DETECTION METHODS FOR SOFTWARE DEFINED NETWORK INTRUSIONS

A PROJECT REPORT

Submitted by

SIGIRALA NAVEEN [RA2011042010086]

P INDIVAR [RA2011042010107]

Under the Guidance of

Dr. K. PRIYADARSINI

Associate Professor, Department of Data Science and Business Systems

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND BUSINESS SYSTEMS

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SRM NAGAR, KATTANKULATHUR-603 203

MAY 2024

Department of Data Science and Business Systems
SRM Institute of Science & Technology
Own Work Declaration Form

Degree/ Course	:	**Bachelor of Technology, Computer Science and Business Systems**
Student Name	:	**Sigirala Naveen, P Indivar**
Registration Number	:	**RA2011042010086, RA2011042010107**
Title of Work	:	**DETECTION METHODS FOR SOFTWARE DEFINED NETWORK INTRUSIONS**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that we have received from others (e.g.fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook /University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:	
We are aware of and understand the University's policy on Academic misconduct and plagiarism and We certify that this assessment is our own work, except where indicated by referring, and that we have followed the good academic practices noted above.	
Sigirala Naveen [RA2011042010086] Sign: Date:	P Indivar [RA2011042010107] Sign: Date:

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that Project Evaluation-II [18CSP462L] report titled **"DETECTION METHODS FOR SOFTWARE DEFINED NETWORK INTRUSIONS"** is the bonafide work of **"SIGIRALA NAVEEN[RA2011042010086], P INDIVAR[RA2011042010107]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion for this or any other Candiate.

Signature

Dr. K. Priyadarsini
Supervisor
Associate professor
Department of Data Science
and Business Systems

Signature

Dr. M. Lakshmi
Professor and HOD
Department of Data Science
and Business Systems

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department**, Dr.M.Lakshmi,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Academic Advisor**, Dr.E.Sasikala,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Dr. Prasanna V,** Associate Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr.K.Priyadarsini,** Associate Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under her mentor-ship. She provided me with the freedom and support to explore the research topics of my interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Sigirala Naveen[RA2011042010086]

P Indivar[RA2011042010107]

ABSTARCT

These days, Intrusion Detection Systems (IDS) are gaining a lot of attention as an essential component of system defense. To safeguard the network, IDSs gather data on network traffic from various locations inside the computer system or network. It is quite tough and takes a lot of effort to discern between typical and intrusive network traffic operations. To determine the order of the network connection intrusion, an analyst needs to examine all the extensive and varied data. It thus requires a method to represent the current network traffic that can identify network intrusion. In this paper, a unique approach to utilizing data mining techniques and evolutionary algorithms for ML to identify intrusion characteristics for IDS was developed. Classification using a generic algorithm of decision trees is the method used to produce rules. These rules can identify the features of an intrusion and then be used as preventative measures in the GA. to prevent intrusions in addition to identifying their presence.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

ABBREVIATIONS	EXPANSION
P.D	Pandas
ML	ML
NumPy	Numerical python
DT	Decision Tree
GA	Genetic Algorithm
TN	True negative
TP	True positive
FN	False negative
FP	False Positive
PCP	Principle Component Analysis
IDS	Intrusion Detection System

CHAPTER 1

INTRODUCTION

Computer systems are increasingly vulnerable to security attacks because of the Internet's recent fast expansion. Computer system security is still extremely challenging, even with all the technological advancements in information assurance. Consequently, while the real software systems are operating, unauthorized breaches happen. The conveniences that modern technology has provided us are appreciated, but computer systems are becoming more vulnerable to internal and external security threats. A truly safe system is practically unachievable, even with several defenses in place. As a result, network intrusion detection, which monitors network traffic and detects unwanted network access, attacks from hostile computer systems, and unusual network behaviors, is becoming increasingly popular. important. An intrusion occurs when a person tries to access a computer system or do something prohibited by law. External and internal intruders can be divided into two categories. The first group includes those who attack the system using various penetration techniques and do not have permission to access that system. It describes people who are allowed access but want to do things that are not allowed. An accurate way to identify intrusions and notify authorities is to use IDS. [3]. IDS fall into two main categories: anomaly detection and abuse detection. While DS anomalies detect deviations from typical network activity and flag possible unknown threats, the abuse detection system detects hackers using Use patterns they are familiar with. Some IDSs create hybrid detection systems by combining anomaly and abuse detection. Depending on where they look for intrusion, IDS can also be divided into two groups. Many techniques have been developed to detect network vulnerabilities, often using soft computing methods. These approaches provide the flexibility to handle problems of approximation, uncertainty, imprecision, and partial truth, to achieve both cost-effectiveness and robust solutions. In the field of intrusion detection, these techniques are often integrated with rule-based expert systems to represent knowledge using if-then rules. Intrusion detection can be classified into two main categories: abuse detection and anomaly detection. Abuse detection can recognize established attack patterns by reviewing system audit logs, while anomaly detection builds a baseline of normal behavior and triggers alerts. reporting deviations, helpful in detecting new or unexpected threats. Abuse detection is known for its high detection rate, while anomaly detection is prized for its ability to identify new attacks, despite its potential to generate many false positives than. Machine learning techniques applied to networkmonitoring have many different applications in many different fields. For example, a method using bidirectional long-term and short-term memory neural networks isintroduced to analyze and detect road accidents in social media networks. The proposed method uses query-based web mining to collect phrases related to traffic- related incidents, such as traffic jams and road closures, from media platforms social networks like Facebook and Twitter. Once collected, data goes through several pre- processing steps, including segmentation, tokenization, root tagging, and POS tagging to ensure efficient organization. Latent Dirichlet allocation (LDA) technique is then used to automatically classify the data into “traffic” or “non-traffic” categories. The labeled traffic data is then divided into positive, negative or neutral classifications, resulting in statements classified as traffic-related or non-traffic-related, along with sentiment corresponding. In the next stage, a neural network is deployed to determine language, location, traffic incident type, and polarity, using a multi-layer classification approach powered by the Soft Max layer. The proposed strategy evaluates various traditional deep learning and machine learning techniques based on performance metrics such as F-score and accuracy. Intrusion detection systems (IDS) are security components designed to detect a wide range of intrusions, including hacker attacks, abuse, and unauthorized access to systems or the Internet. These systems monitor network and computer traffic and behavior, using trained models to identify and establish patterns, which are then stored in a database.

When potential threats are detected, the IDS generates alerts to notify administrators. Various machine learning methods have been used to build intrusion detection systems (IDS), including neural networks, linear genetic programming (LGP), support vector machines (SVM), decision trees and Bayesian networks such as multivariate adaptive regression lines (Mars) and fuzzy inference system (FIS).

There are two main types of IDS: network-based IDS (NIDS) and host-based IDS (HIDS).

In NIDS, the system monitors and controls all incoming and outgoing traffic at a network element by placing collection devices (sensors) such as sniffers at key points. IDS can be classified into two main types: anomaly-based and abuse detection-based. Abuse detection systems, the most used type, identify intruders based on recognized attack patterns. These patterns originate from various aspects of the network packet, including source and destination addresses, ports, or specific keywords from thepacket content. However, their limitation is that they can only detect attacks that are already in their database. This approach has a low false positive rate but requires frequent updates.

1.1: Organization of the Report:

Chapter 1 gives a brief introduction to the concept of Information Security which is a broad field that covers many areas such as physical security, endpoint security, network security, etc. This chapter also tells about various attacks and security breaches that happen either on the internet or physically at various physical systems. It highlights the importance of Information security which is very essential to prevent these attacks. To do this, Intrusion Detection System is employed as a security measure for this project.

Chapter 2 describes the system analysis of the Intrusion detection system by exploring the existing systems, problem statement and the new proposed system for the project. It also describes the advantages of the Intrusion Detection System (IDS).

Chapter 3 provides the information about the Literature Review that is done while researching about the project. In this chapter, the findings about various algorithms and methods used for the implementation of IDS are presented. Various research papers were studies and analyzed in this chapter to get insights about Intrusion Detection System.

Chapter 4 describes about the system design of the Intrusion Detection System where we explore the methodology used for the project. Here the steps involved in the system design are Dataset Collection, Preprocessing, Model Training, Model Testing and prediction. We use NSL-KDD dataset for the project and the algorithm used for model training is Genetic Algorithm (GA) and Decision Trees (DT).

Chapter 5 presents the various use case diagrams used for the Intrusion Detection System. Here we have the diagrammatic representation of the system design which explains the process of the project. Block Diagrams explains about how the data is divided into training and testing phases. In Sequence diagram, we see how the network traffic flows on the internet and how to differentiate normal and malicious data traffic. Data flow diagram describes about how the data goes from the pre-processing stage to the final prediction stage.

Chapter 6 gives an overview about the requirement specification for the project. It gives details about Project Overview, Stakeholder requirements, Functional requirements, Performance requirements, Risk Assessment, Project Timeline, etc.

Chapter 8 presents us with the findings after the implementation of the project. Each Screenshot of the datasets, trained model, graph plots for both normal and attack datasets and visualization of classes of both DOS and probe attacks are presented in this chapter.

Chapter 9 gives us the final conclusion of the project along with some suggestions about enhancements that can made to the project in the future.

CHAPTER 2

SYSTEM ANALYSIS

2.1 : Existing System:

- In today's context, networks have become an integral part of public infrastructure due to the emergence of both public and private cloud computing.
- The conventional approach to networking has grown excessively complex.
- This intricacy has posed obstacles to the development of innovative services within individual data centers, the interconnection of data centers, internal enterprise networking, and the overall expansion of the Internet.

2.2 : Problem Statement:

- Differentiating between normal network traffic and potential intrusions is a challenging and time-consuming task.
- Analysts are required to thoroughly analyze extensive data to identify patterns of intrusion within network connections.
- There's a need for a more efficient method to promptly detect network intrusions within real-time network traffic.
- Intrusion Prevention Systems (IPS), encompassing both Intrusion Detection Systems (IDS) and firewalls, are proposed to address these concerns.

2.3: Proposed System:

- GA is recommended as an important machine learning technique for intrusion detection.
- DT represents a single attribute test for given cases, while leaf nodes indicate whether the classifier output falls into the “normal” category or one of the invasive categories input differently during the classification process or not.
- A new approach is proposed to identify intrusion patterns using DT combined with GA to generate classification rules

2.4: Advantages:

- Intrusion detection can be performed manually or automatically.
- With the increasing number of alerts and events, intrusion detection systems (IDS) must be equipped to handle large and expanding data streams.
- The use of decision trees is considered necessary to automate the intrusion detection and response process, which is becoming increasingly important.

CHAPTER 3

LITERATURE REVIEW

Kumar, Das, and Sinha (2021): They presented UIDS, an IDSs tailored for IoT networks, addressing unique challenges and emphasizing adaptability and enhanced security[1].

Neha Gupta et.al : They developed LIO-IDS, an innovative approach for addressing class imbalance in IDS, showing exceptional accuracy and outperforming other existing IDSs in attack detection rates and computational efficiency[2].

Nishtha Kesswani: Their literature survey highlights the vital role of IDS in bolstering network security, providing insights into IDSs and their significance in protecting networks[3].

Hamizan Suhaimi: Introduces a cutting-edge strategy for identifying malicious connections in computer networks, with a focus on developing classification rules from thorough analysis of connection data attributes[4].

Sydney Mambwe Kasongo: Developed a cutting-edge IDS specifically for Industrial Internet of Things (IIoT) networks, utilizing feature selection algorithms and tree-based classifiers to strengthen security and privacy[5].

Syurahbil's work: Their literature survey on IDSs highlights the utilization of DT Data Mining technique to differentiate normal network traffic from intrusions. The work provides practical firewall rules derived from DT and demonstrates the system's effectiveness in identifying various types of intrusions, including DoS[6].

