CYBER ATTACK DETECTION MODEL FOR NETWORK INTRUSION USING MACHINE LEARNING TECHNIQUE

Major Project submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Mr. Y. Siva
Assistant Professor

By

VUDHANTHI NEERAJA (19R11A0548)
VANKAYALAPATI VINITHA (19R11A0544)
KAMUJU NOVA (19R11A0517)

Department of Computer Science and Engineering
Accredited by NBA

Geethanjali College of Engineering and Technology
(UGC Autonomous)
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.
May-2023

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Accredited by NBA

CERTIFICATE

This is to certify that the B.Tech Major Project report entitled **"CYBER ATTACK DETECTION MODEL FOR NETWORK INTRUSION USING MACHINE LEARNING TECHNIQUE"** is a bonafide work done by **Vudhanthi Neeraja (19R11A0548), Vankayalapati Vinitha (19R11A0544), Kamuju Nova (19R11A0517)** in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering**" from Jawaharlal Nehru Technological University, Hyderabad during the year 2022-2023.

Internal Guide	**HOD-CSE**
Mr. Y. Siva	**Dr. A. Sree Lakshmi**
Assistant Professor	**Professor**

External Examiner

Geethanjali College of Engineering & Technology

(UGC Autonomous)
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

DECLARATION BY THE CANDIDATE

We, **Vudhanthi Neeraja, Vankayalapati Vinitha, Kamuju Nova,** bearing Roll Nos. **19R11A0548 ,19R11A0544 ,19R11A0517** , hereby declare that the project report entitled "**CYBER ATTACK DETECTION MODEL FOR NETWORK INTRUSION USING MACHINE LEARNING TECHNIQUE**" is done under the guidance of **Mr. Y. Siva, Assistant Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering.**

This is a record of bonafide work carried out by us in **Geethanjali College of Engineering and Technology** and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Vudhanthi Neeraja (19R11A0548)
Vankayalapati Vinitha (19R11A0544)
Kamuju Nova (19R11A0517)

ACKNOWLEDGEMENT

We are greatly indebted to the authorities of Geethanjali College of Engineering and Technology, Cheeryal, Medchal District, for providing us the necessary facilities to successfully carry out this main project work titled “**CYBER ATTACK DETECTION MODEL FOR NETWORK INTRUSION USING MACHINE LEARNING TECHNIQUE**”.

Firstly, we thank and express our solicit gratitude to our HOD, **Dr. A. SREE LAKSHMI**, Professor, CSE department, Geethanjali College of Engineering and Technology, for her invaluable help and support which us a lot in successfully completing our main project.

Secondly, we express our gratitude to **Mr. Y. Siva**, Assistant Professor, internal guide, Geethanjali College of Engineering and Technology, for his suggestion and encouragement which helped us in the successful completion of our main project.

We would like to express our sincere gratitude to our Principal **Dr. S. UDAYA KUMAR** for providing the necessary infrastructure to complete our project.

We convey our gratitude to our Chairman, **Mr. G. RAVINDER REDDY**, for his invaluable support and encouragement for propelling our innovation forward. Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our goals.

Vudhanthi Neeraja (19R11A0548)

Vankayalapati Vinitha (19R11A0544)

Kamuju Nova (19R11A0517)

ABSTRACT

A network intrusion is an unauthorized penetration of a computer in your enterprise or an address in your assigned domain. Some intrusions are simply meant to let you know the intruder was there, defacing your Web site with various kinds of messages or crude images. Others are more malicious, seeking to extract critical information on either a one-time basis or as an ongoing parasitic relationship that will continue to siphon off data until it's discovered. An Intrusion Detection System (IDS) is a device or software application that monitors a network for malicious activity or policy violations. Signature-based IDS is the detection of attacks by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware. Anomaly-based intrusion detection systems were primarily introduced to detect unknown attacks, in part due to the rapid development of malware.

The major problems in the current Intrusion Detection Systems are that: Although signature-based IDS can easily detect known attacks, it is difficult to detect new attacks, for which no pattern is available. While the anomaly-based intrusion detection approach enables the detection of previously unknown attacks, it may suffer from the time-consumption during detection process that degrades the performance of IDS's.

To overcome these challenges of detecting new attacks and the problem of time-consumption during detection, we put forth an automated approach that can repetitively extract attack signatures and evaluate these signatures based on the detection rules out of complex data and large network data volumes. In a large dataset not, all features contribute to represent the traffic, therefore reducing and selecting a number of adequate features improves the speed and accuracy of the intrusion detection system. In our project, a feature selection mechanism has been proposed which aims to eliminate non-relevant features as well as identify the features which will contribute to improve the detection rate, based on the score each feature have established during the selection process. To achieve that objective, a recursive feature elimination process will be employed and associated with a decision tree-based classifier.

