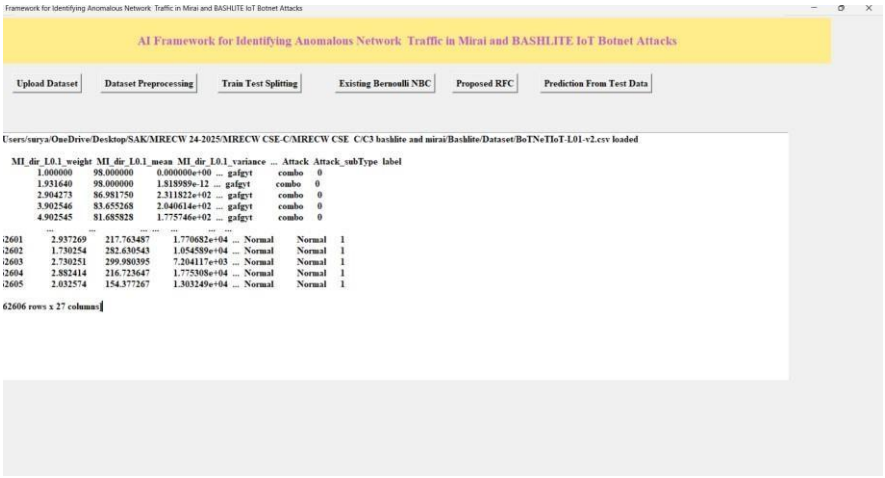


Figure 10.1: GUI interface and Overview of the Dataset

This figure displays the first few rows and last few rows of the dataset.

	MI_dir_L0.1_weight	MI_dir_L0.1_mean	MI_dir_L0.1_variance	H_L0.1_weight	H_L0.1_mean	H_L0.1_variance	HH_L0.1_weight	HH_L0.1_mean	HH_L0.1_std
count	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06	7.062606e+06
mean	3.400682e+03	1.794441e+02	1.931062e+04	3.400682e+03	1.794441e+02	1.931066e+04	1.892359e+03	1.792406e+02	4.415659e+0
std	2.897012e+03	1.537109e+02	2.636844e+04	2.897012e+03	1.537107e+02	2.636842e+04	2.523083e+03	2.059018e+02	2.243629e+0
min	1.000000e+00	6.000000e+01	0.000000e+00	1.000000e+00	6.000000e+01	0.000000e+00	1.000000e+00	6.000000e+01	0.000000e+0
25%	1.000000e+00	6.000000e+01	0.000000e+00	1.000000e+00	6.000000e+01	0.000000e+00	1.000000e+00	6.000000e+01	0.000000e+0
50%	3.644882e+03	7.412707e+01	9.807711e+01	3.644882e+03	7.412707e+01	9.810144e+01	1.071281e+00	7.020764e+01	0.000000e+0
75%	6.354692e+03	3.486463e+02	4.887076e+04	6.354692e+03	3.486463e+02	4.887076e+04	4.201684e+03	9.314709e+01	3.293467e+0
max	8.946997e+03	1.401994e+03	4.520011e+05	8.946997e+03	1.401994e+03	4.520011e+05	7.944987e+03	1.470000e+03	6.784580e+0

Figure 10.2: Dataset Description

This figure shows the descriptive statistics of the dataset, including measures such as count, mean, standard deviation, min, and max values for each feature. It offers insights into the range and distribution of data points.

