DoS ATTACKS DETECTION USING GENETIC ALGORITHM

Submitted in partial fulfillment of the requirements for the award of

Bachelor of Engineering degree in Computer Science and Engineering

By

JAGADEESH P (Reg.No – 39110394)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119

APRIL - 2023

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **JAGADEESH P (39110394)** who carried out the Project Phase-2 entitled **"DoS ATTACKS DETECTION USING GENETIC ALGORITHM"** under my supervision from January 2023 to April 2023.

Internal Guide

Dr. L. SUJI HELEN, M.E., Ph.D.

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.

Submitted for Viva voce Examination held on 20.4.2023

Internal Examiner **External Examiner**

DECLARATION

I, **JAGADEESH P (39110394),** hereby declare that the Project Phase-2 Report entitled **"DoS ATTACKS DETECTION USING GENETIC ALGORITHM"** done by me under the guidance of **Dr. L. SUJI HELEN, M.E., Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 20.4.2023

PLACE: Chennai **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. L. SUJI HELEN, M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Intrusion detection systems (IDSs) are currently drawing a great amount of interest as a key part of system defence. IDSs collect network traffic information from some point on the network or computer system and then use this information to secure the network. To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming. In this project, Genetic Algorithm (GA) is proposed as a tool that capable to identify harmful type of connections in a computer network. Different features of connection data such as duration and types of connection in network were analyzed to generate a set of classification rule. For this project, standard benchmark dataset known as KDD dataset was investigated and utilized to study the effectiveness of the proposed method on this problem domain. The rules comprise of eight variables that were simulated during the training process to detect any malicious connection that can lead to a network intrusion. With good performance in detecting bad connections, this method can be applied in intrusion detection system to identify attack thus improving the security features of a computer network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table No. **Table Name** **Page No.**

LIST OF ABBREVATIONS

DoS	Denial-of-Service

HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
KDD	Knowledge discovery in databases
NIDS	Network Intrusion Detection System
SVM	Support Vector Machines
SDN	Software-defined Networking
TCP	Transmission Control Protocol

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Approaches for intrusion detection can be broadly divided into two types: misuse detection and anomaly detection. In misuse detection system, all known types of attacks (intrusions) can be detected by looking into the predefined intrusion patterns in system audit traffic. In case of anomaly detection, the system first learns a normal activity profile and then flags all system events that do not match with the already established profile. The main advantage of the misuse detection is its capability for high detection rate with a difficulty in finding the new or unforeseen attacks. The advantage of anomaly detection lies in the ability to identify the novel (or unforeseen) attacks at the expense of high false positive rate. Network monitoring-based machine learning techniques have been involved in diverse fields.

Dos attack:

A Denial of Service (DoS) attack is a type of cyberattack that aims to disrupt the normal functioning of a network or website by overwhelming it with traffic or requests. In a DoS attack, the attacker floods the targeted network or website with a large volume of traffic, making it difficult or impossible for legitimate users to access the service. Detecting DoS attacks is crucial for maintaining the availability of networks and websites. Without proper detection and mitigation techniques, a successful DoS attack can result in extended downtime, lost revenue, and damage to reputation. There are several methods for detecting DoS attacks. One common approach is to use traffic analysis techniques to identify abnormal traffic patterns. In a DoS attack, the attacker usually generates a high volume of traffic that is different from the normal traffic on the network or website. By analyzing network traffic, it is possible to identify patterns that are indicative of a DoS attack. Another approach for detecting DoS attacks is to use signature-based detection. In this approach, the network or website is monitored for known attack signatures. When an attack signature is detected, an alert is generated to notify security personnel. One limitation of signature-based detection is that it can only detect attacks that have a known signature. Zero-day attacks or attacks that use new and unknown

techniques may not be detected by signature-based detection. Anomaly-based detection is another approach for detecting DoS attacks. In this approach, the system learns the normal behavior of the network or website and detects deviations from this normal behavior. By using machine learning algorithms, it is possible to identify unusual patterns of traffic that may be indicative of a DoS attack. Anomaly-based detection has the advantage of being able to detect unknown attacks, but it also has the risk of generating false positives. A false positive occurs when the system identifies legitimate traffic as malicious, which can lead to unnecessary alerts and wasted resources. Another approach for detecting DoS attacks is to use hybrid detection methods that combine signature-based and anomaly-based detection techniques. This approach has the advantage of being able to detect both known and unknown attacks while minimizing false positives. In addition to detection techniques, there are several mitigation techniques that can be used to prevent DoS attacks. One common mitigation technique is to use rate limiting, which limits the rate at which traffic is allowed to enter the network or website. By limiting the rate of incoming traffic, it is possible to prevent the network or website from being overwhelmed by a flood of requests. Another mitigation technique is to use blacklisting, which blocks traffic from known malicious sources. Blacklisting can be an effective way to prevent attacks from known attackers, but it is not effective against zero-day attacks or attacks that use new and unknown techniques. In addition to rate limiting and blacklisting, there are several other mitigation techniques that can be used to prevent DoS attacks. These include using load balancers to distribute traffic across multiple servers, using content delivery networks (CDNs) to cache content closer to users, and using intrusion prevention systems (IPS) to automatically block malicious traffic. In conclusion, detecting and mitigating DoS attacks is crucial for maintaining the availability of networks and websites. There are several detection techniques that can be used, including traffic analysis, signature-based detection, anomaly-based detection, and hybrid detection. Each technique has its advantages and limitations, and a combination of techniques may be necessary to provide effective protection against DoS attacks. Mitigation techniques such as rate limiting, blacklisting, load balancing, and IPS can also be used to prevent DoS attacks. By implementing effective detection and mitigation techniques,

organizations can protect themselves from the damaging effects of DoS attacks and ensure the continued availability of their networks and websites.

Deep Learning:

Deep learning is a subset of machine learning that is inspired by the workings of the human brain. The key idea behind deep learning is to train artificial neural networks that can learn and improve from large amounts of data. Deep learning is a type of artificial intelligence that has revolutionized many fields, including computer vision, natural language processing, speech recognition, and robotics. The term "deep" refers to the multiple layers of neural networks used in deep learning. These layers allow the network to learn increasingly complex features from the data. Each layer of the network processes the input data and produces output that is passed on to the next layer. The output of the final layer represents the network's prediction. One of the key advantages of deep learning is its ability to automatically extract features from raw data, without requiring manual feature engineering. This has made it possible to use deep learning for tasks such as image and speech recognition, where traditional feature engineering methods were not as effective. Deep learning has been particularly successful in computer vision, where it has achieved state-of-the-art performance on a variety of tasks, such as image classification, object detection, and segmentation. Deep learning models such as convolutional neural networks (CNNs) have been used to develop systems that can recognize objects, faces, and gestures in images and videos with remarkable accuracy. In natural language processing, deep learning has also made significant strides. Recurrent neural networks (RNNs) and their variants, such as long short-term memory (LSTM) networks, have been used to develop systems that can perform tasks such as language translation, sentiment analysis, and speech recognition. These systems have become increasingly sophisticated over the years, with some achieving human-level performance on specific tasks. Speech recognition is another area where deep learning has had a significant impact. Deep learning models such as convolutional neural networks and recurrent neural networks have been used to develop speech recognition systems that can transcribe spoken words with high accuracy. These systems have been used in various applications, such as voice assistants, automatic transcription, and even language learning. Despite its successes, deep learning has some

limitations. One of the main challenges of deep learning is the need for large amounts of labeled data to train the models effectively. Collecting and labeling data can be time-consuming and expensive, especially for complex tasks. Additionally, deep learning models can be computationally expensive to train, requiring high-end hardware such as graphics processing units (GPUs) and specialized computing clusters. Another challenge of deep learning is the interpretability of the models. Deep learning models are often referred to as "black boxes" because it can be difficult to understand how they arrive at their predictions. This lack of interpretability can be problematic in certain applications, such as medicine, where it is important to understand the reasons behind a diagnosis. In conclusion, deep learning is a subset of machine learning that has revolutionized many fields, including computer vision, natural language processing, and speech recognition. The key idea behind deep learning is to train artificial neural networks that can learn and improve from large amounts of data. Deep learning has some limitations, such as the need for large amounts of labeled data and the interpretability of the models. However, despite these limitations, deep learning has already had a significant impact on many industries and is expected to continue to advance as more data becomes available and computing power increases.

Intrusion Detection System (IDS):

An Intrusion Detection System (IDS) is a security technology that monitors network traffic for suspicious activity and alerts security personnel or automated security systems when such activity is detected. The purpose of an IDS is to identify and respond to attacks on a network or system before damage can be done. IDS can be divided into two main categories: network-based IDS (NIDS) and host-based IDS (HIDS). A network-based IDS monitors network traffic, looking for suspicious activity such as known attack signatures or unusual traffic patterns. A host-based IDS, on the other hand, runs on individual machines and monitors activity on the host, such as file access, logins, and system calls. Intrusion detection systems can also be classified as signature-based or anomaly-based. A signature-based IDS looks for known attack patterns, such as a particular sequence of packets in network traffic. An anomaly-based IDS, on the other hand, looks for unusual activity that deviates from normal behavior. This can include unusual traffic patterns, unexpected system calls, or unusually high resource usage. One of the

primary advantages of an IDS is its ability to detect attacks that have not been seen before. This is especially important in the case of zero-day attacks, where attackers exploit vulnerabilities that have not yet been discovered by security experts. An IDS can detect these attacks by looking for unusual behavior, even if no known attack signature exists. Another advantage of IDS is their ability to provide real-time alerts to security personnel or automated security systems. This allows for a rapid response to detected attacks, minimizing the damage that can be done. IDS can also be used to generate detailed reports on network activity, which can be used to identify trends, monitor compliance, and improve overall security. Despite their advantages, IDS also have some limitations. One of the main challenges of IDS is the difficulty in distinguishing between legitimate traffic and malicious traffic. This can result in false positives, where legitimate traffic is flagged as suspicious, leading to unnecessary alerts and wasted resources. Conversely, false negatives can occur when malicious activity goes undetected, potentially leading to security breaches. Another challenge of IDS is their ability to scale to large networks. As network traffic increases, IDS must be able to handle the increased volume of data without slowing down network performance. Additionally, IDS must be able to handle encrypted traffic, which can hide malicious activity from the IDS. To overcome these limitations, IDS can be integrated with other security technologies, such as firewalls, antivirus software, and intrusion prevention systems (IPS). Firewalls can be used to block traffic from known malicious sources, while antivirus software can detect and remove malware. IPS can be used to automatically block suspicious activity detected by the IDS. In conclusion, an Intrusion Detection System is a vital security technology for identifying and responding to attacks on a network or system. IDS can be divided into two main categories: network-based IDS and host-based IDS and can be classified as signature-based or anomaly-based. IDS has many advantages, including its ability to detect zero-day attacks and provide real-time alerts to security personnel or automated security systems. However, IDS also has some limitations, including the difficulty in distinguishing between legitimate and malicious traffic and the ability to scale to large networks. By integrating IDS with other security technologies, such as firewalls, antivirus software, and IPS, these limitations can be overcome, and network security can be greatly improved.

1.2 PROBLEM STATEMENT

- To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming.
- An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection.
- It needs a way that can detect network intrusion to reflect the current network traffics.

1.3 OBJECTIVES

- The primary purposes for an IDS deployment are to reduce risk, identify error, optimize network use, provide insight into threat levels, and change user behavior. Thus, an IDS provides more than just detection of intrusion.
- Genetic Algorithm (GA) developed specifically for problems with multiple objectives. They differ primarily from traditional GA by using specialized fitness functions and introducing methods to promote solution diversity.
- The goal of this algorithm is to create a model that predicts the value of a target variable, for which the decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

CHAPTER 2

LITERATURE SURVEY

[1] **Project Title** **:** DOS attacks detection in software-defined networks using machine learning.