Feng Gao's work: Their literature survey focuses on semi-supervised anomaly detection, emphasizing the cost-effective utilization of limited positive samples during model training. This approach offers a more economical solution for reducing the need for extensive labeled data, while still ensuring accurate anomaly detection[7].

Yuanyi Chen's revolutionary work in road anomaly detection uses scale-invariant features, diverse data sources, and advanced ML algorithms. Their model excels in accuracy and efficiency, surpassing existing methods[8].

Yunseung Lee and Pilsung Kang:In their groundbreaking paper AnoViT" presents an unsupervised approach using vision transformers for accurate and precise image anomaly detection[9].

Hongju Cao's contribution in the literature survey includes exploring OSP for hyperspectral anomaly detection, incorporating data sphering, and demonstrating the superiority of the OSP-based method[10].

Seunghyun's work focuses on using advanced SDN technology and the innovative concept of Moving Target Defense (MTD) to increase uncertainty for potential attackers, effectively countering cyberattacks[11].

Haocheng Shen's research introduces a groundbreaking anomaly detection method that eliminates the need for abnormal data by training a Generative AdversarialNetwork (GAN) with synthetic normal data, surpassing traditional techniques in accuracy and resilience[12].

Mehdi Shajar's work introduces a revolutionary online network anomaly detection and diagnosis system using tensor-based techniques, offering efficient detection and automated response capabilities for various network types[13].

Hwan Kim's survey focuses on the significance of Graph Neural Networks (GNNs) in detecting anomalies in graphs, providing a comprehensive examination of different types of graphs and highlighting the challenges of incorporating GNNs[14].

Adam Lundström's paper explores the complexities of detecting anomalies in multivariate time series using deep learning, proposing an effective approach with separated anomaly scoring and emphasizing the practical significance in real-world scenarios[15].

Ujjan et al. (2020) explored using deep learning techniques in SDN to detect DDoS attacks efficiently. They introduced sFlow and adaptive polling sampling for data collection, contributing to enhancing security measures in SDN through the utilization of machine learning and adaptive sampling techniques[16].

This reference is a foundational source that defines the concept of Software-Defined Networking (SDN) as per the Open Networking Foundation's perspective. It provides a formal definition and an overview of SDN, which is essential for understanding the basic principles and ideas behind SDN technology. You can use this reference to establish the fundamental understanding of SDN in your literature review[17].

Garg et al. (2019) proposed a hybrid deep learning-based scheme to detect suspicious network flows in SDN, focusing on social multimedia data[18].

Nobakht et.al. introduced a framework for securing Smart Home IoT systems using Open Flow technology[19].

Elsayed et.al. delved into machine learning techniques for network attack detection in SDN. These works offer valuable insights into securing SDN and IoT environments through adaptive techniques and ML[20].

Choudhary and Kesswani: Explored deep learning for IoT security using datasets, contributing to enhancing anomaly detection in IoT environments[21].

Dahiya and Srivastava: Investigated the use of Apache Spark for network intrusion detection, addressing scalability and performance challenges[22].

Dlamini and Fahim : Introduced a data generative model to improve anomaly detection in network security, particularly for rare threats[23].

Faker and Dogdu : Explored combining big data and deep learning for intrusion detection, providing insights into network security techniques[24].

Gupta et al. : Explored various data mining techniques for network intrusion detection, providing insights into the evolution of IDSs[25].

Ibrahim Aliyu et al.: Developed a security framework for vehicle networks using blockchain technology and statistical detection techniques, focusing on "adversarial examples" to mitigate attacks in the automotive ecosystem[26].

Muhammad Waqas Nadeem et al.: Investigated the application of deep learning for detecting and mitigating botnet attacks in SDN, offering valuable insights into countering sophisticated and evolving threats[27].

Gun-Yoon Shin et.al.: Explored data discretization and analysis techniques to detect unfamiliar network attacks, contributing to enhancing intrusion detection byuncovering previously unrecognized threats[28].

CHAPTER 4

SYSTEM DESIGN

4.1 : METHODOLOGY

4.1.1 : DATASET COLLECTION

NSL-KDD Data Set

- A denial-of-service (DoS) attack is intended to disrupt the target system's ability to transmit and receive network traffic. In contrast, a surveillance attack, often referred to as a probe, is focused on collecting data within a network. In this case, the objective is to impersonate unauthorized access and extract sensitive information, such as financial or personal client data.
- The dataset is comprised of four distinct feature categories:

  4 Categorical features (e.g., Features 2, 3, 4, 42), 6 Binary features (e.g., Features 7, 12, 14, 20, 21, 22), 23 Discrete features (e.g., Features 8, 9, 15, 23–41, 43), and 10 Continuous features (e.g., Features 1, 5, 6, 10, 11, 13, 16, 17, 18, 19).

4.1.2 : PREPROCESSING:

- In the preprocessing phase, continuous attribute values are transformed to a scale between 0 and 1. This normalization is carried out to prevent any single attribute from dominating the analysis.
- To perform this scaling using the mean and standard deviation, you can utilize the "transform(data)" method.

4.1.3 : MODEL TRAINING:

- The Decision Tree technique uses internal nodes to test attributes, branches to represent the test results, and leaf nodes to store class labels.
- The process of selecting attributes in decision tree algorithms involves identifying the characteristics that minimize uncertainty or impurity in partitions. These chosen attributes make the tuple categorization process more efficient by reducing the required information and anticipated number of tests. The ID3 algorithm uses entropy to determine which attributes to search at each node in the decision tree.

Genetic Algorithms (GAs) have demonstrated significant potential in the realm of computer security, particularly when applied to Intrusion Detection Systems (IDS). Detecting network intrusions in real-time requires efficient utilization of critical data. Principal Component Analysis (PCA), also known as the Karhunen-Loève transform, is utilized to pinpoint the most essential data features. The primary goal of PCA is to reduce data conditionality by identifying a limited number of orthogonal linear combinations of variables with the highest variance.

Despite the development of efficient machine learning techniques in the field of intrusion detection, there is an ongoing need to enhance detection rates while reducing the occurrence of false alarms. GAs stand out due to their distinctive approach, outperforming conventional methods in terms of reducing false alarms and improving detection rates for identifying network intrusions based on anomalies.

GAs are search algorithms inspired by genetic and natural selection principles. They initiate with an initial population of individuals and evolve it into a population of high-quality individuals, each representing a potential solution to the given problem. Each individual possesses a set of genes, often referred to as chromosomes. A fitness function quantitatively assesses how well each individual adapts to its environment, thereby determining its quality.

The process begins with the random selection of an initial population. Over several generations, the population evolves, with each individual's traits steadily improving, as indicated by an increase in the fitness value, which serves as a proxy for quality. Three fundamental genetic operators—selection, crossover, and mutation—aresuccessively applied to individuals with specific probabilities during each generation.

By employing biologically inspired operators like mutation, crossover, and selection, GAs are frequently used to generate excellent solutions to optimization and search problems. Like other search algorithms, GAs are used in artificial intelligence to explore a space of potential solutions to identify the one that resolves the problem. GAs excel at navigating broad, potentially vast search spaces in search of the best possible combinations of elements—solutions that might be challenging to discover through traditional methods.

The three primary genetic operations of a GA are crossover, mutation, and fittest selection. The basic process for a GA consists of the following stages:

1. Initialization: Create an initial population.

2. Evaluation: Each member of the population is then assessed, and a 'fitness' value is calculated for that individual.

3. Selection: The goal is to continuously improve the population's overall fitness.

To assess the effectiveness of this approach, the KDD Cup '99 dataset was employed. PCA was used to identify specific network attributes that are more likely to be associated with network breaches. The training dataset underwent PCA to identify the most relevant features for detecting network attacks. This process led to the selection of three features from the 41 available in the KDD Cup '99 dataset to characterize each network connection. The primary objective was to maintain a high intrusion detection rate while using the fewest features necessary, enabling real-time detection. The chosen features and their explanations are presented in down, where each characteristic represents a single gene within the chromosome.

The chosen network features consist of three components: Duration (representing the length in seconds of the connection), Src_bytes (indicating the number of data bytes from source to destination), and Dst_host_srv_serror_rate (reflecting the percentage of connections with "SYN" errors).

Each of these features corresponds to a gene in a chromosome that represents an intrusion detection rule, essentially forming an if-then statement.

In the creation of intrusion detection rules for penetration testing, the selected features from the above data are combined using the AND function in the conditional part of the rule.

For instance, a rule might read: "If an intrusion is detected, then (duration = '1', src_bytes = '0', and dst_host_srv_serror_rate = '0').

" The fitness function is employed to compute the fitness value of each rule, taking into account various factors.These include the total number of attacks (A) and normal connections (B) in the training dataset.Adjustment values range from -1 to 1, where -1 is the lowest value and 1 is the highest value. Higher fitness values are achieved with a low false positive rate and high detection rate. Conversely, high fitness values result from low detection rates and high false positive rates.

4.1.4 : MODEL TESTING

EVALUATION METRICS

• To compare how well ML algorithms perform, a confusion matrix is employed. The values of TN, TP, FN, and FP are combined to create various metrics using this matrix. Here are a few performance metrics that are used to assess models using the confusion matrix.

• Accuracy indicates the percentage of all samples that are correctly classified, or how closely the predicted value matches the original value of the model.

Precision identifies the proportion of relevant instances among the chosen instances that are positive.

•The percentage of correctly identified true positives is calculated using recall, also known as true positive rate (TPR).

4.1.5 : PREDICTION

- The goal was to maintain a high IDS rate while choosing the fewest features feasible. Consequently, real-time detection could be carried out.
- Enter normal data into the GUI's trained model, plot the graph for intrusion detection (which displays a bar graph), and save the log to see the result.

- Enter attack data into the GUI's trained model, plot the graph for intrusion detection (which shows a bar graph of dos and probe attacks), and save the log to see the results

CHAPTER 5
USE CASE DIAGRAMS

5.1 : SYSTEM DESIGN

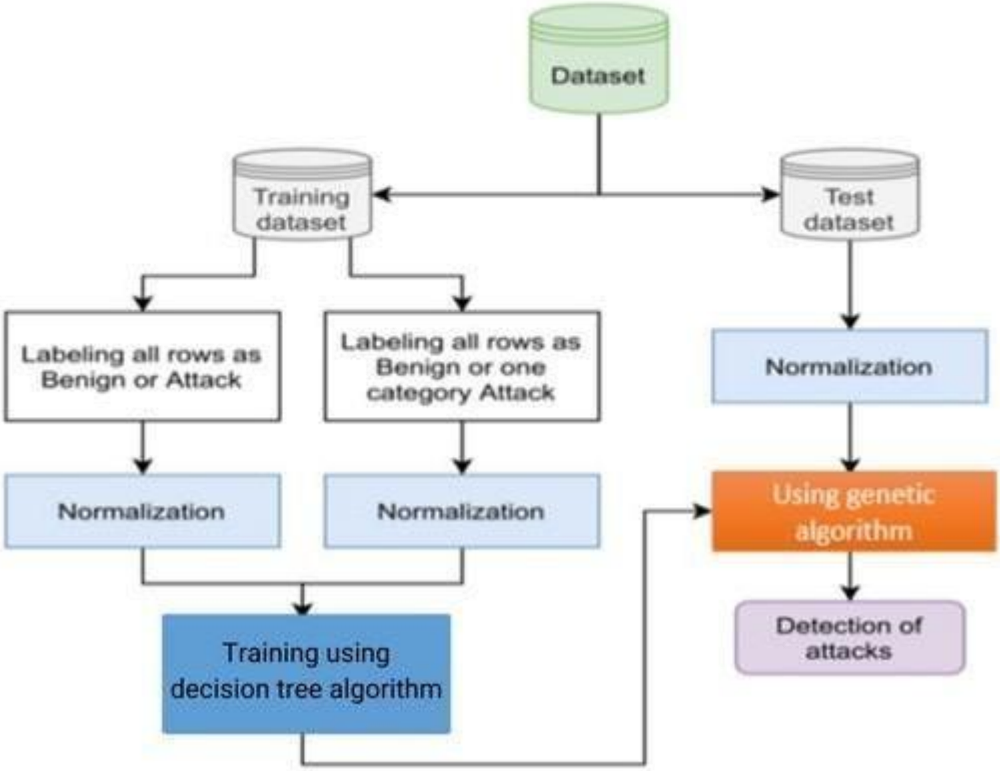


Fig.5.1: SYSTEM DESIGN

Fig.5.1 shows the system design of the application. Of course, include the terms "training," "testing," "normalization," "GA," and "decision tree" in a detailed description of the project architecture. In our project, we want to create a robust system to protect computers. Protect network access from unauthorized persons and malicious activity.It all starts with the collection of data from various sources like system logs, network packets, and other network-related data. This data collection phase is akin to gathering information about what's happening in the computer systemor network.Once we have this data, we don't immediately jump into analysis. First, weneed to 'train' our system to understand what normal network behavior looks like using ML techniques, especially 'decision trees.' Decision trees are like road maps that guide us through the data, helping us create rules to spot unusual behavior.But before we can put these rules to work effectively, we need to clean up and 'normalize' the data, much like tidying up a messy room. We fix inconsistencies, handle missing data,and ensure that everything is in a consistent and understandable format.With our 'trained' system and 'normalized' data, we extract relevant features from the dataset. Think of this as identifying the key details that help us distinguish between normaland abnormal network behavior during 'testing.'