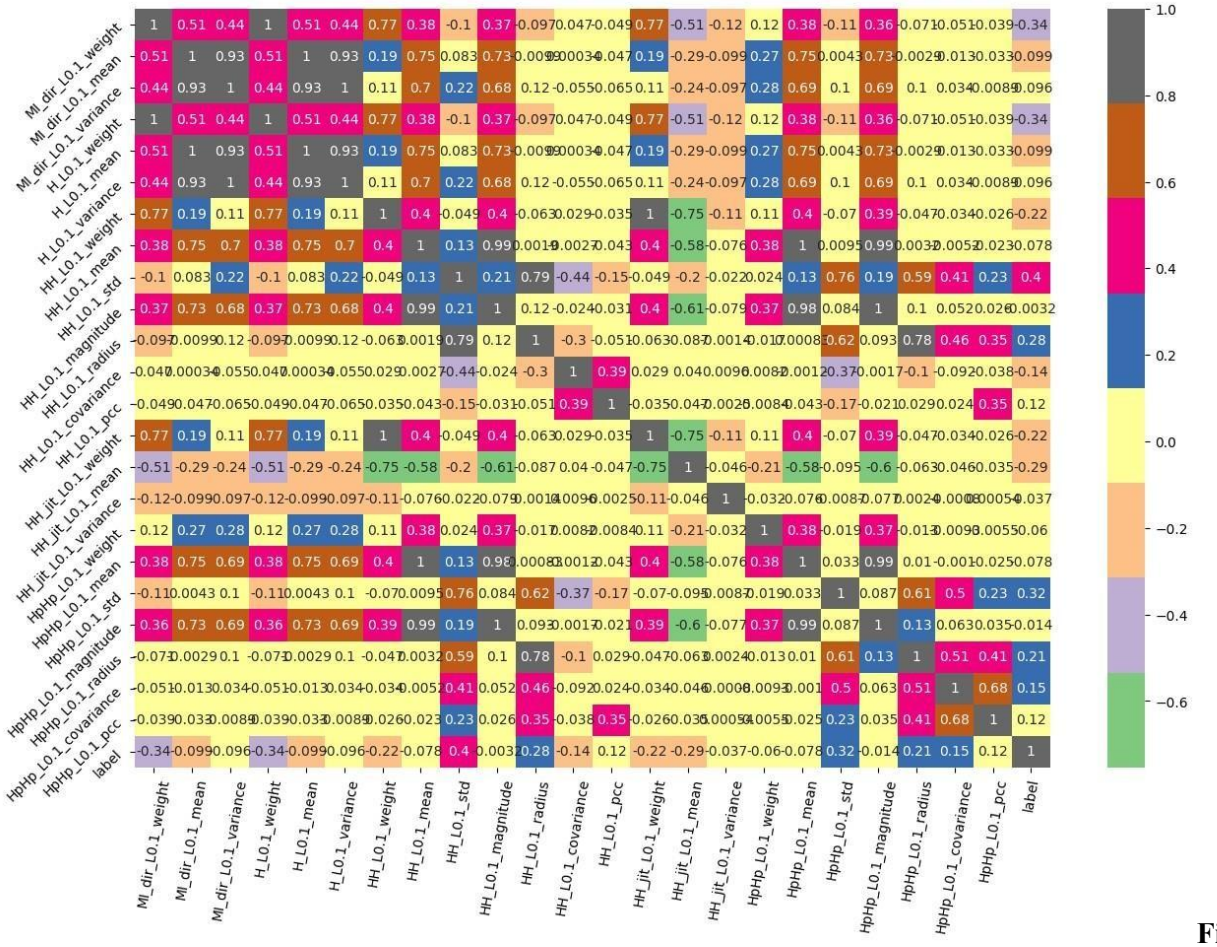


Figure 10.3: Correlation Heatmap

This heatmap illustrates the correlation between different features in the dataset. The color gradient indicates the strength and direction of correlations, helping to identify which features are most related to each other.