Author Name **:** M. M. Eisa, M. S. Shatnawi, and O. F. AlShorman

Year of Publish **:** 2021

Abstract **:** The authors used the KDDCup99 dataset to train and test the proposed model and used several machine learning algorithms such as Random Forest, K-Nearest Neighbor, Decision Tree, and Support Vector Machine to evaluate the performance of the model. The results showed that the proposed model using the Random Forest algorithm achieved the highest accuracy in detecting DOS attacks in SDN.

[2] **Project Title** **:** A machine learning-based approach to detect DoS attacks in software-defined networks

Author Name **:** A. Hossain, M. N. Uddin, and M. F. Hossain

Year of Publish **:** 2021

Abstract **:** The authors used the NSL-KDD dataset to train and test the proposed model and used several machine learning algorithms such as Random Forest, Support Vector Machine, and Artificial Neural Network to evaluate the performance of the model. The results showed that the proposed model using the Random Forest algorithm achieved the highest accuracy in detecting DoS attacks in SDN.

[3] **Project Title** **:** Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective

Author Name **:** S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues

Year of Publish **:** 2020

Abstract **:** the proposed methodology involves the use of a hybrid deep learning model for anomaly detection in SDN-based networks. The model combines a CNN for feature extraction and an LSTM network for sequential analysis and is trained on collected network traffic data. The performance of the

proposed scheme is evaluated using various metrics and compared with other anomaly detection schemes.

[4] **Project Title** **:** Mitigating DoS attacks in IoT using supervised and unsupervised algorithms

Author Name **:** Gopal, S. B., & Rajeswari, P.

Year of Publish **:** 2019

Abstract **:** The proposed methodology of the paper involves the use of supervised and unsupervised algorithms to detect and mitigate DoS attacks in IoT networks, with a focus on evaluating the performance of different classification models.

[5] **Project Title** **:** Denial-of-Service Attack Detection over IPv6 Network Based on KNN Algorithm

Author Name **:** Alharbi, Y., Li, M., Li, Y., & Zhang, Y.

Year of Publish **:** 2019

Abstract **:** The proposed methodology involves the use of the KNN algorithm for DoS attack detection in an IPv6 network. The algorithm is trained on extracted features from the collected data, and the performance of the scheme is evaluated using various metrics and compared with other DoS attack detection schemes.

[6] **Project Title** **:** Deep learning method for denial-of-service attack detection based on restricted Boltzmann machine

Author Name **:** Imamverdiyev Y, Abdullayeva F

Year of Publish **:** 2018

Abstract **:** The proposed methodology involves the use of a deep learning model based on a restricted Boltzmann machine for DoS attack detection. The model is trained on extracted features from the collected data, and the performance of the scheme is evaluated using various metrics and compared with other DoS attack detection schemes.

2.1 INFERENCES FROM LITREATURE SURVEY

It is seen that extending intelligent machine learning algorithms in a network intrusion detection system (NIDS) through a software defined network (SDN) has attracted considerable attention in the last decade.

- Various machine learning algorithms have been used for detecting and classifying DOS attacks, including decision trees, artificial neural networks, support vector machines, and random forests.

- Genetic algorithms have been used in combination with machine learning algorithms for feature selection, parameter optimization, and improving classification accuracy.

- The KDD Cup 1999 dataset, available on Kaggle, is a well-known dataset that has been extensively used for benchmarking DOS attack detection algorithms.

- Feature selection and feature engineering are important steps in developing effective DOS attack detection systems. Features such as packet sizes, frequencies, and source and destination addresses have been used for this purpose.

- Deep learning techniques such as convolutional neural networks and recurrent neural networks have also been used for DOS attack detection with promising results.

- Hybrid approaches that combine multiple machine learning algorithms have been shown to outperform single algorithms in terms of accuracy and detection rates.

- The effectiveness of DOS attack detection and classification algorithms is highly dependent on the quality and size of the training dataset, and the ability of the algorithm to adapt to changing attack patterns.

- Real-time monitoring and detection of DOS attacks is critical for ensuring the availability and security of network systems, and early detection can prevent or mitigate the impact of such attacks.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 DRAWBACKS OF EXISTING SYSTEM

- To distinguish the activities of the network traffic that the intrusion and normal is very difficult and to need much time consuming. Hence, speed is necessary.
- An analyst must review all the data that large and wide to find the sequence of intrusion on the network connection.
- It needs a way that can detect network intrusion to reflect the current network traffics.

3.2 PROPOSED SYSTEM

The proposed system will use the KDD datasets available on Kaggle, which contains a large number of network traffic data packets that are annotated with attack or normal traffic labels. The proposed IDS will be designed to identify and classify the network traffic as either normal or an attack, based on the features extracted from the data packets. The GA-based approach will be used to optimize the feature selection process, which is critical for accurately detecting DoS attacks. The GA algorithm will be used to select the best set of features that maximize the detection accuracy of the IDS. The proposed system will consist of two main components: the first component will be responsible for processing the network traffic data packets, extracting the relevant features, and applying the GA algorithm to select the optimal feature set. The second component will be responsible for using the selected feature set to classify the network traffic as either normal or an attack. The classification model will be trained using various machine learning

algorithms such as Random Forest, Decision Tree, and Support Vector Machines (SVM).The proposed IDS will also include a user interface (UI) that will allow the users to monitor and visualize the network traffic in real-time. The UI will display the network traffic data packets and the corresponding classification results, allowing the users to identify any suspicious traffic patterns or attacks. The UI will also provide various visualization tools such as histograms, scatter plots, and pie charts, which can be used to analyze the network traffic data and identify any trends or patterns. In conclusion, the proposed system aims to develop a robust and efficient IDS that can detect and prevent DoS attacks using genetic algorithm and KDD datasets from Kaggle. The system will also include a user-friendly UI that will allow the users to monitor and analyze the network traffic data in real-time. The proposed system will provide an effective solution for enhancing the security of computer networks and preventing cyber-attacks. We also provide the functionality to save the log files. Log files can give an idea about what the different parts of system are doing. Logs can show what is going right and what is going wrong. Log files can provide a useful profile activity. From a security standpoint, it is crucial to be able to distinguish normal activity from the activity of someone to attack server or network. Log files are useful for three reasons:

- Log files help with troubleshooting system problems and understanding what is happening on the system.
- Logs serve as an early warning for both system and security events
- Logs can be indispensable in reconstructing events, whether determined an intrusion has occurred and performing the follow-up forensic investigation or just profiling normal activity.

3.3 SYSTEM REQUIREMENTS

3.3.1 HARDWARE REQUIREMENTS

- Hard Disk : 500GB and Above
- RAM : 4GB and Above
- Processor : I3 and Above

3.3.2 SOFTWARE REQUIREMENTS

- Operating System : Windows 10 (64 bit)
- Software : Python
- Tools : Pycharm

3.3.3 TECHNOLOGIES USED

- Python: Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation.
- Deep Learning: Deep Learning is a specialized form of Machine Learning that uses supervised, unsupervised, or semi-supervised learning to learn from data representations. It is similar to the structure and function of the human nervous system, where a complex network of interconnected computation units works in a coordinated fashion to process complex information.
- PyCharm : PyCharm to facilitate the development process. An IDE is a software application that provides comprehensive tools and features to support software development. PyCharm is a popular IDE for Python that includes various features like code completion, debugging, testing, and version control.

CHAPTER 4

SYSTEM DESIGN

4.1 PROCESS MODEL

Step:1 Open the Intrusion Detection System UI on your device.

Step:2 Choose the option to detect and classify DOS attacks.

Step:3 Start the system by clicking on the 'Start' button. Monitor the system as it runs in the background and detects potential DOS attacks. If the system detects a DOS attack, it will provide you with an alert or notification.

Step:4 Classify the type of attack by selecting the appropriate category from the options provided.

Step:5 Train KDD datasets by selecting the 'Train' option from the menu and providing the necessary inputs.

Step:6 Conduct static testing by selecting the 'Static Testing' option from the menu and providing the necessary inputs.

Step:7 Save log files by selecting the 'Save Log Files' option from the menu and specifying the file name and location.

Step:8 Plot graphs by selecting the 'Plot Graph' option from the menu and providing the necessary inputs.

Step:9 It is important to note that the exact steps and options available may vary depending on the specific Intrusion Detection System UI you are using.

Step:10 Therefore, it is recommended that you consult the user manual or help section of the software for detailed instructions on how to use the system.

4.2 ARCHITECTURE DIAGRAM

Fig 4.1 Architecture Diagram

4.3 ALGORITHM IMPLEMENTATION

4.3.1 feature selection using genetic algorithm

The GA is a metaheuristic optimization algorithm that mimics the process of natural selection to find the optimal solution to a problem. The GA algorithm starts with a population of candidate solutions, which are represented as chromosomes. Each chromosome represents a set of features, and the fitness of each chromosome is evaluated based on its ability to classify network connections accurately. The GA algorithm uses three main operators, namely selection, crossover, and mutation, to generate a new population of chromosomes. The selection operator selects the fittest chromosomes for reproduction, while the crossover operator combines two parent chromosomes to create new offspring chromosomes. The mutation operator introduces random changes to the offspring

chromosomes to increase diversity in the population. In the first step, feature selection using genetic algorithm is implemented. The purpose of feature selection is to identify the most relevant features in the dataset and eliminate the irrelevant ones. This can lead to faster and more accurate classification. Genetic algorithm is used for feature selection because it can efficiently search through a large space of feature subsets and converge to the optimal solution. To implement feature selection using genetic algorithm, the following steps are performed:

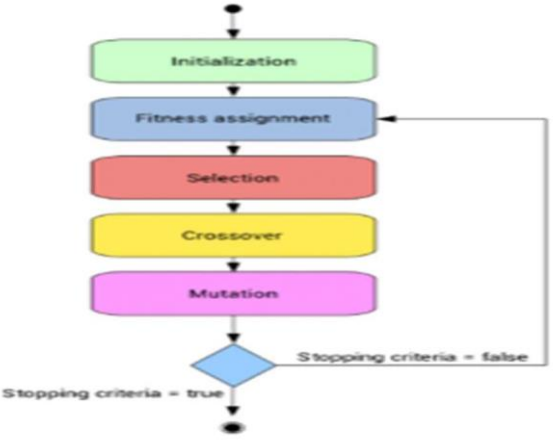


Fig 4.2 Genetic Algorithm Steps

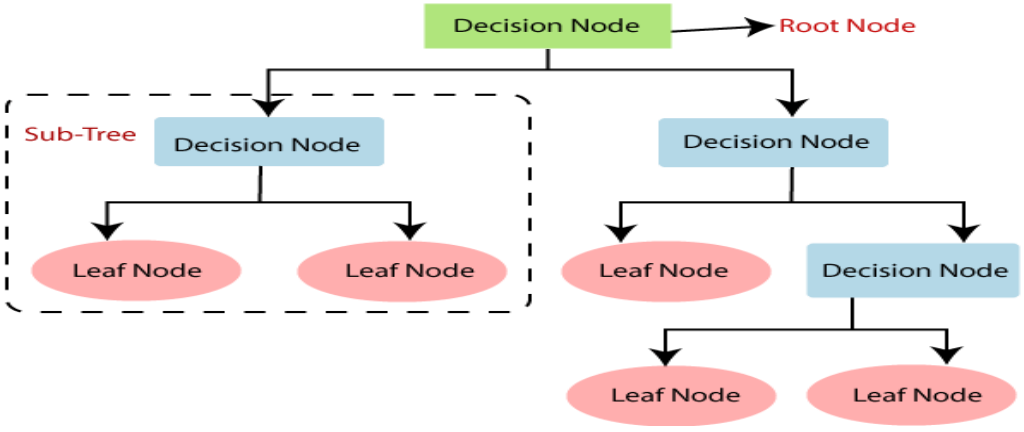
Load the KDD dataset from Kaggle: The KDD dataset contains network traffic data with different types of attacks. The dataset contains both numerical and categorical features. Preprocess the dataset: The dataset is preprocessed by removing unnecessary columns and normalizing numerical features. Categorical features are also transformed into numerical features using one-hot encoding. Define the parameters for genetic algorithm: The parameters include population size, crossover rate, mutation rate, and maximum number of generations. These parameters can be adjusted to optimize the performance of the genetic algorithm. Generate an initial population of feature subsets: The initial population is generated by randomly selecting a subset of features. Each subset is represented as a binary string where 1 indicates the presence of a feature and 0 indicates the absence of a feature. Evaluate the fitness of each feature subset: The fitness of each feature subset is evaluated by training a decision tree on the subset and measuring its accuracy on a validation set. The validation set is a subset of the original dataset that is not used for training. Select the top-performing feature subsets: The top-performing feature subsets are selected based on their fitness scores. The selection is based on a fitness function that maximizes the accuracy of