LIST OF FIGURES

S.No.	Figure Name	Page No.
1	Architecture of Signature-based Intrusion Detection System	3
2	Architecture of Anomaly-based Intrusion Detection System	4
3	Modular data flow of the Intrusion Detection System	10
4	The architecture of the intrusion detection system	21
5	The process of the IDS rule module	22
6	Level 0 DFD of Intrusion Detection System	23
7	Level 1 DFD of Intrusion Detection System	25
8	Use Case Diagram of Intrusion Detection System	27
9	Class Diagram of Intrusion Detection System	29
10	Sequence Diagram of Intrusion Detection System	31
11	Activity Diagram of Intrusion Detection System	32
12	State Diagram of Intrusion Detection System	34
13	Sample NSL-KDD dataset	51

LIST OF TABLES

S.No.	Table Name	Page No.
1.	Features selected using ANOVA F-Test for each attack	12
2.	Prediction and Evaluation results.	14
3.	Test Cases for Intrusion Detection System	56

LIST OF SCREEN SHOTS

S.No.	Description of Screen Shot	Page No.
1.	Webpage capturing packets on the Wi-Fi interface and displays the network details.	57
2.	Webpage showing the frequency graph of the detected attacks.	58
3.	Webpage shows that Send Report button is pressed for report generation.	58
4.	Gmail shows that the report generated is sent to the user email id.	59
5.	Email showing the MAC address of source and destination and attachment of report.	59
6.	Generated report.	60

TABLE OF CONTENTS

S.No.	CONTENTS	Page No.
	Abstract	**ii**
	List of Figures	**iii**
	List of Tables	**iv**
	List of Screen shots	**v**
1.	**INTRODUCTION**	**1**
	1.1 About the project	1
	1.2 Objective of the project	2
2.	**SYSTEM ANALYSIS**	**3**
	2.1 Existing System	3
	2.2 Proposed System	5
	2.2.1 Details	7
	2.2.2 Impact on Environment	8
	2.2.3 Safety	8
	2.2.4 Ethics	8
	2.2.5 Cost	8
	2.2.6 Type	9
	2.2.7 Standards	9
	2.3 Scope of Project	10
	2.4 Modules Description	10
	2.5 System Configuration	15
3.	**LITERATURE OVERVIEW**	**16**
	3.1 Feasibility Study	16
	3.2 Project Literature	17
4.	**SYSTEM DESIGN**	**21**

4.1 System Architecture 21

4.2 UML Diagrams 23

4.3 System Design 35

5. SAMPLE CODE 36

5.1 Coding 36

5.2 Implementation 49

6. TESTING 57

6.1 Testing 57

6.2 Test Cases 58

7. OUTPUT SCREENS 59

8. CONCLUSION 63

8.1 Conclusion 63

8.2 Future Enhancements 64

9. BIBLIOGRAPHY 65

10. APPENDICES 67

11. PLAGARISM REPORT 69

1. INTRODUCTION

1.1 ABOUT THE PROJECT

An attack, via cyberspace, targeting an enterprise's use of cyberspace for the purpose of disrupting, disabling, destroying, or maliciously controlling a computing environment/infrastructure; or destroying the integrity of the data or stealing controlled information is called a Cyber Attack. These attacks are often carried out ingesting malicious queries and transacts the information from the target system and/or violates policies of the network. These attacks on the network without authorization is called "Network Intrusion". A system called an intrusion detection system (IDS) observes network traffic for malicious transactions, policy violations and sends immediate alerts when it is observed and protects a computer network from unauthorized access. Some intrusions are simply meant to let you know the intruder was there, defacing your Web site with various kinds of messages or crude images. Others are more malicious, seeking to extract critical information on either a one-time basis or as an ongoing parasitic relationship that will continue to siphon off data until it's discovered.

An Intrusion Detection System (IDS) is a device or software application that monitors a network for malicious activity or policy violations. Signature-based IDS is the detection of attacks by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware. Anomaly-based intrusion detection systems were primarily introduced to detect unknown attacks, in part due to the rapid development of malware. The major problems in the current Intrusion Detection Systems are that: Although signature-based IDS can easily detect known attacks, it is difficult to detect new attacks, for which no pattern is available. While the anomaly-based intrusion detection approach enables the detection of previously unknown attacks, it may suffer from the time-consumption during detection process that degrades the performance of IDS's.

To overcome these challenges of detecting new attacks and the problem of time-consumption during detection, we put forth an automated approach that can repetitively extract attack signatures and evaluate these signatures based on the detection rules out of complex data and large network data volumes. In a large dataset not, all features contribute to represent the traffic, therefore reducing and selecting a number of adequate features improves the speed and accuracy of the intrusion detection system.

In our proposed system, a feature selection mechanism has been proposed which aims to eliminate non-relevant features as well as identify the features which will contribute to improve the detection rate, based on the score each feature have established during the selection process. To achieve that objective, a recursive feature elimination process will be employed and associated with a decision tree-based classifier.