These 'decision trees' play a vital role in understanding and classifying network behavior. And as an added layer of intelligence, we incorporate a 'GA.' This algorithm is a bit like evolution in nature. It continuously 'evolves' and improves the rules based on what it learns from the data. It's as if our system is constantly 'learning' and 'adapting' to new challenges in the network.Of course, we want to make sure that the system is doing its job properly, so we evaluate its performance using various measures, including 'testing' to see how well it's 'learned' to spot intrusions.Once our system is 'trained,' 'normalized,' and ready, we deploy it. It's like having a vigilant security guard on duty, watching the network all the time. If the system 'spots'something suspicious during its 'testing' phase, it raises the alarm.Throughout the project, we keep thorough documentation. It's like taking notes in class so that we can look back and remember what we did, which is crucial for 'training' and improving thesystem in the future.We also create reports to summarize how well the system is working. These reports help us understand if the system is effectively using 'decision trees' and 'GAs' to protect the network.

Finally, we suggest ways to make the system even better in the future. It's like saying, "Here are some ideas to make the security system even smarter and more effective."

This entire approach helps us build an intelligent security system that 'learns' from the data it 'tests,' adapts using 'GAs,' and keeps the computer system or network safe from intruders.

5.2 : BLOCK DIAGRAM:

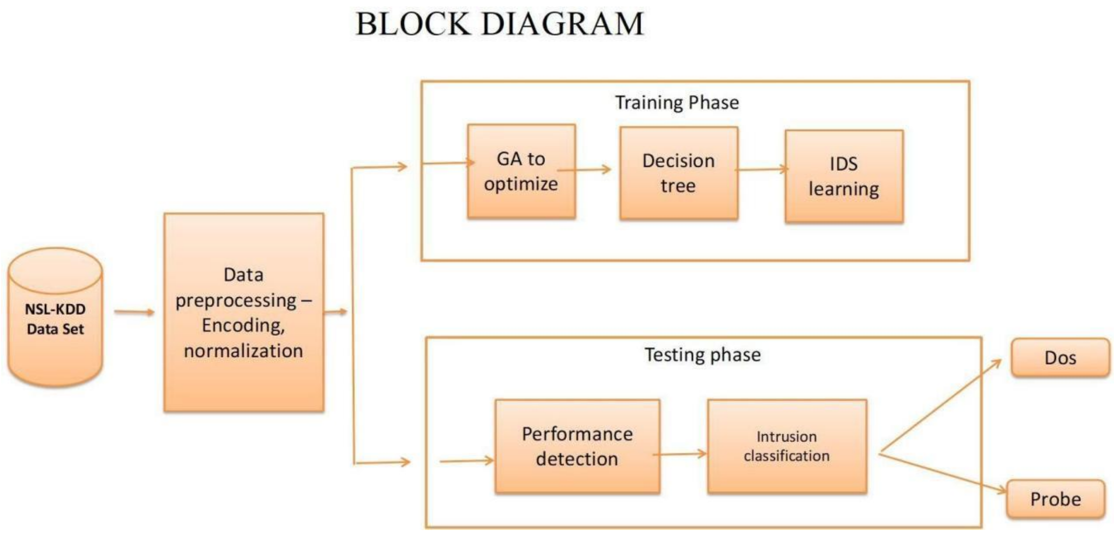


Fig.5.2: BLOCK DIAGRAM

5.3 : SEQUENCE DIAGRAM:

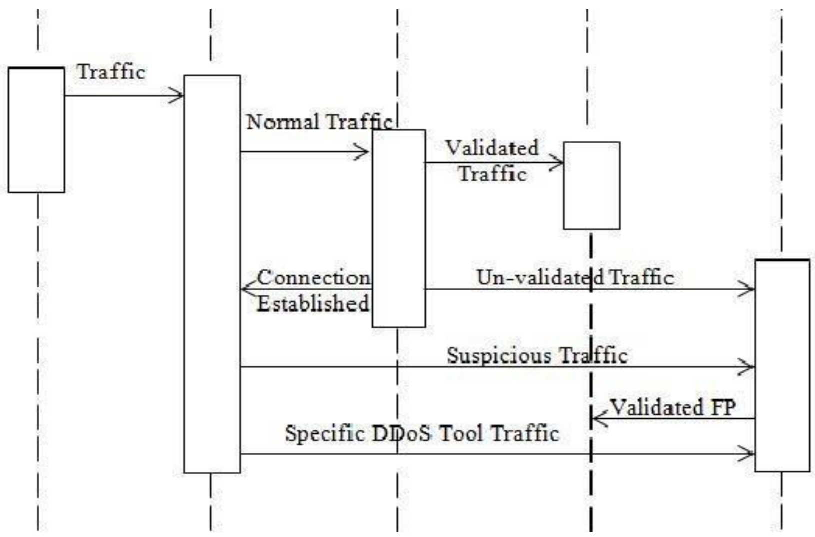


Fig.5.3: SEQUENCE DIAGRAM

5.4 :DATA FLOW DIAGRAM

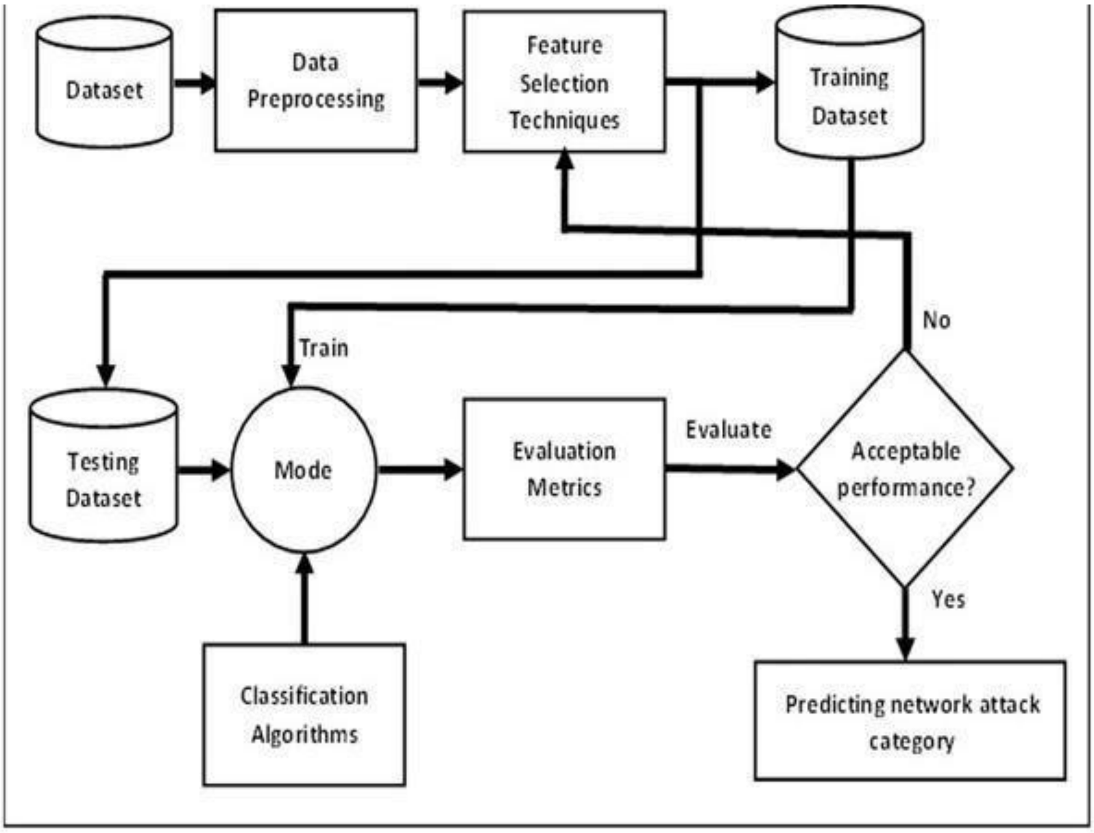


Fig.5.4:DATA FLOW DIAGRAM

Fig.5.4 shows the data flow diagram of the application. In our quest to build a robust IDS, we've incorporated several crucial features that elevate our project's effectiveness and security. First and foremost, our system embraces "Real-Time Monitoring" as a core principle. It's not a one-and-done setup; it stands guard like a vigilant sentry, continually scrutinizing the network in real time. This constant watchfulness ensures that any signs of unauthorized access or malicious activities are swiftly identified. But identification alone is not enough; we've established an "Alerting Mechanism" that complements our real-time monitoring. When our system spots something unusual, it doesn't keep it under wraps. Instead, it promptly raises an alert, notifying administrators and relevant authorities. This immediate response is crucial in facilitating swift action against potential threats, effectively mitigating risks.What sets our system apart is its "Adaptive Learning" capability. It's not static; it's dynamic and learns from every event. This 'adaptive learning' ensures that the system remains effective in the ever-evolving landscape of cybersecurity threats. It continually adapts its rules and strategies, ensuring that it's always one step ahead of potential intruders.

CHAPTER 6

REQUIREMENT SPECIFICATION

- Project Overview: The aim and objective of the project is to develop an IDS to improve the accuracy and effectiveness of network intrusion detection.
- Stakeholder requirements: Identify stakeholders, including network administrators, security professionals, and end users, and document their specific needs and expectations.
- Functional Requirements: IDS Functionality Specification: Implements the use of Genetic Algorithms (GA) for evolutionary optimization.
- Using decision trees (DT) to classify intrusions.
- Monitor and analyze network traffic to detect intrusions.
- Determine if detected events are normal or attacks using DTs.
- Crosscheck attacks using the NLS-KDD dataset to classify them as DOS or Prob attacks.
- Count the number of attacks and their types.
- Performance Requirements: Define performance metrics, such as Detection accuracy.
- Efficiency in processing network traffic.
- Response time for classifying and detecting intrusions.
- Scalability to handle varying network loads.
- Security requirements: Ensure the security of the IDS itself to avoid tampering.
- Protect the confidentiality and integrity of intrusion detection data.
- Compatibility and integration: Ensure compatibility with network architectures and SDN environments.