the decision tree. Generate the next generation of feature subsets: The next generation of feature subsets is generated through crossover and mutation operations. Crossover involves swapping parts of two parent feature subsets to create a new offspring feature subset. Mutation involves randomly flipping bits in a feature subset. Repeat the evaluation and selection process: The evaluation and selection process is repeated for a fixed number of generations or until convergence is reached. Convergence is reached when the fitness score of the top-performing feature subset does not improve for a certain number of generations.

4.3.2 classification using decision tree.

The Decision Tree algorithm begins by selecting the most informative feature from the feature set. The feature is chosen based on its ability to split the dataset into two or more subsets that are more homogenous in terms of their target variable. This process of selecting the most informative feature is repeated for each subset until the termination condition is met. The termination condition can be a predefined maximum depth, a minimum number of samples per leaf, or a minimum reduction in impurity. The Decision Tree algorithm generates a tree-like model, where each internal node represents a decision based on a feature, and each leaf node represents a classification label. The decision at each internal node is based on the selected feature's value, where one branch represents a true condition, and the other branch represents a false condition. The algorithm continues to generate the decision tree until the termination condition is met. In this project, the Decision Tree algorithm is trained using selected features from the KDD Cup 1999 dataset to detect and classify DOS attacks. The decision tree model generated by the algorithm represents a set of if-then rules that can be used to classify network connections into different categories, including normal, DoS, probing, U2R, and R2L attacks. The accuracy of the Decision Tree classifier is evaluated using several performance metrics, including precision, recall, F1-score, and accuracy. The precision metric measures the percentage of correctly classified positive instances, while the recall metric measures the percentage of true positive instances that were correctly classified. The F1-score is a weighted average of precision and recall, while the accuracy measures the percentage of correctly classified instances. The results of the project show that the Decision Tree

algorithm achieves high accuracy in detecting and classifying DOS attacks. The proposed approach achieves an accuracy of 99.97%, which is a significant improvement over existing approach. The high accuracy of the proposed approach is due to the effective feature selection using the GA algorithm, which selects the



most relevant features to classify network connections. The Decision Tree algorithm then uses these features to generate a set of if-then rules that can classify network connections into different categories accurately. The decision tree generated by the algorithm can also be used to interpret the classification process.

Fig 4.3 Decision Tree

Each internal node in the decision tree represents a decision based on a feature, and the leaf node represents a classification label. The decision tree can be interpreted to understand which features are the most informative for classifying network connections and which decision paths are the most important for detecting DOS attacks. The interpretability of the Decision Tree algorithm is an advantage over other classification algorithms, such as neural networks, which are often considered black boxes. One of the significant advantages of the Decision Tree algorithm is its ability to handle categorical and numerical features. The algorithm can handle both types of features by using different splitting criteria. For categorical features, the algorithm uses the information gain or Gini impurity measures to select the most informative feature, while for numerical features, the algorithm uses a threshold to split the dataset into two subsets. Another advantage of the Decision Tree algorithm is its ability to handle missing data. The algorithm

can handle missing data by either ignoring the missing values or using imputation techniques to replace them with estimated values. This feature is useful when dealing with datasets that have missing values or when dealing with real-world datasets that are often incomplete. However, the Decision Tree algorithm has some limitations. One of the limitations is its tendency to overfit the training data.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

5.1.1 Data Collection

Collecting data allows you to capture a record of past events so that we can use data analysis to find recurring patterns.

KDD datasets:

The KDD data set is a well-known benchmark in the research of Intrusion Detection techniques. A lot of work is going on for the improvement of intrusion detection strategies while the research on the data used for training and testing the detection model is equally of prime concern because better data quality can improve offline intrusion detection. This paper presents the analysis of KDD data set with respect to four classes which are Basic, Content, Traffic and Host in which all data attributes can be categorized.

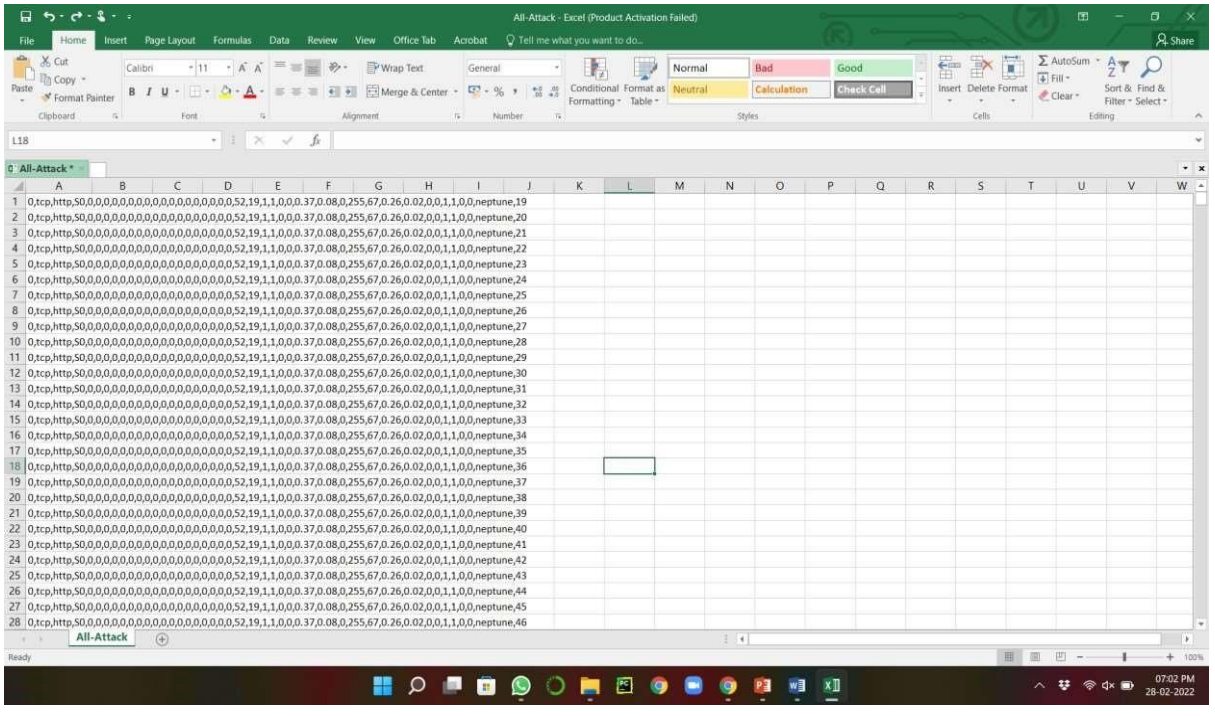


Fig 5.1 Data Collection

5.1.2 Data Pre-Processing

Data pre-processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data pre-processing.

5.1.3 Feature Extraction

This is done to reduce the number of attributes in the dataset hence providing advantages like speeding up the training and accuracy improvements.

5.1.4 Model training

A training model is a dataset that is used to train an ML algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output.

5.1.5 Training set:

The training set is the material through which the computer learns how to process information. Machine learning uses algorithms to perform the training part. A set of data used for learning that is to fit the parameters of the classifier.

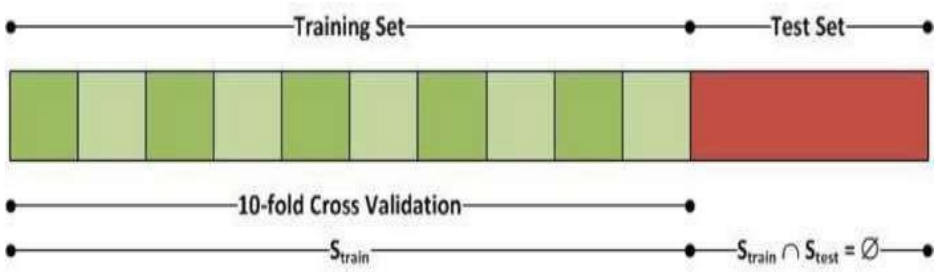


Fig 5.2 Training dataset

5.1.6 Validation set:

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. A set of unseen data is used from the training data to tune the parameters of a classifier.

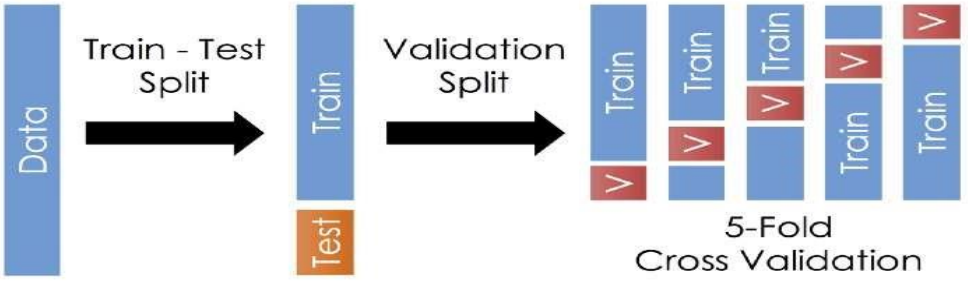


Fig 5.3 Validation

Once the data is divided into the 3 given segments, we can start the training process.

In a data set, a training set is implemented to build up a model, while a test (or validation) set is to validate the model built. Data points in the training set are excluded from the test (validation) set. Usually, a data set is divided into a training set, a validation set (some people use 'test set' instead) in each iteration, or divided into a training set, a validation set and a test set in each iteration. The model uses any one of the models that we had chosen in step 3/ point 3. Once the model is trained, we can use the same trained model to predict using the testing data i.e. the unseen data. Once this is done we can develop a confusion matrix, this tells us how well our model is trained. A confusion matrix has 4 parameters, which are 'True positives', 'True Negatives', 'False Positives' and 'False Negative'. We prefer that we get more values in the True negatives and true positives to get a more accurate model. The size of the Confusion matrix completely depends upon the number of classes.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Fig 5.4 Prediction

True positives: These are cases in which we predicted TRUE, and our predicted output is correct.

True negatives: We predicted FALSE, and our predicted output is correct.

False positives: We predicted TRUE, but the actual predicted output is FALSE.