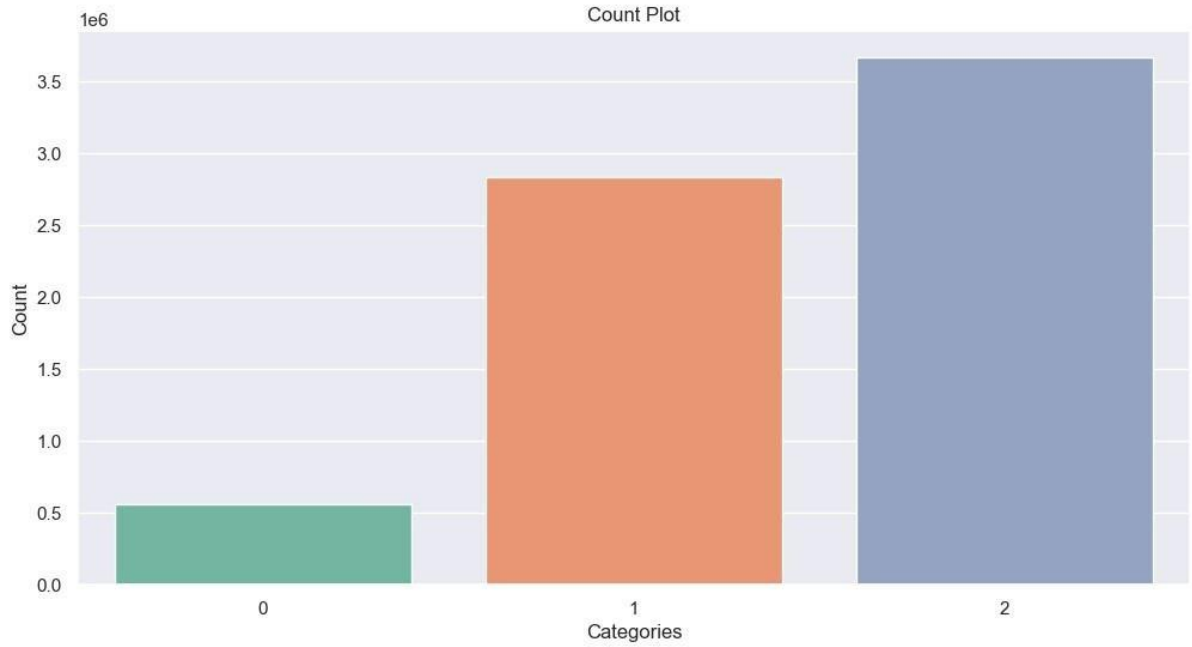


Figure 10.4: Count Plot of Attack Categories

This count plot visualizes the distribution of different attack categories in the dataset. It shows the number of instances for each class: Normal, BASHLITE, and Mirai.