- Integrates with network devices and data sources for real-time monitoring.
- Usability and User Interface: Define user interface requirements for administrators and analysts.
- Ensure the system provides meaningful alerts and reports.
- Regulatory and Compliance Requirements: Identify and comply with relevant data protection and privacy regulations.
- Document data retention policies and legal requirements.
- Maintenance and Support: Specify requirements for system maintenance, updates, and patches.
- Define support and training needs for administrators and users.
- Testing and Validation: Outline testing methodologies, including the validation of GA and DT algorithms.
- Testing with real-world scenarios and datasets.
- Define success criteria for the project.
- Documentation: Describe documentation requirements, including user manuals, system architecture, and operational procedures.
- Project timeline and milestones: Create a project timeline with specific milestones and deadlines. Budget and Resource Requirements: Estimate financial and resource requirements, including hardware, software, and personnel.
- Risk Assessment: Identify potential risks and mitigation strategies for the project.
- Change management: Determine how to manage and approve changes or updates to project requirements.

- Acceptance criteria: Determine the conditions for project success, including the accuracy, efficiency, and security of the IDS as well as its ability to detect attack patterns and root causes.

CHAPTER 7

CODE IMPLEMENTATION

```
import numpy as np

import pandas as pd

from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import Normalizer

from sklearn.metrics import (precision_score, recall_score,f1_score, accuracy_score)

import warnings

warnings.filterwarnings("ignore")

traindata = pd.read_csv('Dataset/tr.csv', header=None)

testdata = pd.read_csv('Dataset/ts.csv', header=None)


X = traindata.iloc[:,1:42]

Y = traindata.iloc[:,0]

C = testdata.iloc[:,0]

T = testdata.iloc[:,1:42]


scaler = Normalizer().fit(X)

trainX = scaler.transform(X)

scaler = Normalizer().fit(T)

testT = scaler.transform(T)

traindata = np.array(trainX)

trainlabel = np.array(Y)

testdata = np.array(testT)

testlabel = np.array(C)
```

```
print("************************************************************")
model = DecisionTreeClassifier()
model.fit(traindata, trainlabel)
print(model)
# make predictions
expected = testlabel
predicted = model.predict(testdata)
np.savetxt('predictedDT.txt', predicted, fmt='%01d')
# summarize the fit of the model
accuracy = accuracy_score(expected, predicted)
recall = recall_score(expected, predicted, average="binary")
precision = precision_score(expected, predicted , average="binary")
f1 = f1_score(expected, predicted , average="binary")

cm = metrics.confusion_matrix(expected, predicted)
print(cm)
tpr = float(cm[0][0])/np.sum(cm[0])
fpr = float(cm[1][1])/np.sum(cm[1])
print("%.3f" %tpr)
print("%.3f" %fpr)
print("Accuracy")
print("%.3f" %accuracy)
print("precision")
print("%.3f" %precision)
print("recall")
print("%.3f" %recall)
```

```
print("f-score")
print("%.3f" %f1)


import Individual
import random



class Population():
    individual = list()
    max_fitness = 0
    max_fittest = list()
    childrens = list()
    def __init__(self,train_dataset,test_dataset,population_size=5, gene_length=18):
        self.population_size = population_size
        self.gene_length=gene_length
        self.train_dataset = train_dataset
        self.test_dataset = test_dataset
        self.no_of_child = self.population_size if (self.population_size%2==0) else
self.population_size-1


    def initialize_population(self):
        for x in range(self.population_size):

self.individual.append(Individual.Individual(self.train_dataset,self.test_dataset))


    def calculate_fitness(self):
        for x in range(len(self.individual)):
            self.individual[x].calculate_fitness()
```

```
        self.individual = sorted(self.individual, key=self._get_fitness, reverse=True) #descending sorting
        self.individual = self.individual[0:self.population_size:1] #cut the extra individual with less fitness
        self.max_fittest = self.individual[0].chromosome
        self.max_fitness = self.individual[0].fitness


    def__get_fitness(self,ind):
        return ind.fitness




    def cross_over(self,parents):
        cut_point = 9 # Or can be generated randomly between range (0 - sizeOfGene)
        c1 = 0
        c2 = 1
        for x in range(self.no_of_child):
            self.childrens.append(Individual.Individual(self.train_dataset,self.test_dataset))
        for parent in parents:
            p1 = self.individual[parent[0]]
            p2 = self.individual[parent[1]]
            self.childrens[c1].chromosome = p2.chromosome[cut_point::] + p1.chromosome[cut_point::]
            self.childrens[c2].chromosome = p1.chromosome[cut_point::] + p2.chromosome[cut_point::]
            c1 = c1+2
            c2 = c2+2


    def mutation(self,mutation_rate):
```

```
        mutation_rate = float(mutation_rate/100)
        for child in self.childrens:
            gene = child.chromosome
            gene = [self._flip_bit(gene[x]) if (random.uniform(0,1)<mutation_rate) else gene[x] for x in range(len(gene))]
            child.chromosome = gene
        self.individual.extend(self.childrens)
        self.childrens.clear() # Clear childrens

    def __flip_bit(self, bit):
        '''process'''
        return 0  if bit==1 else 1

    def clear_population(self):
        self.individual.clear()

from sklearn import tree
import pandas
import string
class DecisionTree():

    def __init_(self):
        self.tree_classifier = tree.DecisionTreeClassifier()
        self._dos = 0
        self._prob = 0
         self._normal = 0
```

```
def train_classifier(self,dataset, attributes):
    header = list(string.ascii_lowercase[0:19])
    kdd_train = pandas.read_csv(dataset, names=header)
    self.selected_attributes = [x for x,y in enumerate(attributes) if y==1]
    self.selected_index= [header[x] for x, y in enumerate(attributes) if y==1]
    var_train, res_train = kdd_train[self.selected_index], kdd_train[header[18]]
    self.tree_classifier.fit(var_train, res_train)
def test_dataset(self,packet):
    packet_list = list()
    packet_list.append([packet[x] for x in self.selected_attributes])
    result  = self.tree_classifier.predict(packet_list)
    result = int(result[0])

    packet_list.clear()
    return self._classification(result)
def__classification(self, status):
    if status == 0:
        self._normal+=1
        result = 0
    elif status in range(1,6):
        self._dos+=1
        result = 1
    else:
        self._prob+=1
        result = 2
    classification = {
```

```
            '0' : 'Normal',
            '1' : 'Dos/Neptune',
            '2' : 'Dos/Back',
            '3' : 'Dos/Apache2',
            '4' : 'Dos/Phf',
            '5' : 'Dos/Saint',
            '6' : 'Prob/IpSweep',
            '7' : 'Prob/PortSweep',
            '8' : 'Prob/Satan',
            '9' : 'Prob/Nmap'
        }
        result_class = classification[str(status)]
        return (result, result_class)
    def reset_class_count(self):
        '''Reset the no.of dos,prob and normal count to zero'''
        self._dos = 0
        self._prob = 0
        self._normal = 0
    def get_class_count(self):
        return (self._normal, self._dos,self._prob)
    def get_log(self):
        total = self.__dos+self.__prob+self.__normal
        log = f'Total = {total}\nNormal = {self.__normal}\nDoS = {self.__dos}\nProb = 
{self._prob}'
        return log
    @staticmethod
    def get_fitness(var_train, res_train, var_test, res_test):
```

```
        #Consume ram<100MB, processor<15%
        clf = tree.DecisionTreeClassifier()
        clf.fit(var_train, res_train)
        return round(clf.score(var_test, res_test),3)

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)


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.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.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


import os


class Dataset():
    @staticmethod
    def refine_dataset(file_path, file_name):
        type_of_services = ['http', 'http_443', 'domain_u']
        directory = os.path.dirname(file_path)
        new_file_path = Dataset.get_new_file_path(directory,file_name)
```

```
        with open(file_path, "r") as file:
            with open(new_file_path, "w") as f:
                for x,line in enumerate(file.readlines()):
                    l = line.split(",")
                    if l[2] in type_of_services:
                        f.write(Dataset.get_attributes(l)+"\n")
        return new_file_path

    @staticmethod
    def get_new_file_path(directory, file_name): # Return path of new file
        os.chdir(directory)
        if os.path.exists(file_name):
            os.remove(file_name)
        return os.path.join(os.getcwd(),file_name)

    @staticmethod
    def get_attributes(attribute_list):
        index_list = [0,1,2,4,5,6,7,22,23,28,29,30,31,32,33,34,35,36,41]   #41 is attack
type
        index = [1,2,41]
        extrated_attributes = []
        for x in index_list:
            if x in index:
                extrated_attributes.append(Dataset.get_mapping(x,attribute_list[x]))
            else:
                extrated_attributes.append(attribute_list[x])
        line = ','.join(extrated_attributes)
```

```
    return line

@staticmethod
def get_mapping(index, value):
    protocol = {
        'tcp' : '6',
        'udp' : '17'
    }
    service =
        { 'http' : '80',
        'http_443' : '443',
        'domain_u' : '53'
    }
    attack =
        { 'normal' :
        '0',
        'neptune' : '1',
        'back' : '2',
        'apache2' : '3',
        'phf' : '4',
        'saint' : '5',
        'portsweep' : '6',
        'ipsweep' : '7',
        'nmap' : '8',
        'satan' : '9'
    }
    if(index==1):
        return protocol[str(value)]
```

```
        elif(index==2):
            return service[str(value)]
        elif(index==41):
            return attack[str(value)]
    # Adding dialog box
import class_
import random



class tree():

    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 = class_.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()
```