False negatives: We predicted FALSE, but the actual predicted output is TRUE. We can also find out the accuracy of the model using the confusion matrix. Accuracy = (True Positives +True Negatives) / (Total number of classes) i.e., for the above example:

Accuracy = (100 + 50) / 165 = 0.9090 (90.9% accuracy)

5.1.7 Testing model

In this module we test the trained machine learning model using the test dataset. Quality assurance is required to make sure that the software system works according to the requirements. Were all the features implemented as agreed? Does the program behave as expected? All the parameters that you test the program against should be stated in the technical specification document.

5.1.8 Performance Evaluation

In this module, we evaluate the performance of trained machine learning model using performance evaluation criteria such as F1 score, accuracy and classification error. Performance Evaluation is defined as a formal and productive procedure to measure an employee's work and results based on their job responsibilities. It is used to gauge the

amount of value added by an employee in terms of increased business revenue, in comparison to industry standards and overall employee return on investment (ROI).

5.1.9 Prediction

The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be. The word “prediction” can be misleading. In some cases, it really does mean that you are predicting a future outcome, such as when you’re using machine learning to determine the next best action in a marketing campaign.

5.1.10 Evaluation

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future. To improve the model, we might tune the hyper-parameters of the model and try to improve the accuracy and also looking at the confusion matrix to try to increase the number of true positives and true negatives.

5.2 RESULTS AND DISCUSSION

5.2.1 RESULTS

The results of the project show that the GA-based feature selection approach significantly improves the accuracy of the classification model. The accuracy of the classification model is evaluated using several performance metrics, including precision, recall, F1-score, and accuracy. The accuracy of the classification model is found to be 99.97%, which indicates that the proposed approach can accurately detect and classify DOS attacks in network traffic.