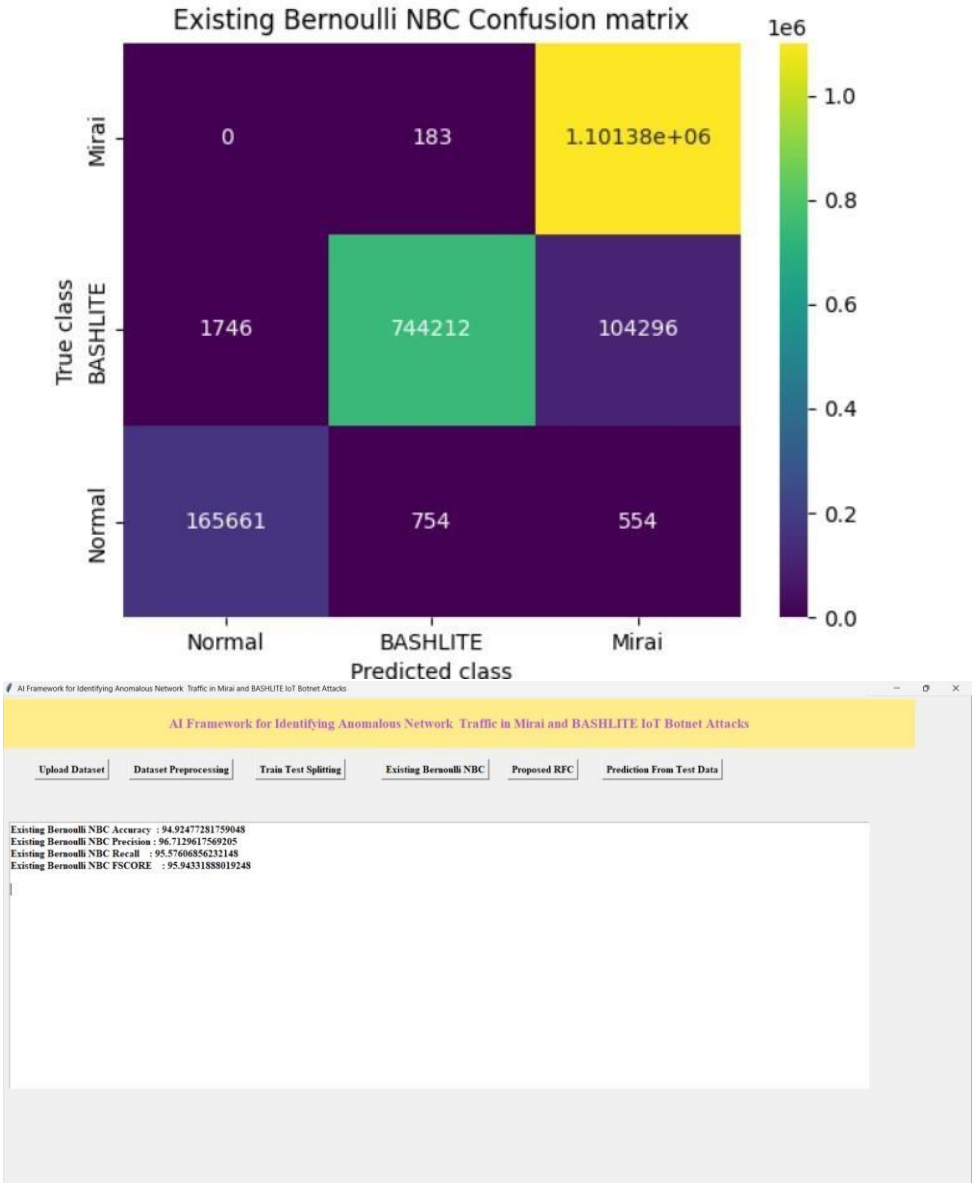


Figure 10.5: Confusion Matrix of Bernoulli Naive Bayes Classifier

This confusion matrix displays the performance of the Bernoulli Naive Bayes Classifier on the test data. It shows the number of true positives, false positives, true negatives, and false negatives for each class.

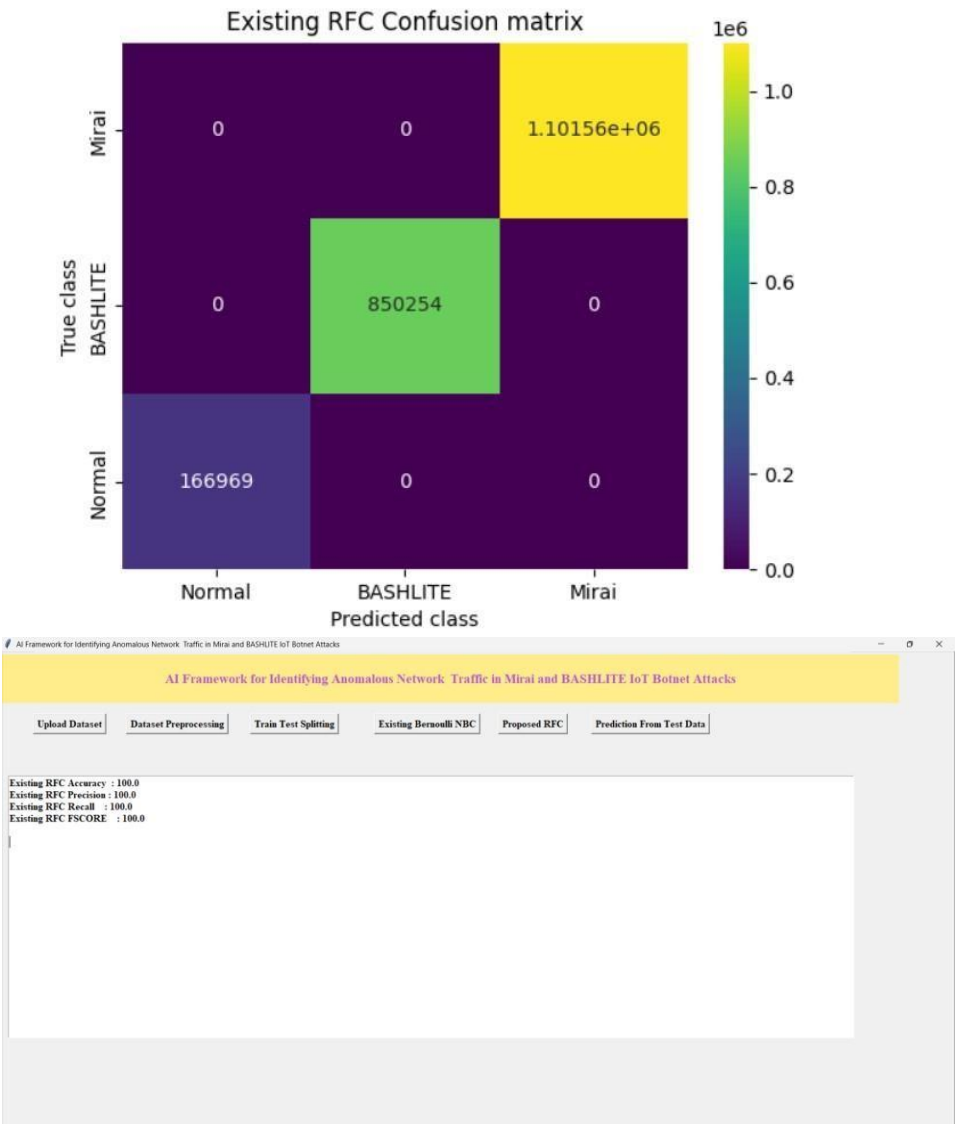


Figure 10.6: Confusion Matrix and Performance metrics of Random Forest Classifier

This figure presents the confusion matrix for the Random Forest Classifier. It provides a detailed breakdown of the classifier's performance, highlighting the correct and incorrect predictions for each attack category.