CHAPTER 8

OUTPUT

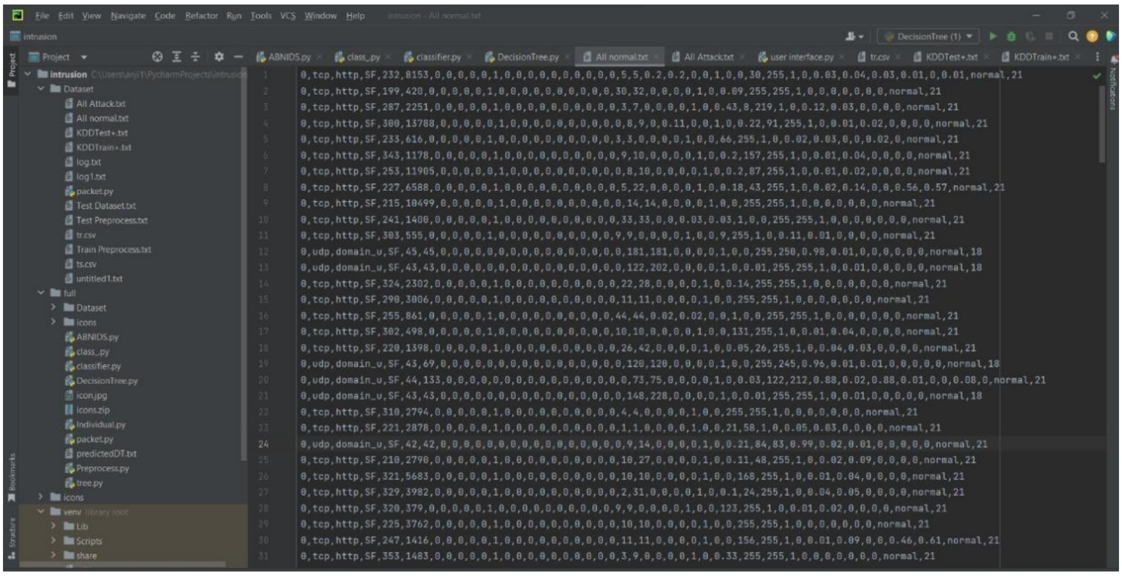


Fig 8.1: All normal dataset collection

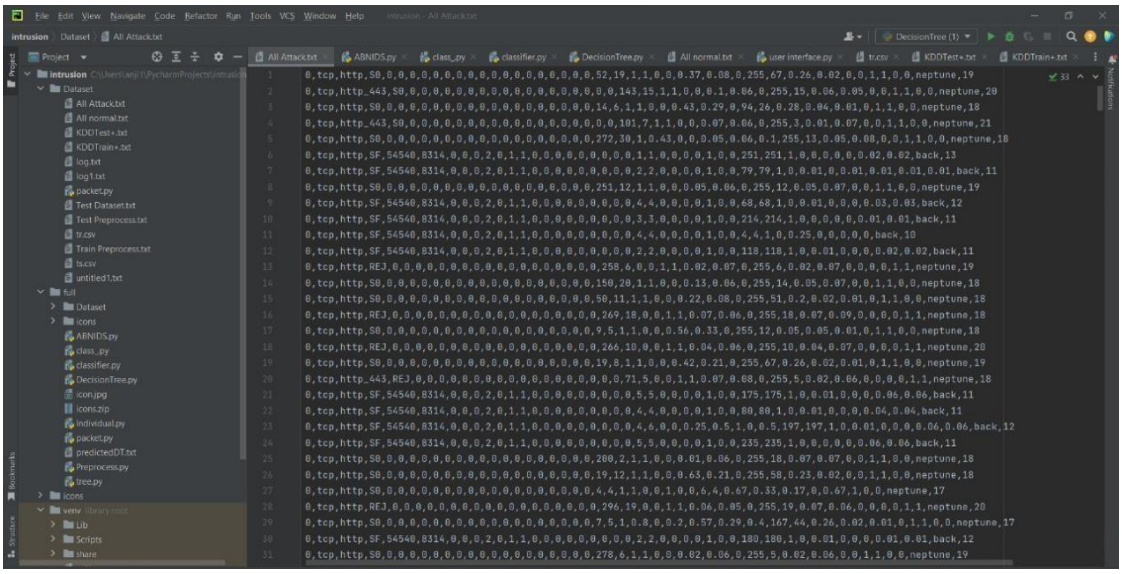


Fig 8.2: All attacks dataset collection

Fig 8.1 and 8.2 shows us about the normal and attack dataset collection which consists of data about the packet and whether the packet is normal or has some type of attack in it. It also consists of information about the type of protocol the packets use and some other features attached to it.

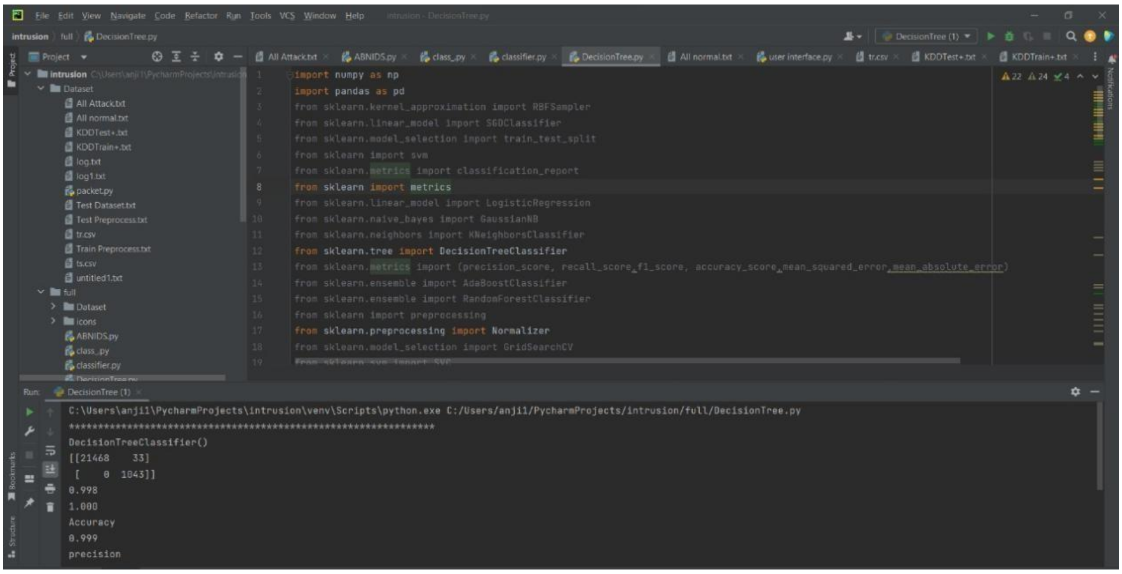


Fig 8.3: Decision tree

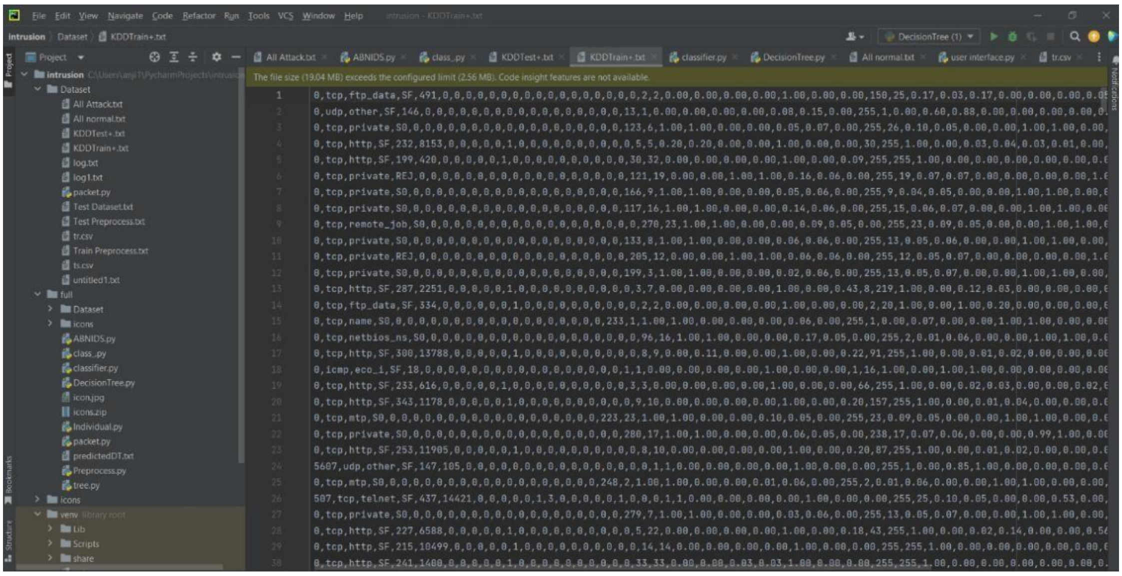


Fig 8.4: KDD Dataset Train data

Fig 8.3 gives us the information about the decision tree and the evaluation metrics like True positive, True negative, False positive and False negative. It also gives us the accuracy and precision of the model.

Fig 8.4 shows us the image of the train data from the KDD data set which consists of information about each packet. This dataset will be helpful for model training.

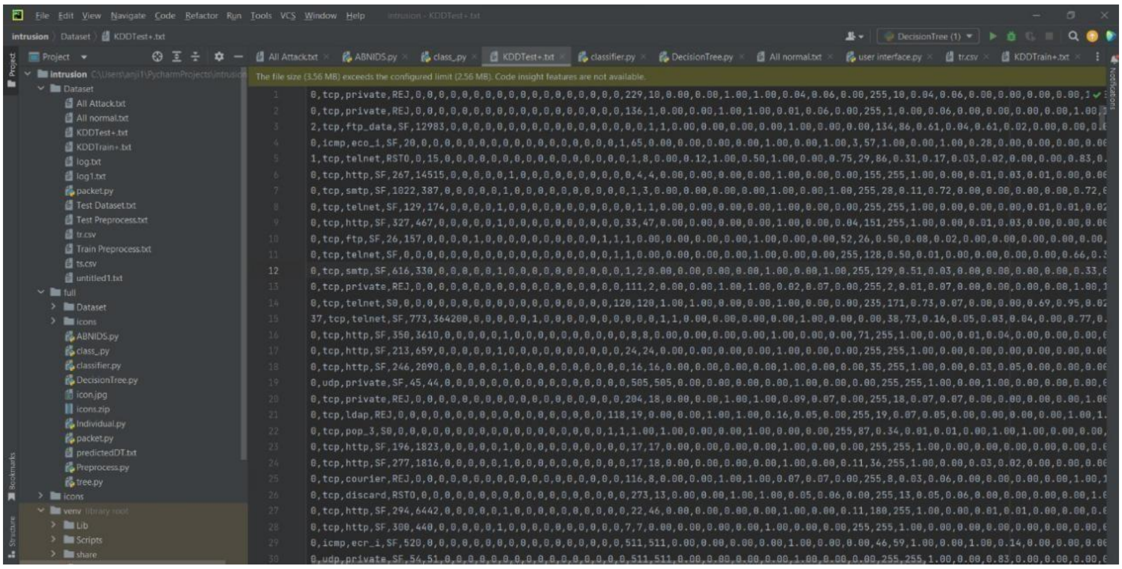


Fig 8.5: KDD Dataset data

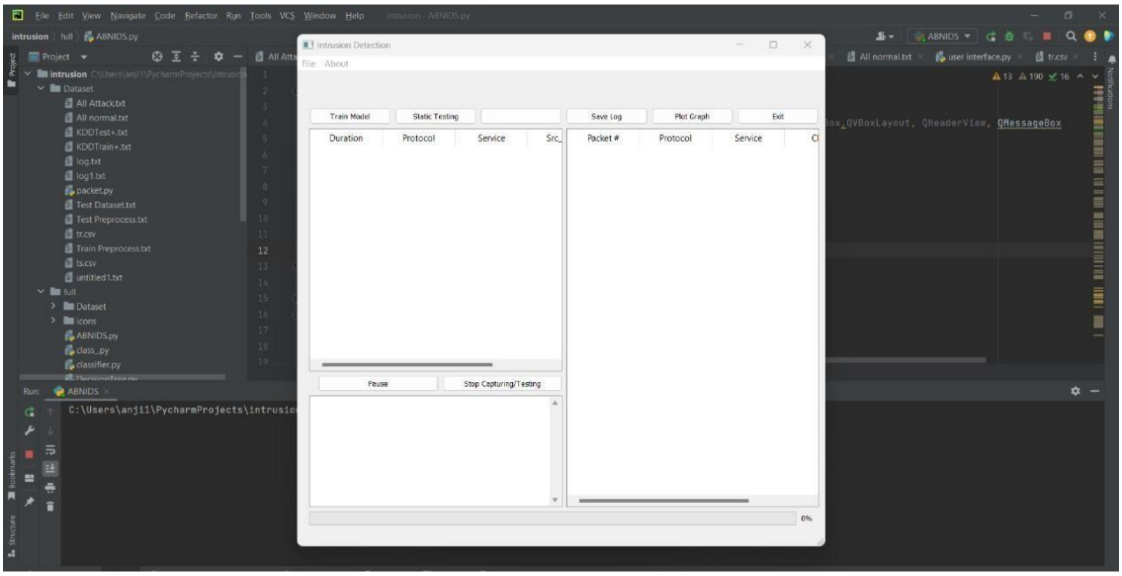


Fig 8.6: Status before running the code

Fig 8.6 shows us the status of the GUI before running the code and training of the model.