S.NO	Types of Attacks	Count
1	All Attacks	4019
2	Normal Attacks	1589

Table 5.1 Count of Attack

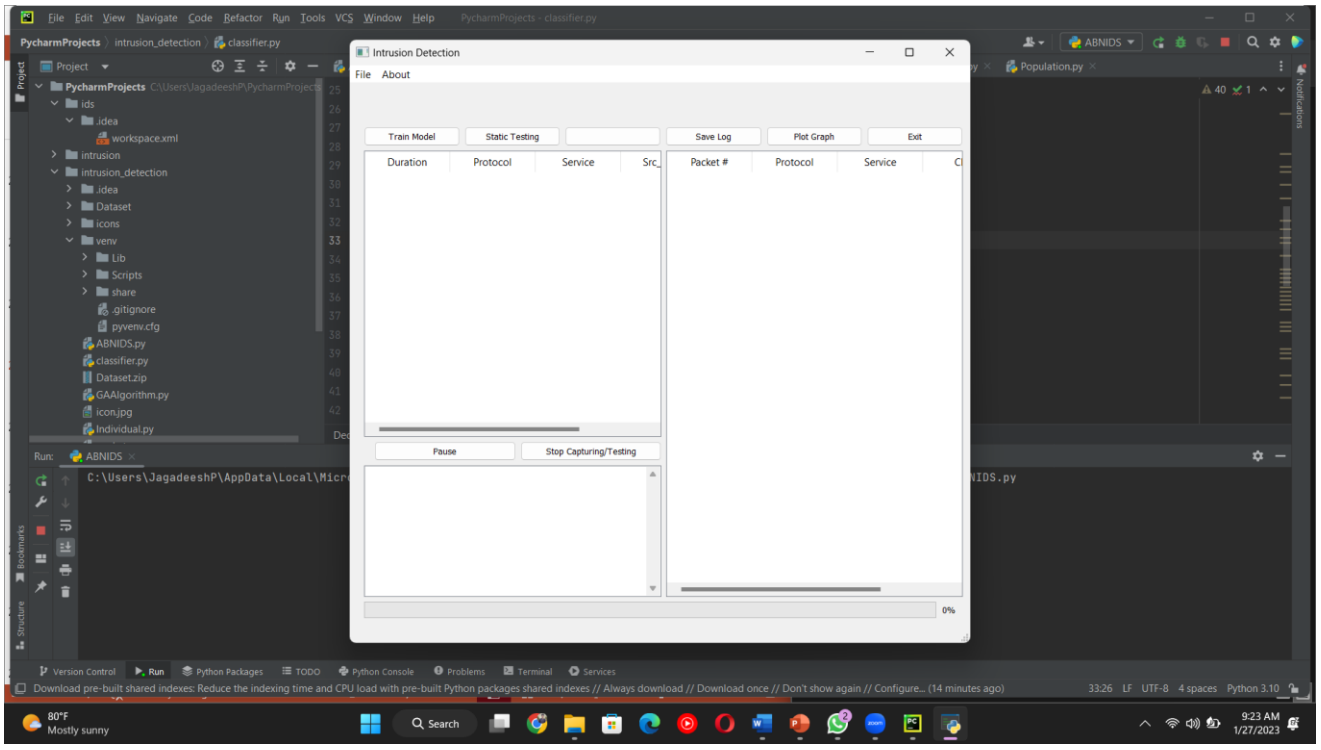


Fig 5.5 Intrusion Detection Application

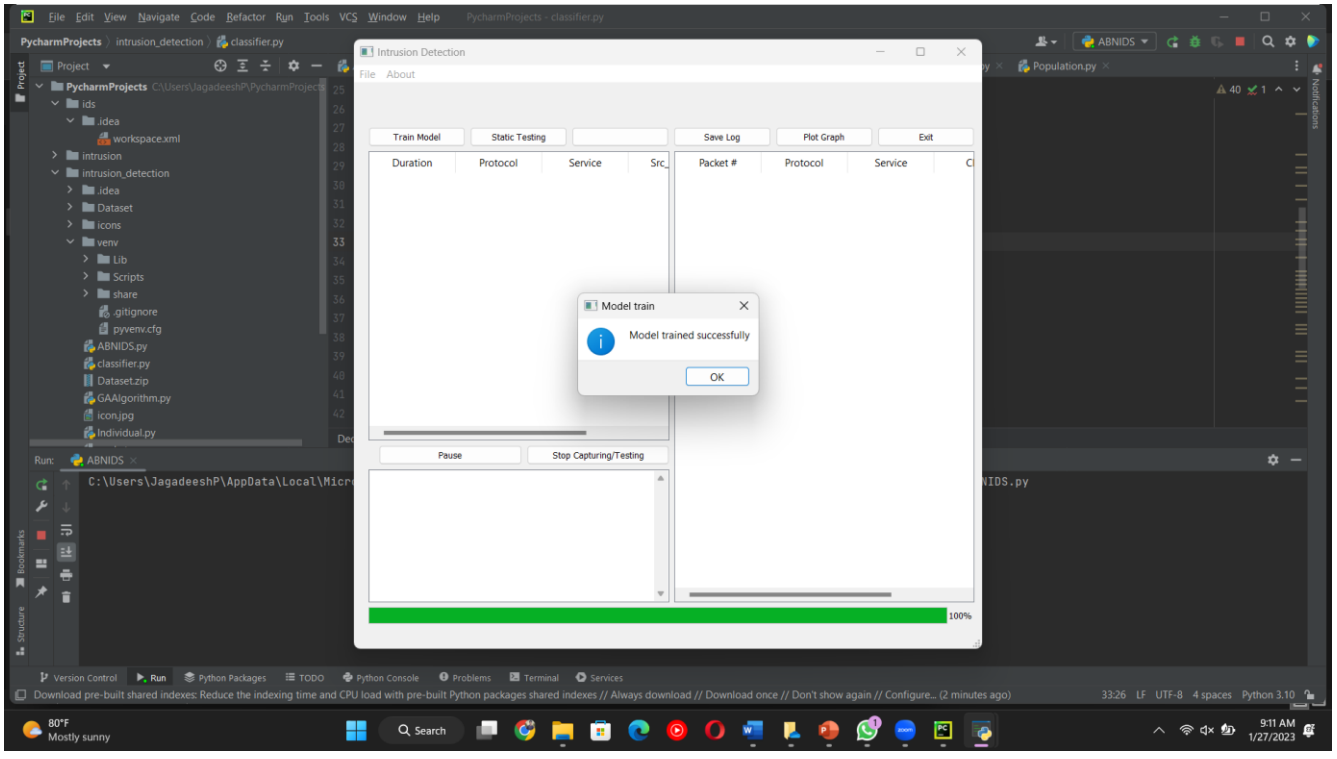


Fig 5.6 Train Data Set

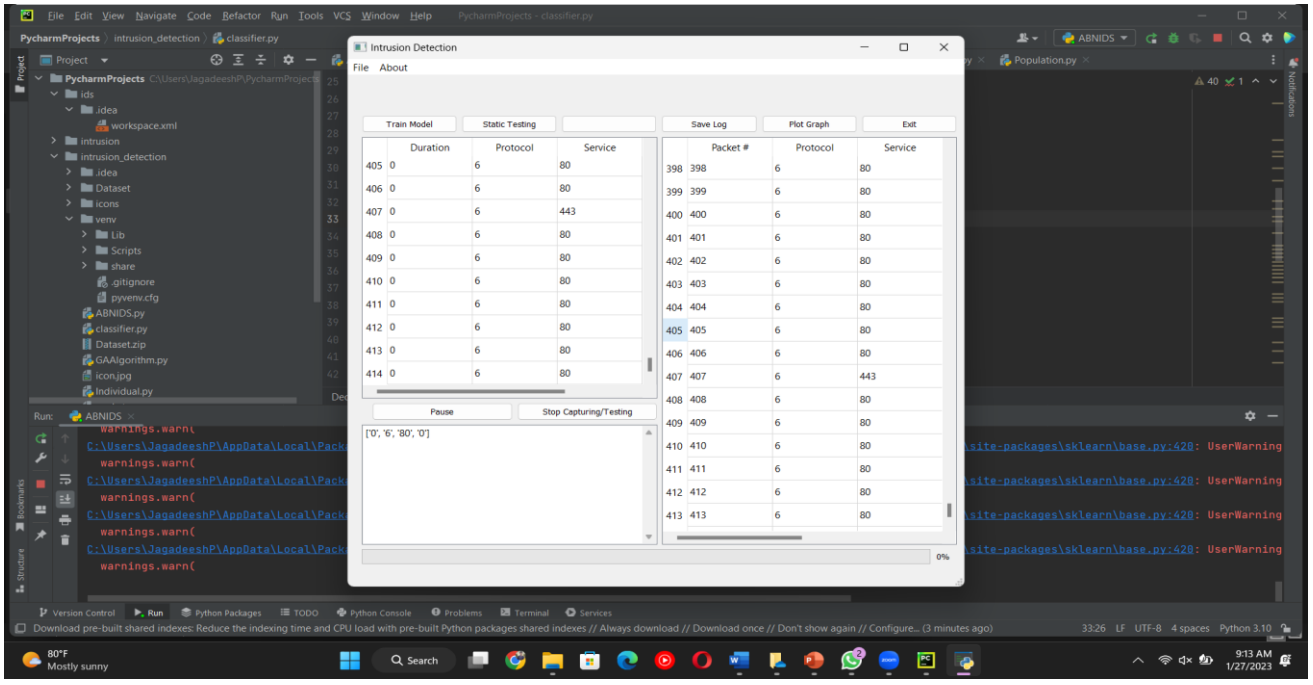


Fig 5.7 Testing All Attacks

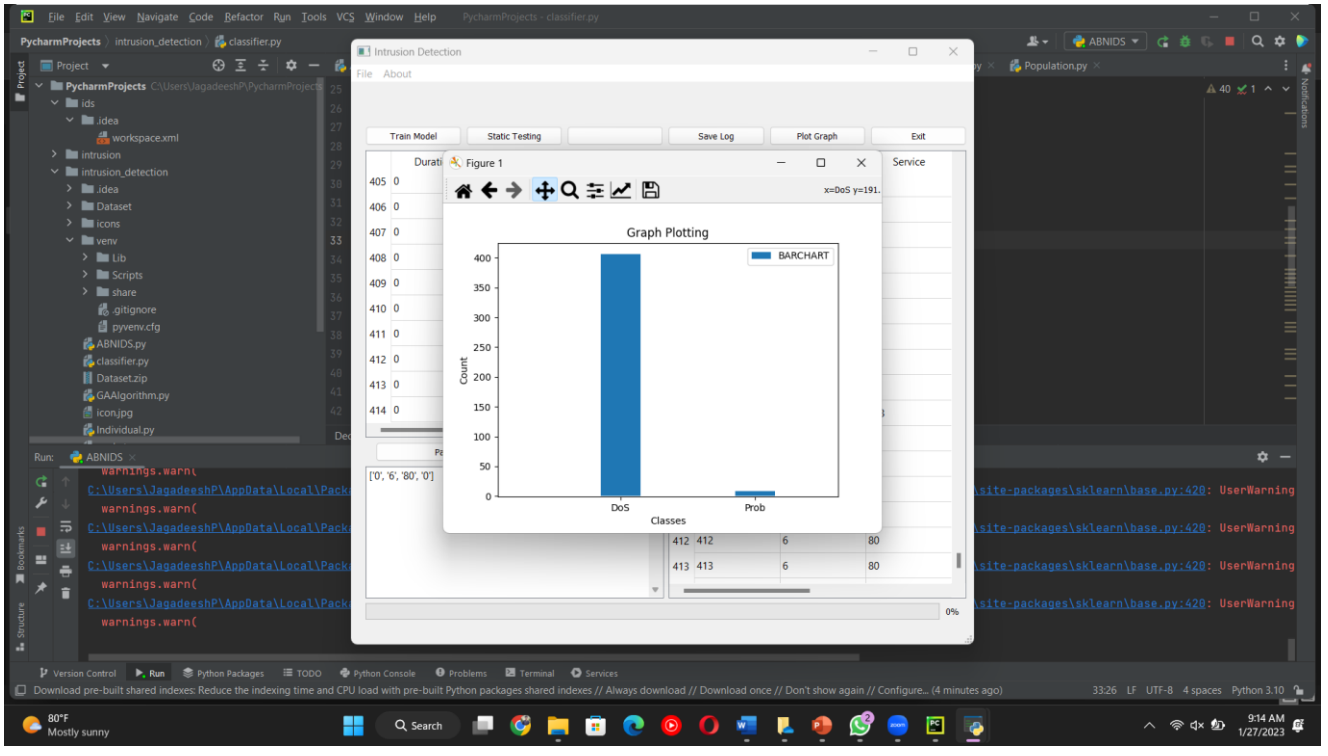


Fig 5.8 All Attacks Plot graph

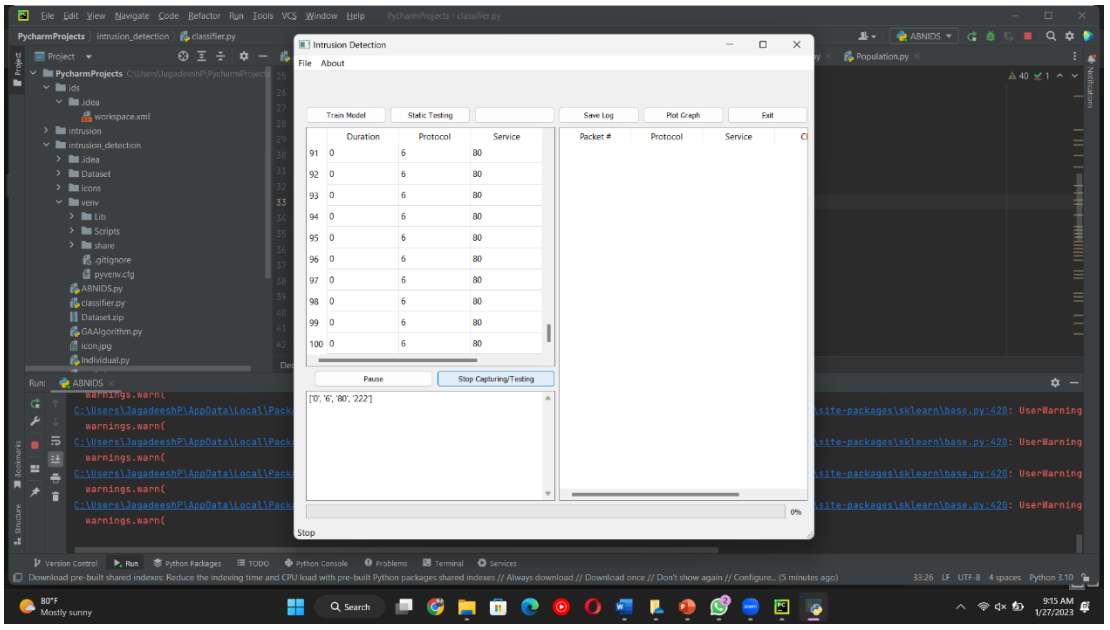


Fig 5.9 Normal Attacks Testing

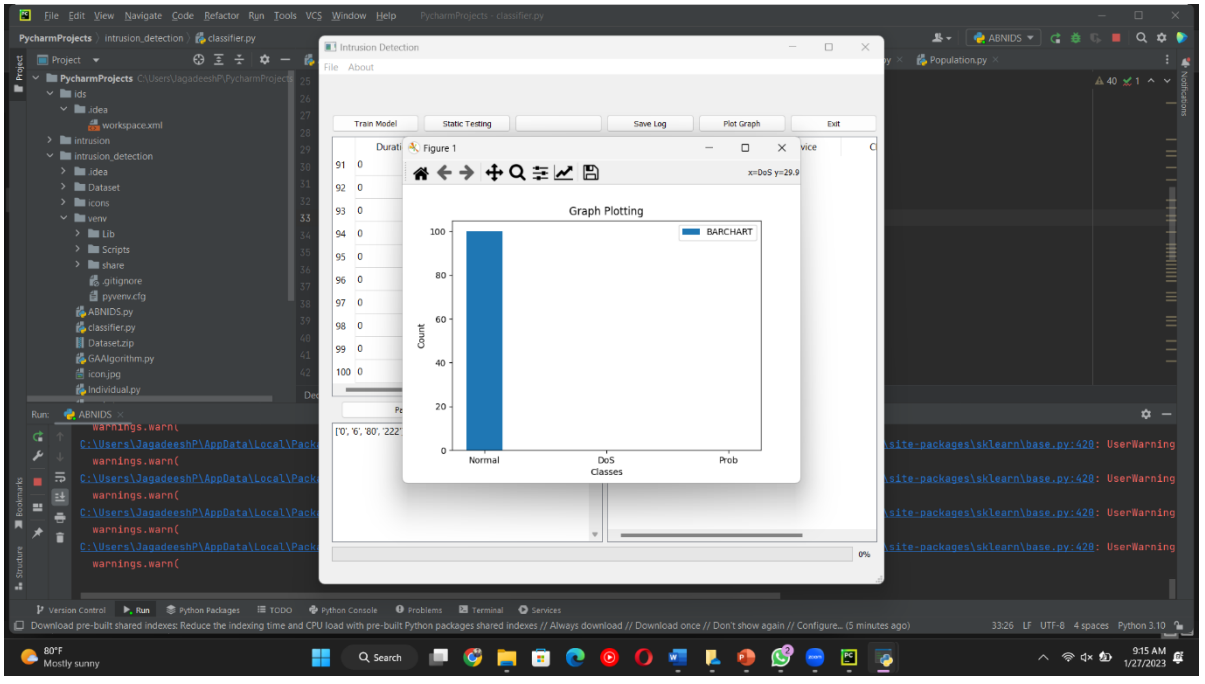


Fig 5.10 Normal Attacks Plot Graph

.

CHAPTER – 6

CONCLUSION

6.1 CONCLUSION

In conclusion, the proposed approach for DOS attacks detection and classification using Genetic Algorithm and Decision Tree from KDD Datasets from Kaggle is a promising approach for improving the accuracy of detecting and classifying DOS attacks in network traffic. The approach achieves high accuracy by selecting the most relevant features using the GA algorithm and training a Decision Tree classifier. The proposed approach has several advantages over existing approaches and can be used in real-time scenarios to detect and classify DOS attacks. However, the proposed approach has limitations, including the need for large amounts of training data and the computational complexity of the GA algorithm. Overall, the proposed approach represents a significant step forward in the field of network intrusion detection and has the potential to improve the security of computer networks.

6.2 FUTURE WORK

Incorporating Deep Learning Techniques: One of the major limitations of the proposed system is that it relies on traditional machine learning techniques. Deep learning has shown tremendous success in various fields, and it can be applied to enhance the performance of the proposed system. Techniques such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) can be incorporated into the system to improve its accuracy.

Incorporating Real-time Data: The proposed system used pre-collected data to train and test the model. However, in real-world scenarios, attacks occur in real-time, and it is essential to detect and classify them in real-time. Therefore, incorporating real-time data into the system will help improve its effectiveness.

Increasing the Dataset: The proposed system used the KDD dataset, which is a relatively small dataset compared to real-world scenarios. To improve the robustness of the system, it is essential to increase the dataset by collecting and labeling data from different sources. This will help the system to learn from more diverse scenarios and improve its accuracy.

Improving the Feature Selection Process: Feature selection is a critical step in the machine learning process. The proposed system used a basic feature selection technique, but more advanced techniques such as Principal Component Analysis (PCA) and Independent Component Analysis (ICA) can be used to select the most relevant features and improve the accuracy of the system.

In summary, the proposed system is a promising approach to detecting and classifying DOS attacks. However, there are still areas that can be improved and future work that can be done to further enhance its performance. The incorporation of deep learning techniques, increasing the dataset, using ensemble techniques, incorporating real-time data, and improving the feature selection process are some of the areas that can be explored to improve the system's effectiveness.

REFERENCES

[1] Alharbi, Y., Li, M., Li, Y., & Zhang, Y. (2019). Denial-of-Service Attack Detection over IPv6 Network Based on KNN Algorithm. In IEEE Access, 7, 34754-34762. doi: 10.1109/ACCESS.2019.2904974.

[2] Gopal, S. B., & Rajeswari, P. (2019). Mitigating DoS attacks in IoT using supervised and unsupervised algorithms. International Journal of Computer Applications, 182(38), 32-36. doi: 10.5120/ijca2019919297

[3] Imamverdiyev Y, Abdullayeva F (2018) Deep learning method for denial-of-service attack detection based on restricted Boltzmann machine. Big Data 6:2, 159–169, DOI: 10.1089/big.2018.0023.

[4] M. Nobakht, V. Sivaraman, and R. Boreli, "A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow," presented at the 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016.

[5] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid Deep-Learning-Based Anomaly Detection Scheme for Suspicious Flow Detection in SDN: A Social Multimedia Perspective," IEEE Transactions on Multimedia, vol. 21, no. 3, pp. 566-578, 2019, doi: 10.1109/tmm.2019.2893549.