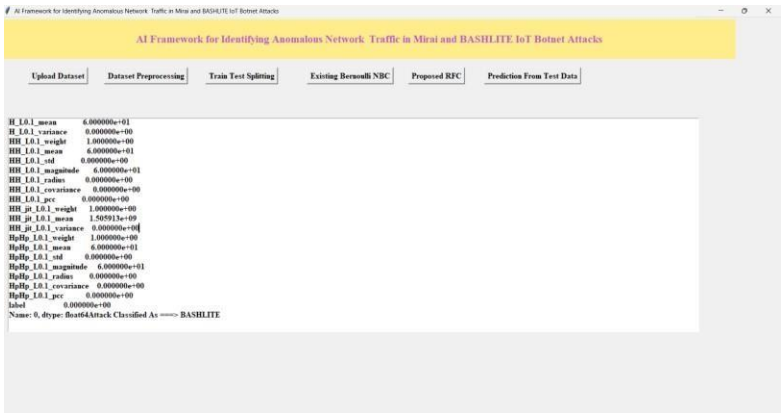


Figure 10.7: Prediction on New Test Data.

This figure presents the predictions made by the Random Forest Classifier on new test data, specifically for the first 15 rows. It shows the predicted class for each instance and the corresponding actual values for comparison.

CHAPTER 10

CONCLUSION

This research successfully demonstrates the application of machine learning techniques to enhance the detection of sophisticated IoT-based botnet attacks. By leveraging the Bernoulli Naive Bayes and Random Forest classifiers, the framework addresses the challenges posed by the increasing volume and complexity of network traffic due to the proliferation of IoT devices.

The implementation focuses on two prevalent botnets, Mirai and BASHLITE, which have historically exploited vulnerabilities in IoT devices to orchestrate large-scale Distributed Denial of Service (DDoS) attacks and other cyber threats. The use of machine learning enables the system to identify anomalous patterns in network traffic, which traditional rule-based systems might overlook. The Random Forest classifier, in particular, provides a high degree of accuracy and robustness, making it a suitable choice for real-world deployment in dynamic network environments.

This AI-driven approach offers significant improvements over manual inspection and traditional anomaly detection methods. The ability to process and analyze vast amounts of data in real time, coupled with the adaptability of machine learning models to evolving attack patterns, ensures that the framework remains

effective even as cyber threats become more sophisticated. Additionally, the integration of performance metrics like accuracy, precision, recall, and F1-score provides a comprehensive evaluation of the models, ensuring they meet the necessary standards for deployment.

CHAPTER 11

FUTURE SCOPE

- 1.Integration with RealTime Systems: One of the primary future directions for this project is to integrate the trained machine learning models into real-time network monitoring systems. This would involve deploying the models within network security infrastructures to continuously monitor and analyze incoming traffic for anomalies, providing instant alerts and automated responses to potential threats. Real-time deployment would also require optimizing the models for speed and efficiency, ensuring minimal latency in detection.
- 2.Incorporating Advanced Machine Learning Techniques: While Bernoulli Naive Bayes and Random Forest classifiers provide a solid foundation, future enhancements could explore more advanced machine learning techniques, such as deep learning. Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) could be employed to capture more complex

patterns in network traffic data, potentially improving detection rates for sophisticated or novel attack vectors.