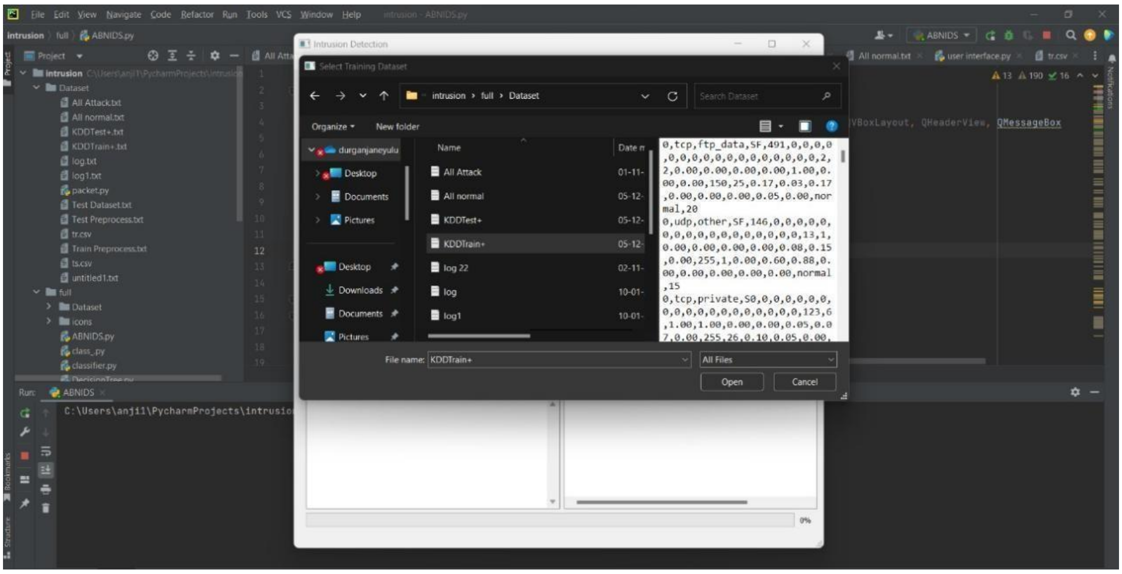


Fig 8.7: Selecting training dataset

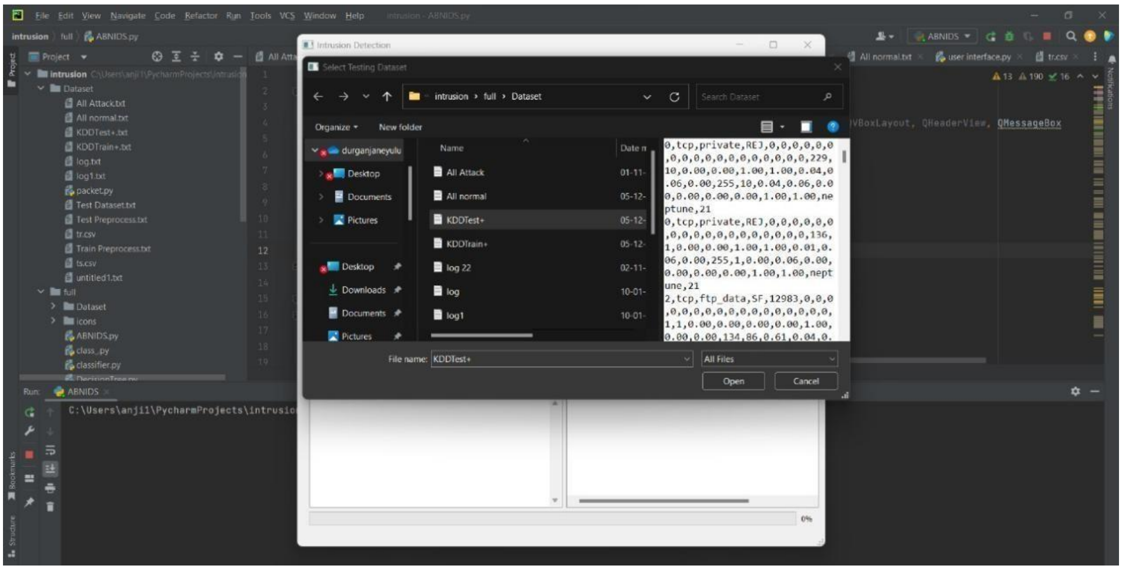


Fig 8.8: Selecting testing dataset

Fig 8.7 and 8.8 both showcase the selection of training and testing dataset while we train the model

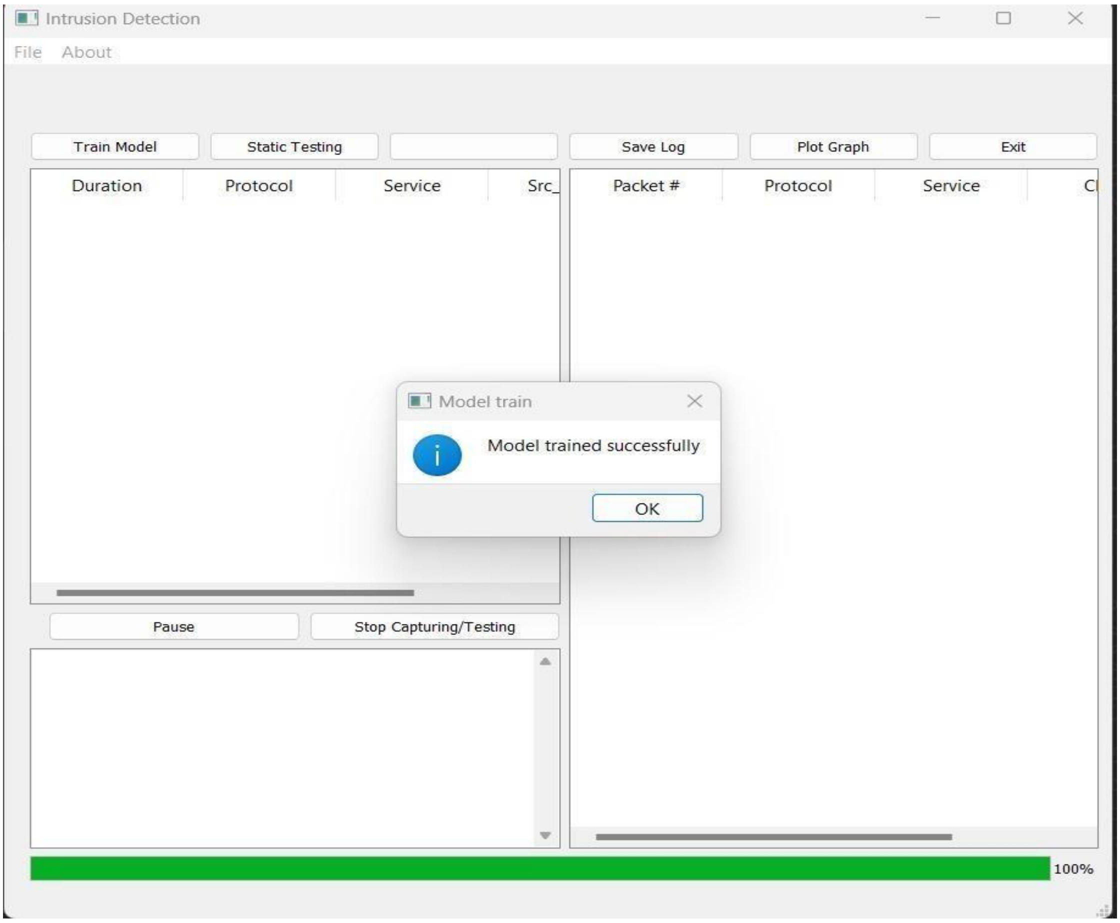


Fig 8.9: Model trained successfully

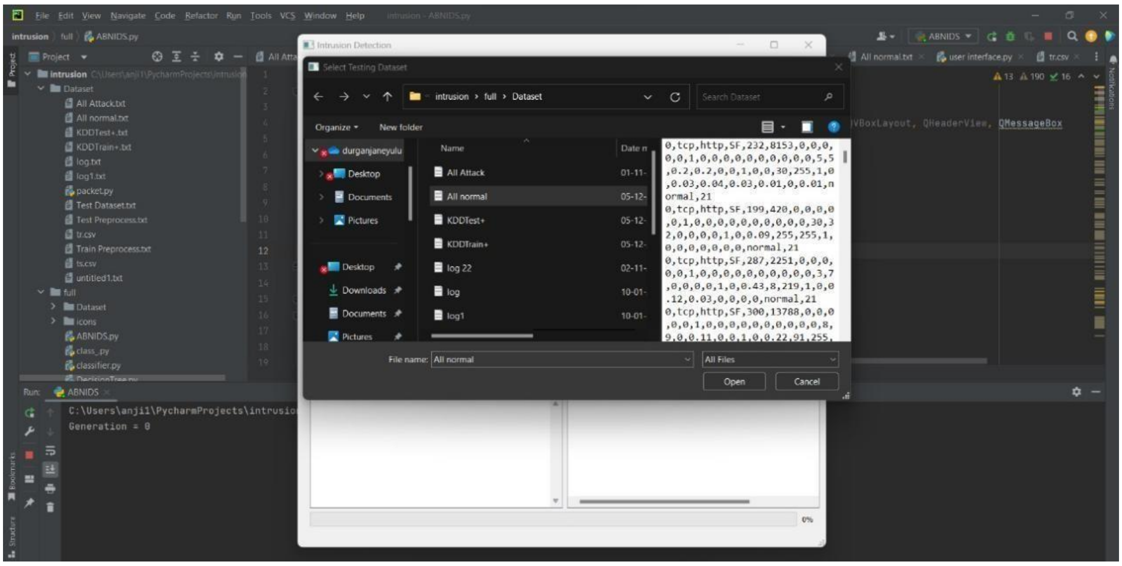


Fig 8.10: Selecting Testing dataset

Fig 8.9 shows the image after the model has been trained successfully in a small dialog box in the GUI.

Fig 8.10 shows us the image while selecting the test data for model testing module.