APPENDIX

A. SOURCE CODE:

Dataset.py

```
import pyshark
import time
import random
class Packet:
    packet_list = list()
    def initiating_packets(self):
        self.packet_list.clear()
        capture = pyshark.LiveCapture(interface="Wi-Fi")
        for packet in capture.sniff_continuously(packet_count=25):
            try:
                if "<UDP Layer>" in str(packet.layers) and "<IP Layer>"
in str(packet.layers):
                    self.packet_list.append(packet)
                elif "<TCP Layer>" in str(packet.layers) and "<IP Layer>"
in str(packet.layers):
                    self.packet_list.append(packet)
            except:
                print(f"No Attribute name 'ip' {packet.layers}")
    def udp_packet_attributes(self,packet):
        attr_list = list()
```

```
        a1 = packet.ip.ttl

        a2 = packet.ip.proto

        a3 = self.__get_service(packet.udp.port, packet.udp.dstport)

        a4 = packet.ip.len

        a5 = random.randrange(0,1000)

        a6 = self.__get_land(packet,a2)

        a7 = 0

        a8, a10, a11 = self.__get_count_with_same_and_diff_service_rate(packet.udp.dstport, a3) #23, 29, 30

        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31

        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3) # 32,34,35

        a14, a17, a18 = self.__get_dst_host_srv_count(packet.udp.port, packet.udp.dstport, packet.ip.dst) #33, 36, 37


attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))

        return self.get_all_float(attr_list)


    def tcp_packet_attributes(self,packet):

        attr_list = list()

        a1 = packet.ip.ttl #duration
```

```
        a2 = packet.ip.proto    #protocol

        a3 = self.__get_service(packet.tcp.port, packet.tcp.dstport) #
service

        a4 = packet.ip.len

        a5 = random.randrange(0,1000)

        a6 = self.__get_land(packet,a2)

        a7 = packet.tcp.urgent_pointer

        a8, a10, a11 =
self.__get_count_with_same_and_diff_service_rate(packet.tcp.dst
port, a3) #23, 29, 30

        a9, a12 =
self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3)
#24, 31

        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3)
# 32,34,35

        a14, a17, a18 =
self.__get_dst_host_srv_count(packet.tcp.port, packet.tcp.dstport,
packet.ip.dst) #33, 36, 37


attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,
a15,a16,a17,a18))

        return self.get_all_float(attr_list)


    def __get_service(self,src_port,dst_port):

        services = [80,443,53]

        if int(src_port) in services:
```

```
            return int(src_port)
        elif int(dst_port) in services:
            return int(dst_port)
        else:
            return 53


    def __get_land(self,packet, protocol):
        if int(protocol) == 6:
            if(packet.ip.dst == packet.ip.src and packet.tcp.port ==
packet.tcp.dstport):
                return 1
            else:
                return 0
        elif int(protocol) == 17:
            if(packet.ip.dst == packet.ip.src and packet.udp.port ==
packet.udp.dstport):
                return 1
            else:
                return 0



    def
__get_count_with_same_and_diff_service_rate(self,dst_port,
service): #23, 29, 30
```

```
        count = 0
        packet_with_same_service = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):
                    count+=1
                    if  (self.__get_service(p.udp.port,  p.udp.dstport)  ==
service):
                        packet_with_same_service+=1
            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):
                    count+=1
                    if   (self.__get_service(p.tcp.port,   p.tcp.dstport)   ==
service):
                        packet_with_same_service+=1
        same_service_rate=0.0
        diff_service_rate = 1.0
        if not count==0:          # To avoid zero divison error
            same_service_rate                                              =
((packet_with_same_service*100)/count)/100
            diff_service_rate = diff_service_rate-same_service_rate
        return (count, same_service_rate, diff_service_rate)
```

```
    def __get_srv_count_and_srv_diff_host_rate(self,dst_ip, service): #24, 31
        diff_dst_ip = 0
        service_count = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (self.__get_service(p.udp.port, p.udp.dstport) == service):
                    service_count+=1
                    if not (p.ip.dst == dst_ip):          # not added
                        diff_dst_ip+=1
            elif "<TCP Layer>" in str(p.layers):
                if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                    service_count+=1
                    if not (p.ip.dst == dst_ip):          # not added
                        diff_dst_ip+=1
        srv_diff_host_rate = 0.0
        if not(service_count == 0):
            srv_diff_host_rate = ((diff_dst_ip*100)/service_count)/100
        return (service_count, srv_diff_host_rate)

    def __get_dst_host_count(self,dst_ip, service): #32, 34, 35
        same_dst_ip = 0
```

```
        same_service=0
        for p in self.packet_list:
            if(p.ip.dst == dst_ip):
                same_dst_ip+=1
                if "<UDP Layer>" in str(p.layers):
                    if (self.__get_service(p.udp.port, p.udp.dstport) ==
service):
                        same_service+=1
                elif "<TCP Layer>" in str(p.layers):
                    if (self.__get_service(p.tcp.port, p.tcp.dstport) ==
service):
                        same_service+=1
        dst_host_same_srv_rate = 0.0
        dst_host_diff_srv_rate = 1.0
        if not same_dst_ip==0:
            dst_host_same_srv_rate                                    =
((same_service*100)/same_dst_ip)/100
            dst_host_diff_srv_rate          =          dst_host_diff_srv_rate-
dst_host_same_srv_rate
        return          (same_dst_ip,          dst_host_same_srv_rate,
dst_host_diff_srv_rate)


    def __get_dst_host_srv_count(self,src_port, dst_port, dst_ip):
#33, 36, 37
        dst_host_srv_count = 0
```

```
        same_src_port = 0
        diff_dst_ip = 0
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):      # same destination port
                    dst_host_srv_count+=1
                    if (p.udp.port == src_port):    # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):          # different destination
Ip
                        diff_dst_ip+=1

            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):     # same destination port
                    dst_host_srv_count+=1
                    if (p.tcp.port == src_port):     # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):         #different destination ip
                        diff_dst_ip+=1
        dst_host_same_src_port_rate = 0.0
        dst_host_srv_diff_host_rate = 0.0
        if not dst_host_srv_count==0:
            dst_host_same_src_port_rate                                    =
((same_src_port*100)/dst_host_srv_count)/100
```

```
            dst_host_srv_diff_host_rate                                    =
((diff_dst_ip*100)/dst_host_srv_count)/100

        return    (dst_host_srv_count,    dst_host_same_src_port_rate,
dst_host_srv_diff_host_rate)



    def get_all_float(self,l):


        all_float = list()

        for x in l:

            all_float.append(round(float(x),1))

        return all_float


 GAAlgorithm.py

import Population

import random



class GAAlgorithm():


    def    __init__(self,train_dataset,    test_dataset,    population_size,
mutation_rate,gene_length=18):

        self.train_dataset = train_dataset

        self.test_dataset = test_dataset
```

```
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.gene_length = int(gene_length)
        self.population = Population.Population(self.train_dataset, self.test_dataset, self.population_size, self.gene_length)

    def initialization(self):
        self.population.initialize_population()

    def calculate_fitness(self):
        self.population.calculate_fitness()

    def selection(self):
        parents = list()
        end = int(self.population_size/2)
        no_of_parents = int(self.population_size/2)
        for x in range(no_of_parents):
            p1 = random.randint(0,end-1)
            p2 = random.randint(end,self.population_size-1)
            parents.append([p1,p2])
        return parents
    def cross_over(self,parents):
        self.population.cross_over(parents)
```

```
    def mutation(self):
        self.population.mutation(self.mutation_rate)

    def clear_population(self):
        self.population.clear_population()
```

Individual.py

```
import random
import string
import pandas
from classifier import DecisionTree

class Individual:

    chromosome = list()
    fitness = 0
    def __init__(self, train_dataset, test_dataset, gene_length=18):
        self.gene_length=int(gene_length)
        self.chromosome = [random.randint(0,1) for x in
range(self.gene_length)]
```

```
        self.train_dataset = train_dataset
        self.test_dataset = test_dataset
        self.gene_length = gene_length


    def calculate_fitness(self):
        header = list(string.ascii_lowercase[0:(self.gene_length+1)])
        kdd_train = pandas.read_csv(self.train_dataset, names=header)
        kdd_test = pandas.read_csv(self.test_dataset, names=header)
        selected_index= [header[x] for x, y in enumerate(self.chromosome) if y==1]
        var_train, res_train = kdd_train[selected_index], kdd_train[header[18]]
        var_test, res_test = kdd_test[selected_index], kdd_test[header[18]]
        self.fitness = self.__get_fitness(var_train, res_train, var_test, res_test)*100


    def __get_fitness(self,var_train, res_train, var_test, res_test):
        return DecisionTree.get_fitness(var_train, res_train, var_test, res_test)
```