- Expanding the Scope to Include Other Botnets: The current framework focuses on detecting Mirai and BASHLITE botnets. Future work could expand the scope to include other emerging IoT botnets, such as Hajime, Amnesia, or Reaper. By incorporating a broader range of attack types, the framework could be made more versatile and capable of handling a wider array of threats.
- Feature Engineering and Dimensionality Reduction: Further exploration into feature engineering could enhance the model's performance. Techniques like Principal Component Analysis (PCA) or t-SNE could be employed to reduce dimensionality, enabling the models to focus on the most critical features while reducing computational overhead. Additionally, the exploration of new features, derived from more granular network traffic data, could improve the model's ability to distinguish between normal and malicious traffic.
- Addressing Data Imbalance and Improving Model Generalization: Given the nature of cyber attack datasets, there is often a significant imbalance between normal and anomalous traffic. Future work could involve the implementation of more sophisticated data balancing techniques, such as Synthetic Minority Over-sampling Technique (SMOTE) or Adaptive Synthetic (ADASYN), to enhance model training. Additionally, techniques like cross-validation could be employed to improve model generalization and reduce overfitting.
- Collaborative Threat Intelligence: Another potential area for future development is the integration of collaborative threat intelligence. By sharing anonymized network traffic patterns and attack signatures across different organizations, the framework could benefit from a broader knowledge base, improving its ability to detect and respond to emerging threats. This could be facilitated through secure, decentralized platforms that allow for the exchange of threat data while maintaining privacy and confidentiality.
- Compliance and Ethical Considerations: As the framework is developed further, attention should be given to ensuring compliance with cybersecurity regulations and ethical considerations. This includes ensuring that the system respects user privacy and data protection laws, as well as addressing any potential biases in the model that could lead to false positives or negatives.

– User Interface and Visualization: To make the system more user-friendly, future work could involve developing an intuitive user interface that allows network administrators to easily interpret the model's predictions and gain insights into detected anomalies. Visualization tools, such as interactive dashboards and real-time alerts, could be integrated to provide a more comprehensive view of the network's security status.

CHAPTER 12

REFERENCES

[1] Abomhara, M.; Køien, G.M. Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks. J. Cyber Secur. Mobil. 2015, 4, 65–88.

[2] Andrea, I.; Chrysostomou, C.; Hadjichristofi, G. Internet of things: Security vulnerabilities and challenges. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 180–187.

[3] Deogirikar, J.; Vidhate, A. Security attacks in IoT: A survey. In Proceedings of the 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 10–11 February 2017; pp. 32–37.

[4] Neshenko, N.; Bou-Harb, E.; Crichigno, J.; Kaddoum, G.; Ghani, N. Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. IEEE Commun. Surv. Tutor. 2019, 21, 2702–2733.

[5] Ronen, E.; Shamir, A. Extended functionality attacks on IoT devices: The case of smart lights. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbrücken, Germany, 21–24 March 2016; pp. 3–12.

[6] Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of things security: A survey. J. Netw. Comput. Appl. 2017, 88, 10–28.

[6] McDermott, C.D.; Majdani, F.; Petrovski, A.V. Botnet detection in the internet of things using deep learning approaches. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.

[7] Brun, O.; Yin, Y.; Gelenbe, E. Deep learning with dense random neural network for detecting attacks against IoT-connected home environments. Procedia Comput. Sci. 2018, 134, 458–463.

[8] Meidan, Y.; Bohadana, M.; Shabtai, A.; Ochoa, M.; Tippenhauer, N.O.; Guarnizo, J.D.; Elovici, Y. Detection of unauthorized IoT devices using machine learning techniques. arXiv 2017, arXiv:1709.04647.

[9] Doshi, R.; Apthorpe, N.; Feamster, N. Machine learning DDoS detection for consumer internet of things devices. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24–24 May 2018; pp. 29–35.

[10] Sivaraman, V.; Gharakheili, H.H.; Vishwanath, A.; Boreli, R.; Mehani, O. Network-level security and privacy control for smart-home IoT devices. In Proceedings of the 2015 [11] IEEE 11th International

Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Abu Dhabi, UAE, 19–21 October 2015; pp. 163–167.

[11]International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Abu Dhabi, UAE, 19–21 October 2015; pp. 163–167.

[12] Yavuz, F.Y.; Devrim, Ü.N.A.L.; Ensar, G.Ü.L. Deep learning for detection of routing attacks in the internet of things. Int. J. Comput. Intell. Syst. 2018, 12, 39–58.

[13] Shiranzaei, A.; Khan, R.Z. An Approach to Discover the Sinkhole and Selective Forwarding Attack in IoT. J. Inf. Secur. Res. 2018, 9, 107.

[14] Soni, V.; Modi, P.; Chaudhri, V. Detecting Sinkhole attack in wireless sensor network. Int. J. Appl. Innov. Eng. Manag. 2013, 2, 29–32.

[15] Palacharla, S.; Chandan, M.; GnanaSuryaTeja, K.; Varshitha, G. Wormhole attack: A major security concern in internet of things (IoT). Int. J. Eng. Technol. 2018, 7, 147–150.