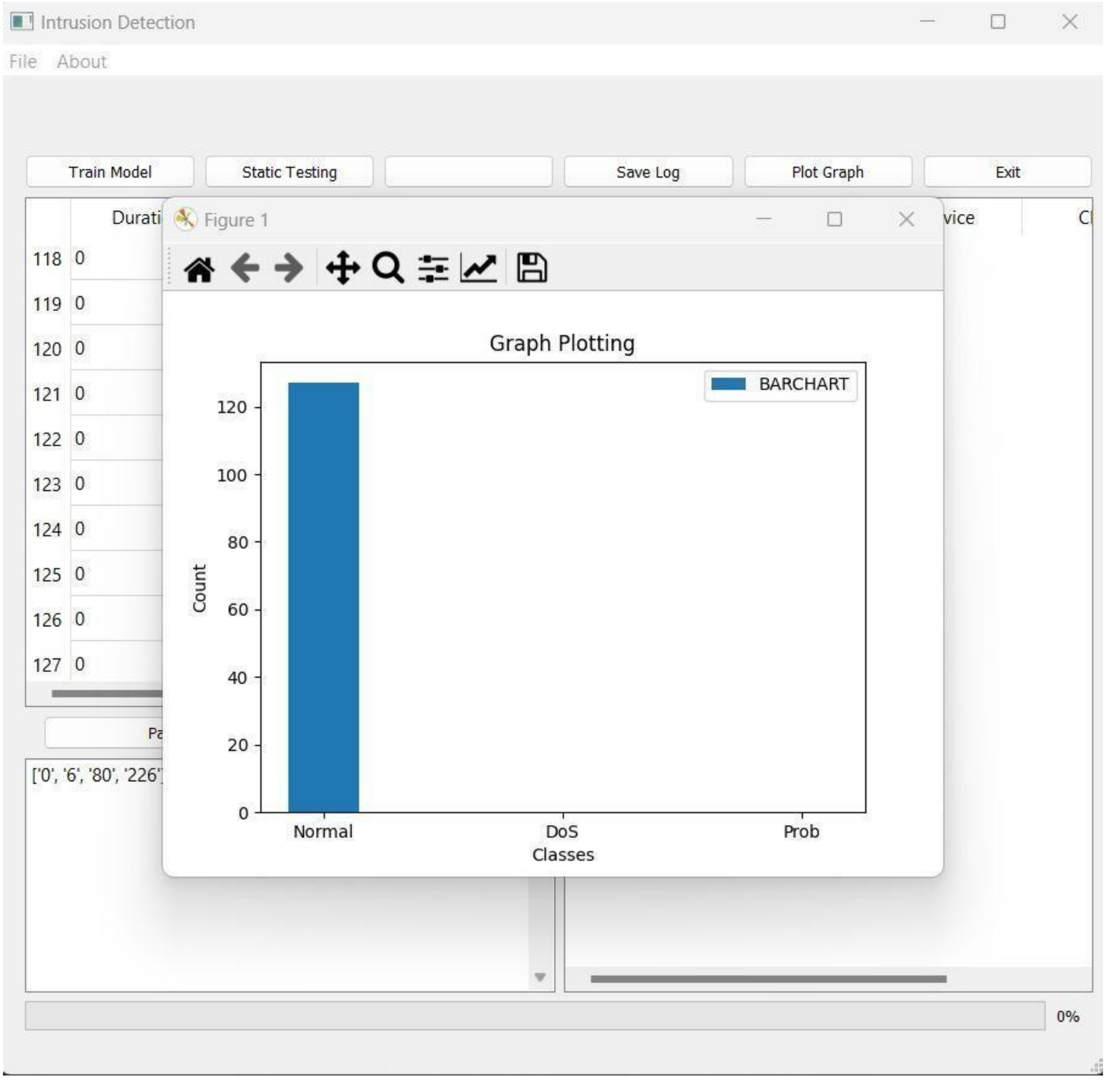


Fig 8.11: Plot graph for normal

Fig 8.11 shows us the graph plot obtained in the GUI for the normal packets present in the dataset.

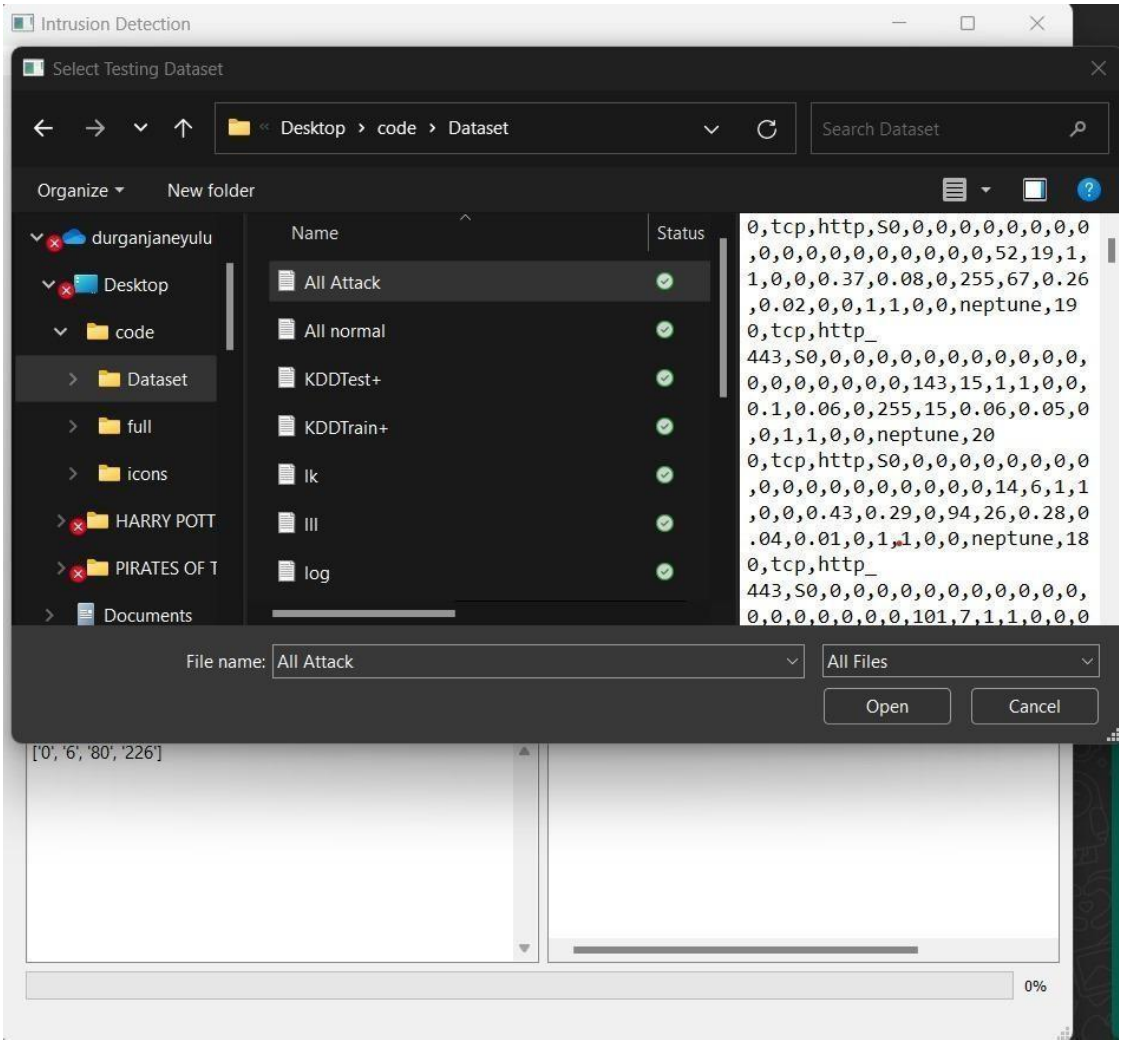


Fig 8.12: Selecting training data for all attack

Fig 8.12 shows the image while selecting the all attack data from the KDD dataset

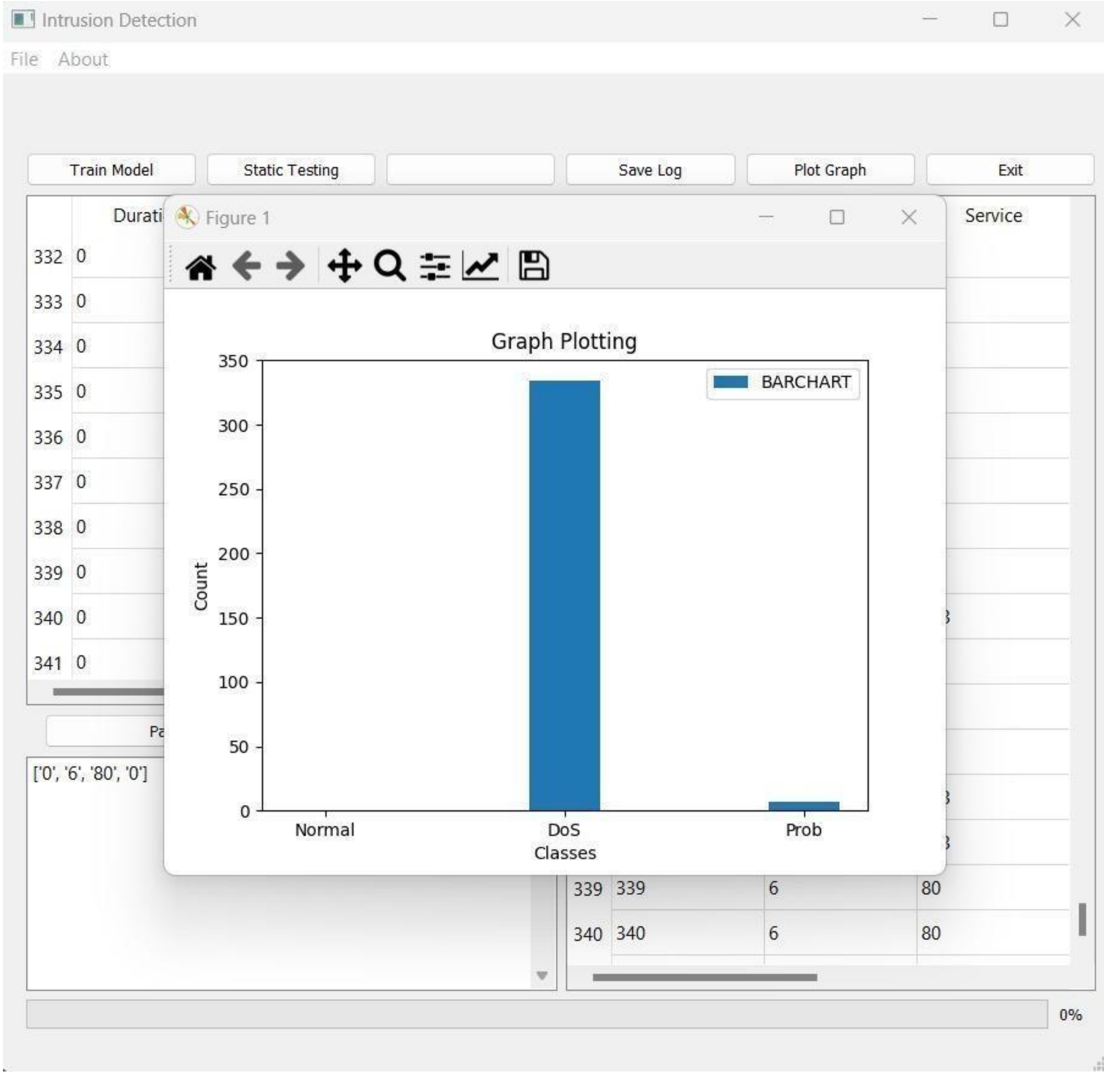


Fig 8.13: Visualization of classes between Dos and Probe in attacked data

Fig 8.13 shows us the graph plot obtained in the GUI for the all attack data i.e. Dos and probe attacks which are present in the dataset.

CHAPTER 9

CONCLUSION

In this study, intrusion detection is performed by GA combined with the decision tree method. The proposed technique is provided in software form. Here, principal component analysis is used to identify key properties of network connectivity, and GA is used to generate classification rules for intrusion detection programs. The GA approach is easy to implement and maintain because it uses complete and simple representations of useful relevance rules and functions. If appropriate attack classification and training datasets are available, the system can also be deployed in a variety of application scenarios due to its flexibility. Since the primary goal of identification is to identify attacks in real-time so they can be stopped before they cause harm, threat classification has no impact on intrusion detection. By applying this approach to intrusion detection in lieu of any additional techniques commonly used with other software computing techniques, benefits include low false positive rates and detection rates high attack. To improve intrusion detection speed and enable applications on high-speed networks, the system exploits only three network connectivity properties while maintaining high detection speed.

CHAPTER 10

REFERENCE

[1] Kumar, Vikash & Das, Ayan & Sinha, Ditipriya. (2021). UIDS: a unified IDS for IoT environment. Evolutionary Intelligence. 14. 10.1007/s12065-019-00291-w.

[2] Gupta, Neha & Jindal, Vinita & Bedi, Punam. (2021). LIO-IDS: Handling class imbalance using LSTM and Improved One-vs-One technique in IDS. Computer Networks. 192. 108076. 10.1016/j.comnet.2021.108076.

[3] Kumar, Munish & Saluja, Krishan. (2021). Intrusion detection techniques in network environment: a systematic review. Wireless Networks. 27. 10.1007/s11276-020-02529-3.

[4] Suhaimi, Hamizan & Suliman, Saiful & Musirin, Professor Dr. Ismail & Harun, Afdallyna & Mohamad, Roslina. (2019). Network IDS by using GA. Indonesian Journal of Electrical Engineering and Computer Science. 16. 1593. 10.11591/ijeecs.v16.i3.pp1593-1599.