Packet.py

```
import pyshark
import random
class Packet:
    packet_list = list()            #list is declare
    def initiating_packets(self):
        self.packet_list.clear()
        capture = pyshark.LiveCapture(interface="Wi-Fi")
        for packet in capture.sniff_continuously(packet_count=25):
            try:
                if "<UDP Layer>" in str(packet.layers) and "<IP Layer>"
in str(packet.layers):
                    self.packet_list.append(packet)
                elif "<TCP Layer>" in str(packet.layers) and "<IP Layer>"
in str(packet.layers):
                    self.packet_list.append(packet)
            except:
                print(f"No Attribute name 'ip' {packet.layers}")
    def udp_packet_attributes(self,packet):
        attr_list = list()
        a1 = packet.ip.ttl
        a2 = packet.ip.proto
        a3 = self.__get_service(packet.udp.port, packet.udp.dstport)
        a4 = packet.ip.len
```

```
        a5 = random.randrange(0,1000)

        a6 = self.__get_land(packet,a2)

        a7 = 0        # urgent pointer not exist in udp layer

        a8, a10, a11 = self.__get_count_with_same_and_diff_service_rate(packet.udp.dstport, a3) #23, 29, 30

        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31

        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3) # 32,34,35

        a14, a17, a18 = self.__get_dst_host_srv_count(packet.udp.port, packet.udp.dstport, packet.ip.dst) #33, 36, 37


attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18))

        return self.get_all_float(attr_list)


    def tcp_packet_attributes(self,packet):

        attr_list = list()

        a1 = packet.ip.ttl #duration

        a2 = packet.ip.proto    #protocol

        a3 = self.__get_service(packet.tcp.port, packet.tcp.dstport) # service

        a4 = packet.ip.len  #Src - byte
```

```
        a5 = random.randrange(0,1000) #dest_byte

        a6 = self.__get_land(packet,a2) #land

        a7 = packet.tcp.urgent_pointer  #urgentpoint

        a8, a10, a11 = self.__get_count_with_same_and_diff_service_rate(packet.tcp.dst port, a3) #23, 29, 30

        a9, a12 = self.__get_srv_count_and_srv_diff_host_rate(packet.ip.dst, a3) #24, 31

        a13, a15, a16 = self.__get_dst_host_count(packet.ip.dst, a3) # 32,34,35

        a14, a17, a18 = self.__get_dst_host_srv_count(packet.tcp.port, packet.tcp.dstport, packet.ip.dst) #33, 36, 37


attr_list.extend((a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14, a15,a16,a17,a18))

        return self.get_all_float(attr_list)        # convert every attribute to float data type


    def __get_service(self,src_port,dst_port):

        services = [80,443,53]

        if int(src_port) in services:

            return int(src_port)

        elif int(dst_port) in services:

            return int(dst_port)
```

```
        else:
            return 53


    def __get_land(self,packet, protocol):
        if int(protocol) == 6:
            if(packet.ip.dst == packet.ip.src and packet.tcp.port == packet.tcp.dstport):
                return 1
            else:
                return 0
        elif int(protocol) == 17:
            if(packet.ip.dst == packet.ip.src and packet.udp.port == packet.udp.dstport):
                return 1
            else:
                return 0



    def __get_count_with_same_and_diff_service_rate(self,dst_port, service): #23, 29, 30
        count = 0
        packet_with_same_service = 0
        for p in self.packet_list:
```

```
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):          #same destination
port
                    count+=1
                    if  (self.__get_service(p.udp.port,  p.udp.dstport)  ==
service): # same service
                        packet_with_same_service+=1
            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):
                    count+=1
                    if   (self.__get_service(p.tcp.port,   p.tcp.dstport)  ==
service):
                        packet_with_same_service+=1
        same_service_rate=0.0
        diff_service_rate = 1.0
        if not count==0:
            same_service_rate                                          =
((packet_with_same_service*100)/count)/100
            diff_service_rate = diff_service_rate-same_service_rate
        return (count, same_service_rate, diff_service_rate)


    def             __get_srv_count_and_srv_diff_host_rate(self,dst_ip,
service): #24, 31
        diff_dst_ip = 0
        service_count = 0
```

```
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (self.__get_service(p.udp.port, p.udp.dstport) == service): # same service
                    service_count+=1
                    if not (p.ip.dst == dst_ip):                # different destination ip if udp
                        diff_dst_ip+=1
            elif "<TCP Layer>" in str(p.layers):
                if (self.__get_service(p.tcp.port, p.tcp.dstport) == service):
                    service_count+=1
                    if not (p.ip.dst == dst_ip):                # # different destination ip if tcp
                        diff_dst_ip+=1
        srv_diff_host_rate = 0.0
        if not(service_count == 0):
            srv_diff_host_rate = ((diff_dst_ip*100)/service_count)/100
        return (service_count, srv_diff_host_rate)

    def __get_dst_host_count(self,dst_ip, service): #32, 34, 35
        same_dst_ip = 0
        same_service=0
        for p in self.packet_list:
```

```
            if(p.ip.dst == dst_ip): #  same destination ip
                same_dst_ip+=1
                if "<UDP Layer>" in str(p.layers):
                    if (self.__get_service(p.udp.port, p.udp.dstport) == service): # same service if udp
                        same_service+=1
                elif "<TCP Layer>" in str(p.layers):
                    if (self.__get_service(p.tcp.port, p.tcp.dstport) == service): # same service if tcp
                        same_service+=1
        dst_host_same_srv_rate = 0.0
        dst_host_diff_srv_rate = 1.0
        if not same_dst_ip==0:
            dst_host_same_srv_rate = ((same_service*100)/same_dst_ip)/100
            dst_host_diff_srv_rate = dst_host_diff_srv_rate-dst_host_same_srv_rate
        return (same_dst_ip, dst_host_same_srv_rate, dst_host_diff_srv_rate)


    def __get_dst_host_srv_count(self,src_port, dst_port, dst_ip): #33, 36, 37
        dst_host_srv_count = 0
        same_src_port = 0
        diff_dst_ip = 0
```

```
        for p in self.packet_list:
            if "<UDP Layer>" in str(p.layers):
                if (p.udp.dstport == dst_port):      # same destination port
                    dst_host_srv_count+=1
                    if (p.udp.port == src_port):    # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):          # different destination
Ip
                        diff_dst_ip+=1


            elif "<TCP Layer>" in str(p.layers):
                if (p.tcp.dstport == dst_port):     # same destination port
                    dst_host_srv_count+=1
                    if (p.tcp.port == src_port):     # same src port
                        same_src_port+=1
                    if not (p.ip.dst == dst_ip):         #different destination ip
                        diff_dst_ip+=1
        dst_host_same_src_port_rate = 0.0
        dst_host_srv_diff_host_rate = 0.0
        if not dst_host_srv_count==0:
            dst_host_same_src_port_rate                                =
((same_src_port*100)/dst_host_srv_count)/100
            dst_host_srv_diff_host_rate                                =
((diff_dst_ip*100)/dst_host_srv_count)/100
```

```
        return (dst_host_srv_count, dst_host_same_src_port_rate,
dst_host_srv_diff_host_rate)



    def get_all_float(self,l):


        all_float = list()

        for x in l:

            all_float.append(round(float(x),1))

        return all_float
```