[5] Mambwe Sydney, Kasongo. (2021). An advanced IDS for IIoT Based on GA and Tree based Algorithms. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3104113.

[6] Syurahbil, & Ahmad, Noraziah & Zolkipli, Mohamad & Abdalla, Ahmed. (2009). Intrusion Preventing System using IDS Decision Tree Data Mining. American Journal of Engineering and Applied Sciences. 2. 10.3844/ajeassp.2009.721.725.

[7] Gao, Feng, et al. "Connet: Deep semi-supervised anomaly detection based on sparse positive samples." IEEE Access 9 (2021): 67249-67258.

[8] Chen et al. "Toward practical crowdsourcing-based road anomaly detection with scale-invariant feature." IEEE Access 7 (2019): 67666-67678.

[9] Smith, Antony & Du, Shengzhi & Kurien, Anish. (2023). Vision Transformers for Anomaly Detection and Localisation in Leather Surface Defect Classification Based on Low-Resolution Images and a Small Dataset. Applied Sciences. 13. 8716. 10.3390/app13158716.

[10] Chang, Chein-I & Cao, Hongju & Song, Meiping. (2021). Orthogonal Subspace Projection Target Detector for Hyperspectral Anomaly Detection. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. PP. 1-1. 10.1109/JSTARS.2021.3068983.

[11] Yoon, Seunghyun & Cho, Jin-Hee & Kim, Dan & Moore, Terrence & Free-Nelson, Frederica & Lim, Hyuk. (2020). Attack Graph-Based Moving Target Defense in Software-Defined Networks. IEEE Transactions on Network and Service Management. PP. 1-1. 10.1109/TNSM.2020.2987085.

[12] Xie, Chen & Yang, Kecheng & Wang, Anni & Chen, Chunxu & Li,Wei. (2021). A Mura Detection Method Based on an Improved Generative Adversarial Network. IEEEAccess.9.68826-68836.10.1109/ACCESS.2021.3076792.

[13] Shajari, Mehdi, et al. "Tensor-based online network anomaly detection and diagnosis." IEEE Access 10 (2022): 85792-85817.

[14] Kim, Hwan & Lee, Byung & Shin, Won-Yong & Lim, Sungsu. (2022). Graph Anomaly Detection With Graph Neural Networks: Current Status and Challenges. IEEE Access. PP. 1-1. 10.1109/ACCESS.2022.3211306.

[15] Lundström, Adam, et al. "Improving deep learning based anomaly detection on multivariate time series through separated anomaly scoring." IEEE Access 10 (2022): 108194-108204.

[16] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R.Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning-based DDoS detection in SDN," Future Generation Computer Systems, vol. 111, pp. 763-779, 2020, Doi: 10.1016/j.future.2019.10.015.

[17] Gaur, Kuntal & Choudhary, Pranjal & Yadav, Priya & Jain, Ayush & Kumar, Pradeep. (2021). Software Defined Networking: A review on Architecture, Security and Applications. IOP Conference Series: Materials Science and Engineering. 1099. 012073. 10.1088/1757-899X/1099/1/012073.

[18] 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.

[19] M. Nobakht, V. Sivaraman, and R. Borelli, "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.

[20] M. S. Elsayed, N. Le-Khac, S. Dev, and A. D. Jurcut, "ML Techniques for Detecting Attacks in SDN," in 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), 19-20 Oct. 2019, 2019, pp. 277-281, Doi: 10.1109/ICCSNT47585.2019.8962519.

[21] Choudhary S, Kesswani N. 2020. Analysis of KDD-Cup’ 99, NSL-KDD and UNSW-NB15 datasets using deep learning in IoT. Procedia Computer Science 167(2019):1561-1573.

[22] Dahiya, Priyanka & Srivastava, Devesh. (2018). Network Intrusion Detection in Big Dataset Using Spark. Procedia Computer Science. 132. 253-262. 10.1016/j.procs.2018.05.169.

[23] Dlamini G, Fahim M. 2021. DGM: a data generative model to improve minority class presence in anomaly detection domain. Neural Computing and Applications 33(20):13635-13646.

[24] Faker, Osama & Dogdu, Erdogan. (2019). Intrusion Detection Using Big Data and Deep Learning Techniques. 10.1145/3299815.3314439.

[25] Gupta, Dikshant & Singhal, Suhani & Malik, Shamita & Singh, Archana. (2016). Network IDS using various data mining techniques. 1-6.
10.1109/RAINS.2016.7764418.

[26] Aliyu, Ibrahim & van Engelenburg, Sélinde & Mu'Azu, Mb & Kim, Jinsul & Lim, Chang-Gyoon. (2022). Statistical Detection of Adversarial Examples in Blockchain-Based Federated Forest In-Vehicle Network IDSs. IEEE Access. PP. 1-1. 10.1109/ACCESS.2022.3212412. .

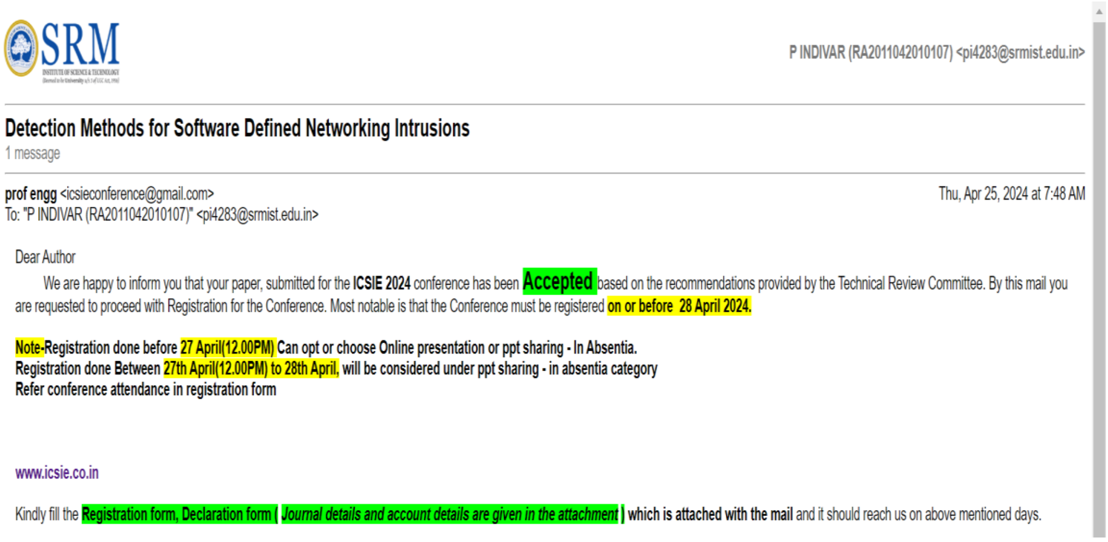
[27] Nadeem, Waqas & Goh, Hock Guan & Yc, Aun & Ponnusamy, Vasaki. (2023). Detecting and Mitigating Botnet Attacks in Software-Defined Networks Using Deep Learning Techniques. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3277397. .

[28] Shin, Gun-Yoon & Kim, Dong-Wook & Han, Myung-Mook. (2022). Data Discretization and Decision Boundary Data Point Analysis for Unknown Attack Detection. IEEE Access. PP. 1-1. 10.110

CHAPTER 11

RESEARCH PAPER ACCEPTANCE

Our Research paper on **Detection Methods for Software Defined Networking Intrusions** was submitted for ICSIE 2024 conference and the paper has been accepted by their technical review committee. The proof of acceptance has been attached below:



SRM

P INDIVAR (RA2011042010107) <pi4283@srmist.edu.in>

Detection Methods for Software Defined Networking Intrusions

1 message

prof engg <icsieconference@gmail.com> Thu, Apr 25, 2024 at 7:48 AM
To: "P INDIVAR (RA2011042010107)" <pi4283@srmist.edu.in>

Dear Author

We are happy to inform you that your paper, submitted for the **ICSIE 2024** conference has been **Accepted** based on the recommendations provided by the Technical Review Committee. By this mail you are requested to proceed with Registration for the Conference. Most notable is that the Conference must be registered **on or before 28 April 2024.**

Note-Registration done before 27 April(12.00PM) Can opt or choose Online presentation or ppt sharing - In Absentia.
Registration done Between 27th April(12.00PM) to 28th April, will be considered under ppt sharing - in absentia category
Refer conference attendance in registration form

www.icsie.co.in

Kindly fill the **Registration form, Declaration form (*Journal details and account details are given in the attachment*) which is attached with the mail** and it should reach us on above mentioned days.

Fig 11.1: Research paper acceptance proof

CHAPTER 12

PLAGIARISM REPORT

Updated report doc for plag check 107,86

ORIGINALITY REPORT

9%	7%	4%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	scialert.net Internet Source	1%
2	Submitted to Higher Education Commission Pakistan Student Paper	1%
3	researchonline.ljmu.ac.uk Internet Source	1%
4	www.mdpi.com Internet Source	1%
5	www.researchgate.net Internet Source	1%
6	Submitted to University of Southampton Student Paper	1%
7	fastercapital.com Internet Source	1%
8	Submitted to University of Michigan, Dearborn Student Paper	<1%

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES

(To be attached in the dissertation/ project report)

1	Name of the Candidate **(IN BLOCKLETTERS)**	SIGIRALA NAVEEN P INDIVAR
2	Address of the Candidate	201, Smiles Inn PG, Potheri, Chengalpattu, Tamil Nadu-603203 A501, Akshaya Metropolis, Maraimalai Nagar, Chengalpattu, Tamil Nadu-603209
3	Registration Number	RA2011042010086 RA2011042010107
4	Date of Birth	10 JANUARY,2002 30 AUGUST, 2002
5	Department	DATA SCIENCE AND BUSINESS SYSTEMS
6	Faculty	ENGINEERING AND TECHNOLOGY, SCHOOL OF COMPUTING
7	Title of the Dissertation/Project	DETECTION METHODS FOR SOFTWARE DEFINED NETWORK INTRUSIONS
8	Whether the above project /dissertationis done by	Individual or group : (Strike whichever is not applicable) **a)** If the project/ dissertation is done in group, then how many students together completed the project : 2(TWO) b) Mention the Name & Register number ofother candidates : SIGIRALA NAVEEN (RA2011042010086) P INDIVAR(RA2011042010107)
9	Name and address of the Supervisor /Guide	Dr.K.PRIYADARSINI, Associate Professor, Department of Data Science and Business Systems, School of Computing, Faculty of Engineering and Technology, SRM Institute of Science and Technology Kattankulathur-603203 **Mail ID:** priyadak@srmist.edu.in **Mobile Number:**9791515576
10	Name and address of Co-Supervisor /Co- Guide (if any)	**NIL**

11	Software Used	TURNITIN		
12	Date of Verification			
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self-citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	INTRODUCTION	1%	1%	1%
2	SYSTEM ANALYSIS	1%	1%	1%
3	LITERATURE REVIEW	1%	1%	1%
4	SYSTEM DESIGN	0%	1%	0%
5	USE CASE DIAGRAMS	1%	1%	1%
6	LIST OF REQUIREMENTS SPECIFICATIONS	1%	1%	1%
7	CODE IMPLEMENTATION	1%	1%	1%
8	OUTPUT	1%	0%	1%
9	CONCLUSION	0%	1%	0%
10	REFERENCE	1%	1%	1%

We declare that the above information has been verified and found true to the best of our knowledge.

Signature of the Candidates	**Name & Signature of the Staff (Who uses the plagiarism check software)**
Name & Signature of the Supervisor/ Guide	**Name & Signature of the Co-Supervisor /Co-Guide**

Name & Signature of the HOD