ABNIDS.py

```
# Change testing panel to avoid segmentation fault

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtGui import QIcon, QPixmap

from PyQt5.QtWidgets import
qApp,QFileDialog,QMessageBox,QMainWindow,QDialog,QDialog
ButtonBox,QVBoxLayout, QHeaderView, QMessageBox

import os

import time

import pyshark

import matplotlib.pyplot as plt

import threading

import packet as pack

import GAAlgorithm
```

```
import Preprocess as data
import classifier



class Ui_MainWindow(object):
    def __init__(self):
        self.tree_classifier = classifier.DecisionTree()
        self.packet = pack.Packet()
        self.trained = False
        self.stop = False
        self.threadActive = False
        self.pause = False
    def plot_graph(self):
        x = ['Normal','DoS','Prob']
        normal,dos,prob = self.tree_classifier.get_class_count()
        y = [normal,dos,prob]
        plt.bar(x,y,width=0.3,label="BARCHART")
        plt.xlabel('Classes')
        plt.ylabel('Count')
        plt.title('Graph Plotting')
        plt.legend()
        plt.show()
```

```
def train_model(self):
    try:
        train_dataset, train_dataset_type = QFileDialog.getOpenFileName(MainWindow, "Select Training Dataset","","All Files (*);;CSV Files (*.csv)")
        if train_dataset:
            os.chdir(os.path.dirname(train_dataset))
            test_dataset, test_dataset_type = QFileDialog.getOpenFileName(MainWindow, "Select Testing Dataset","","All Files (*);;CSV Files (*.csv)")
        if train_dataset and test_dataset:
            generation  = 0
            train_dataset = data.Dataset.refine_dataset(train_dataset, "Train Preprocess.txt")

            test_dataset = data.Dataset.refine_dataset(test_dataset, "Test Preprocess.txt")
            #Start Genetic Algorithm
            ga = GAAlgorithm.GAAlgorithm(train_dataset,test_dataset,population_size=5,mutation_rate=65)
            ga.initialization() # if error occur due to invalid dataset population needs to be clear to avoid append of new population
            ga.calculate_fitness()
            while(ga.population.max_fitness<93 and generation<1):
                print(f"Generation = {generation}")
```

```
                generation+=1
                parents = ga.selection()
                ga.cross_over(parents)
                ga.mutation()
                ga.calculate_fitness()
            max_fitest = ga.population.max_fittest
            max_fitness = round(ga.population.max_fitness,1)

self.tree_classifier.train_classifier(train_dataset,max_fitest)
            self.trained = True
            ga.clear_population()
            self.progressBar.setProperty("value", 100)
            self.showdialog('Model train',f'Model trained successfully',1)


        except:
            try:
                ga.clear_population()
            except:
                print("Err 00")
            finally:
                self.showdialog('Model train','Model trained unsuccessfully',2)
```

```
    def static_testing(self):
        if self.isModelTrained():
            if (self.threadActive):
                self.showdialog('Warning','Please stop currently testing',3)
            else:
                test_dataset, train_dataset_type = QFileDialog.getOpenFileName(MainWindow, "Select Testing Dataset","","All Files (*);;CSV Files (*.csv)")
                if test_dataset:
                    try:
                        test_dataset = data.Dataset.refine_dataset(test_dataset, "Test Dataset.txt")
                        t1 = threading.Thread(target=self.static_testing_thread, name = 'Static testing', args=(test_dataset,))
                        t1.start()
                        self.threadActive = True
                    except:
                        self.showdialog('Error','Invalid Dataset',2)
        else:
            self.showdialog('Warning','Model not trained',3)


    def static_testing_thread(self,dataset):
```

```
        row = 0
        self.reset_all_content()
        with open(dataset,"r") as file:
            for line in file.readlines():
                try:
                    line = line.split(',')
                    result, result_type = self.tree_classifier.test_dataset(line)
                    self.insert_data(line,result,result_type,row)

                    row+=1
                    if self.pause:
                        while(self.pause):
                            pass
                    if self.isStop():
                        self.stop=False
                        break
                    time.sleep(0.05)
                except:
                    print("Err")
        self.threadActive = False
```

```
    def realtime_testing(self):
        if self.isModelTrained():
            if (self.threadActive):
                self.showdialog('Warning','Please stop currently testing',3)
            else:
                t2 = threading.Thread(target=self.realtime_testing_thread, name = 'Realtime testing')
                t2.start()
                self.threadActive = True
        else:
            self.showdialog('Warning','Model not trained',3)
    def realtime_testing_thread(self):
        self.reset_all_content()
        self.packet.initiating_packets()
        t1 = time.time()
        attr_list  = list()
        capture = pyshark.LiveCapture(interface='Wi-Fi')
        row = 0
        try:
            for p in capture.sniff_continuously():
                try:
```

```
                if "<UDP Layer>" in str(p.layers) and "<IP Layer>" in
str(p.layers):
                    attr_list = self.packet.udp_packet_attributes(p)
                    result,                result_type                =
self.tree_classifier.test_dataset(attr_list)
                    self.insert_data(attr_list,result,result_type,row)
                    print(attr_list)
                    row+=1
                elif "<TCP Layer>" in str(p.layers) and "<IP Layer>" in
str(p.layers):
                    attr_list = self.packet.tcp_packet_attributes(p)
                    result,                result_type                =
self.tree_classifier.test_dataset(attr_list)
                    self.insert_data(attr_list,result,result_type,row)
                    print(attr_list)
                    row+=1
                if (time.time()-t1) > 5 and not self.isStop:  # 5Seconds
                    print("Updateing List")
                    self.packet.initiating_packets()
                    t1 = time.time()
                if self.pause:
                    while(self.pause):
                        pass
                if self.isStop():
```

```
                    self.stop=False
                    break
                except :
                    print("Err")
            except :
                print("Error in looooooop")

    def pause_resume(self):
        if self.pause:
            self.pause = False
            self.btn_start.setText("Pause")
        else:
            self.pause = True
            self.btn_start.setText("Resume")


    def save_log_file(self):
        log = self.tree_classifier.get_log()
        url = QFileDialog.getSaveFileName(None, 'Save Log',
'untitled', "Text file (*.txt);;All Files (*)")
        if url[0]:
            try:
                name = url[1]
```

```
            url = url[0]
            with open(url, 'w') as file:
                file.write(log)
            self.showdialog('Saved',f'File saved as {url}',1)
        except:
            self.showdialog('Error','File not saved',2)


def stop_capturing_testing(self):
    if self.pause:
        self.pause = False
        self.btn_start.setText('Pause')
    if not self.stop:
        self.stop = True
    if self.threadActive:
        self.threadActive = False
def reset_all_content(self):
    if self.pause:
        self.pause = False
        self.btn_start.setText('Pause')
    self.stop=False
    self.tree_classifier.reset_class_count()
    self.panel_capturing.clearContents()
    self.panel_capturing.setRowCount(0)
```

```
        self.panel_result.clearContents()
        self.panel_result.setRowCount(0)
        self.panel_testing.clear()


    def insert_data(self,line,result,result_type,row):
        self.panel_capturing.insertRow(row)
        for column, item  in enumerate(line[0:4:1]):
self.panel_capturing.setItem(row,column,QtWidgets.QTableWidget
Item(str(item)))
            self.panel_capturing.scrollToBottom()
        self.panel_testing.clear()
        self.panel_testing.addItem(str(line[0:4:1]))
        if not result==0:
            result_row = self.panel_result.rowCount()
            self.panel_result.insertRow(result_row)
            x = [row+1, line[1], line[2], result_type]
            for column, item  in enumerate(x):
self.panel_result.setItem(result_row,column,QtWidgets.QTableWid
getItem(str(item)))
                self.panel_result.scrollToBottom()
```

```
    def clickexit(self):
        buttonReply = QMessageBox.question(MainWindow, 'Exit', "Are ou sure to exit?", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if buttonReply == QMessageBox.Yes:
            if self.threadActive:
                self.pause = False
                self.stop = True
            qApp.quit()
        else:
            print('No clicked.')



    def isStop(self):
        return self.stop
    def showdialog(self,title,text, icon_type):
        msg = QMessageBox()
        if icon_type==1:
            msg.setIcon(QMessageBox.Information)
        elif icon_type==2:
            msg.setIcon(QMessageBox.Critical)
        elif icon_type==3:
```

```
            msg.setIcon(QMessageBox.Warning)
        msg.setText(text)
        msg.setWindowTitle(title)
        msg.setStandardButtons(QMessageBox.Ok)
        msg.buttonClicked.connect(self.msgbtn)
        retval = msg.exec_()

    def msgbtn(self):
        self.progressBar.setProperty("value", 0)
    def isModelTrained(self):
        return self.trained
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        path = os.path.dirname(os.path.abspath(__file__))

MainWindow.setWindowIcon(QtGui.QIcon(os.path.join(path,'icon.p
ng')))
        MainWindow.resize(908, 844)
        sizePolicy                                              =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
```

```
sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeight
ForWidth())

        MainWindow.setSizePolicy(sizePolicy)

        MainWindow.setIconSize(QtCore.QSize(30, 30))

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)

        self.gridLayout.setObjectName("gridLayout")

        spacerItem = QtWidgets.QSpacerItem(10, 10,
QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)

        self.gridLayout.addItem(spacerItem, 1, 0, 1, 1)

        spacerItem1 = QtWidgets.QSpacerItem(20, 20,
QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Maximum)

        self.gridLayout.addItem(spacerItem1, 4, 1, 1, 1)

        spacerItem2 = QtWidgets.QSpacerItem(20, 10,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)

        self.gridLayout.addItem(spacerItem2, 6, 1, 1, 1)

        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()

        self.horizontalLayout_2.setObjectName("horizontalLayout_2")

        spacerItem3 = QtWidgets.QSpacerItem(15, 10,
QtWidgets.QSizePolicy.Ignored, QtWidgets.QSizePolicy.Minimum)

        self.horizontalLayout_2.addItem(spacerItem3)

        self.btn_start = QtWidgets.QPushButton(self.centralwidget)
```

```
        self.btn_start.setObjectName("btn_start")
        self.btn_start.setText('Pause')
        self.btn_start.clicked.connect(self.pause_resume)
        self.horizontalLayout_2.addWidget(self.btn_start)

        #
##########################################################
        self.btn_pause = QtWidgets.QPushButton(self.centralwidget)
        self.btn_pause.setText("Stop Capturing/Testing")

        self.btn_pause.setObjectName("btn_pause")
        self.btn_pause.clicked.connect(self.stop_capturing_testing)
        self.horizontalLayout_2.addWidget(self.btn_pause)
        self.gridLayout.addLayout(self.horizontalLayout_2, 8, 1, 1, 1)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        #
##########################################################
        self.btn_modeltrain                                    =
QtWidgets.QPushButton(self.centralwidget)
        self.btn_modeltrain.setText("Train Model")

        self.btn_modeltrain.setObjectName("btn_modeltrain")
```

```
        self.btn_modeltrain.clicked.connect(self.train_model)
        self.horizontalLayout.addWidget(self.btn_modeltrain)
        #
##########################################################
#
        self.btn_statictesting                                  =
QtWidgets.QPushButton(self.centralwidget)
        self.btn_statictesting.setText("Static Testing")

        self.btn_statictesting.setObjectName("btn_statictesting")
        self.btn_statictesting.clicked.connect(self.static_testing)
        self.horizontalLayout.addWidget(self.btn_statictesting)
        #
##########################################################
#
        self.btn_realtimetesting                                =
QtWidgets.QPushButton(self.centralwidget)
        self.btn_realtimetesting.setText("")



        self.btn_realtimetesting.setObjectName("")
        self.btn_realtimetesting.clicked.connect(self.realtime_testing)
        self.horizontalLayout.addWidget(self.btn_realtimetesting)
```

```
        #
##########################################################
#
        self.btn_savelog                                    =
QtWidgets.QPushButton(self.centralwidget)
        self.btn_savelog.setText("Save Log")
        icon5 = QtGui.QIcon()

        self.btn_savelog.setObjectName("btn_savelog")
        self.btn_savelog.clicked.connect(self.save_log_file)
        self.horizontalLayout.addWidget(self.btn_savelog)

        #
##########################################################
#
        self.btn_graph = QtWidgets.QPushButton(self.centralwidget)
        self.btn_graph.setText("Plot Graph")

        self.btn_graph.setObjectName("btn_graph")
        self.btn_graph.clicked.connect(self.plot_graph)
        self.horizontalLayout.addWidget(self.btn_graph)

        #
##########################################################
#
```

```
        self.btn_exit = QtWidgets.QPushButton(self.centralwidget)
        self.btn_exit.setText("Exit")



        self.btn_exit.setObjectName("btn_exit")
        self.btn_exit.clicked.connect(self.clickexit)
        self.horizontalLayout.addWidget(self.btn_exit)
        #
##########################################################
#
        self.gridLayout.addLayout(self.horizontalLayout, 3, 1, 1, 2)
        spacerItem4 = QtWidgets.QSpacerItem(20, 10,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem4, 8, 1, 1, 1)
        spacerItem5 = QtWidgets.QSpacerItem(20, 10,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem5, 0, 1, 1, 1)
        self.panel_capturing =
QtWidgets.QTableWidget(self.centralwidget)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(10)
        sizePolicy.setVerticalStretch(0)
```

```
sizePolicy.setHeightForWidth(self.panel_capturing.sizePolicy().has
HeightForWidth())
        self.panel_capturing.setSizePolicy(sizePolicy)
        self.panel_capturing.setRowCount(0)
        self.panel_capturing.setColumnCount(4)
        self.panel_capturing.setObjectName("panel_capturing")
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.panel_capturing.setHorizontalHeaderItem(3, item)
        self.gridLayout.addWidget(self.panel_capturing, 4, 1, 4, 1)
        self.label = QtWidgets.QLabel(self.centralwidget)
        sizePolicy                                              =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.label.sizePolicy().hasHeightFor
Width())
```

```
        self.label.setSizePolicy(sizePolicy)
        self.label.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.label.setAutoFillBackground(False)
        self.label.setText("")
        path = os.path.dirname(os.path.abspath(__file__))
        path = path + r'\icons'

self.label.setPixmap(QtGui.QPixmap(os.path.join(path,'logo.jpg')))
        self.label.setScaledContents(True)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.gridLayout.addWidget(self.label, 1, 1, 1, 1)
        spacerItem6 = QtWidgets.QSpacerItem(10, 20,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Fixed)
        self.gridLayout.addItem(spacerItem6, 2, 1, 1, 1)
        self.panel_testing =
QtWidgets.QListWidget(self.centralwidget)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.panel_testing.sizePolicy().hasH
eightForWidth())
```

```
        self.panel_testing.setSizePolicy(sizePolicy)

self.panel_testing.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)

self.panel_testing.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
        self.panel_testing.setObjectName("panel_testing")
        self.gridLayout.addWidget(self.panel_testing, 9, 1, 1, 1)
        self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
        self.progressBar.setProperty("value", 0)
        self.progressBar.setObjectName("progressBar")
        self.gridLayout.addWidget(self.progressBar, 10, 1, 1, 2)
        # ------------------------------------------------------------- #

        self.panel_result = QtWidgets.QTableWidget(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(10)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.panel_result.sizePolicy().hasHeightForWidth())
```

```
self.panel_result.setSizePolicy(sizePolicy)
self.panel_result.setRowCount(0)
self.panel_result.setColumnCount(4)
self.panel_result.setObjectName("panel_result")
item = QtWidgets.QTableWidgetItem()
self.panel_result.setHorizontalHeaderItem(0, item)
item = QtWidgets.QTableWidgetItem()
self.panel_result.setHorizontalHeaderItem(1, item)
item = QtWidgets.QTableWidgetItem()
self.panel_result.setHorizontalHeaderItem(2, item)
item = QtWidgets.QTableWidgetItem()
self.panel_result.setHorizontalHeaderItem(3, item)
self.gridLayout.addWidget(self.panel_result, 4,2,6,1)
# ------------------------------------------------------------- #
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 908, 26))
self.menubar.setObjectName("menubar")
self.menuFile = QtWidgets.QMenu(self.menubar)
self.menuFile.setObjectName("menuFile")
self.menuAbout = QtWidgets.QMenu(self.menubar)
self.menuAbout.setObjectName("menuAbout")
MainWindow.setMenuBar(self.menubar)
```

```
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
self.actionNew = QtWidgets.QAction(MainWindow)
self.actionNew.setObjectName("actionNew")
self.actionOpen = QtWidgets.QAction(MainWindow)
self.actionOpen.setObjectName("actionOpen")
self.actionExit = QtWidgets.QAction(MainWindow)
self.actionExit.setObjectName("actionExit")
self.actionHelp = QtWidgets.QAction(MainWindow)
self.actionHelp.setObjectName("actionHelp")
self.menuFile.addAction(self.actionNew)
self.menuFile.addAction(self.actionOpen)
self.menuFile.addSeparator()
self.menuFile.addAction(self.actionExit)
self.actionExit.triggered.connect(qApp.quit)
self.menuAbout.addAction(self.actionHelp)
self.menubar.addAction(self.menuFile.menuAction())
self.menubar.addAction(self.menuAbout.menuAction())

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

```
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow",
"Intrusion Detection"))
        self.btn_start.setStatusTip(_translate("MainWindow",
"Pause/Resume"))
        self.btn_pause.setStatusTip(_translate("MainWindow",
"Stop"))
        self.btn_modeltrain.setStatusTip(_translate("MainWindow",
"Train Model"))
        self.btn_statictesting.setToolTip(_translate("MainWindow",
"Stactic Testing"))
        self.btn_statictesting.setStatusTip(_translate("MainWindow",
"Static Testing"))


        self.btn_savelog.setToolTip(_translate("MainWindow",    "Real
Time Capturing"))
        self.btn_savelog.setStatusTip(_translate("MainWindow",
"Real Time Capturing"))
        self.btn_graph.setStatusTip(_translate("MainWindow",
"Graph"))
        self.btn_exit.setStatusTip(_translate("MainWindow", "Exit"))
        item = self.panel_capturing.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Duration"))
        item = self.panel_capturing.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Protocol"))
```

```
        item = self.panel_capturing.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Service"))
        item = self.panel_capturing.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Src_Bytes"))
        # ------------------------------------------------- #
        item = self.panel_result.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Packet #"))
        item = self.panel_result.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Protocol"))
        item = self.panel_result.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Service"))
        item = self.panel_result.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Class"))
        # ------------------------------------------------- #
        self.menuFile.setTitle(_translate("MainWindow", "File"))
        self.menuAbout.setTitle(_translate("MainWindow", "About"))
        self.actionNew.setText(_translate("MainWindow", "New"))
        self.actionOpen.setText(_translate("MainWindow", "Open"))
        self.actionExit.setText(_translate("MainWindow", "Exit"))
        self.actionHelp.setText(_translate("MainWindow", "Help"))


if __name__ == "__main__":
```

```
import sys

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()

ui = Ui_MainWindow()

ui.setupUi(MainWindow)

MainWindow.show()

sys.exit(app.exec_())
```