By using this approach, we can significantly improve the performance of IDSs and reduce the likelihood of successful cyber attacks. This approach has the potential to detect new and unknown attacks while reducing the amount of data that needs to be processed, making it faster and more efficient than existing IDSs. In addition, it can be adapted to different types of network environments and can be customized to meet the specific needs of different organizations.

1.2 OBJECTIVE OF THE PROJECT

Cyber-attacks have become increasingly sophisticated, making it essential to have effective intrusion detection systems in place. The use of machine learning algorithms, such as decision trees, has shown promise in detecting network intrusions. However, selecting the most relevant features for such a model can be challenging due to the high dimensionality of the data. This project aims to develop a cyber-attack detection model using decision trees, NSL KDD dataset, and ANOVA F-test for recursive feature elimination. The objective is to determine whether this model can accurately detect network intrusions while minimizing false positives and false negatives, and to evaluate the effectiveness of the ANOVA F-test for feature selection in improving the model's performance.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

At present there are two types of voting methods, they are:

- Signature-based Intrusion Detection System (IDS)
- Anomaly-based Intrusion Detection System (IDS)

A. Signature-based Intrusion Detection System (IDS)

A signature-based intrusion detection system (IDS) is a type of IDS that identifies potential intrusions by comparing network traffic and system logs against known attack patterns, or signatures. These signatures are essentially a set of rules that describe the characteristics of known attacks, including specific protocol fields, data patterns, or behavior patterns. When network traffic matches a signature, the IDS raises an alert indicating a potential intrusion.

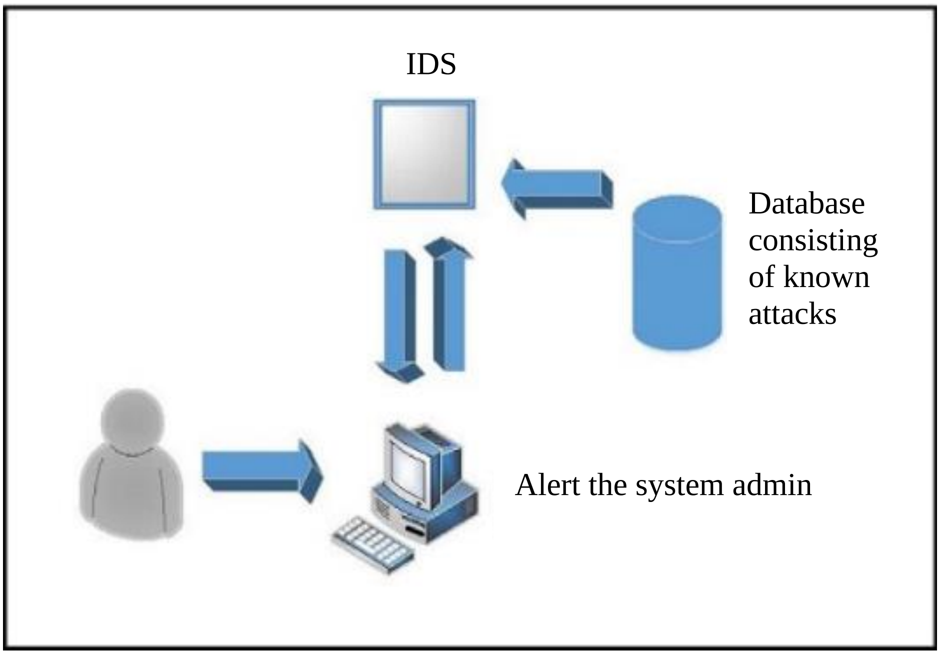


Fig 1.1 Architecture of Signature-based Intrusion Detection System

The advantage of a signature-based IDS is that it can detect known attacks accurately and in real-time, as long as the signature database is up to date. This makes it a useful tool for preventing attacks that have already been identified and documented.

However, signature-based IDSs also have several limitations. They cannot detect new, unknown attacks or variations of known attacks that have been modified to evade detection. Additionally, creating and maintaining a comprehensive signature database can be challenging and time-consuming, and false positives can occur if legitimate network traffic matches a signature.

Despite these limitations, signature-based IDSs are still widely used as a component of a larger security strategy, alongside other types of intrusion detection and prevention technologies.

B. Anomaly-based Intrusion Detection System (IDS)

An anomaly-based intrusion detection system (IDS) is a type of IDS that identifies potential intrusions by analyzing network traffic and system behavior for deviations from established normal patterns. Unlike signature-based IDSs, which rely on known attack patterns, anomaly-based IDSs detect abnormal or suspicious activities that may indicate a previously unknown attack or behavior. Anomaly-based IDSs use various techniques such as statistical analysis, machine learning, and rule-based systems to establish a baseline of normal behavior and identify deviations from it. This baseline can be established by analyzing historical network traffic, system logs, or user behavior, and can be updated periodically to reflect changes in the network environment.

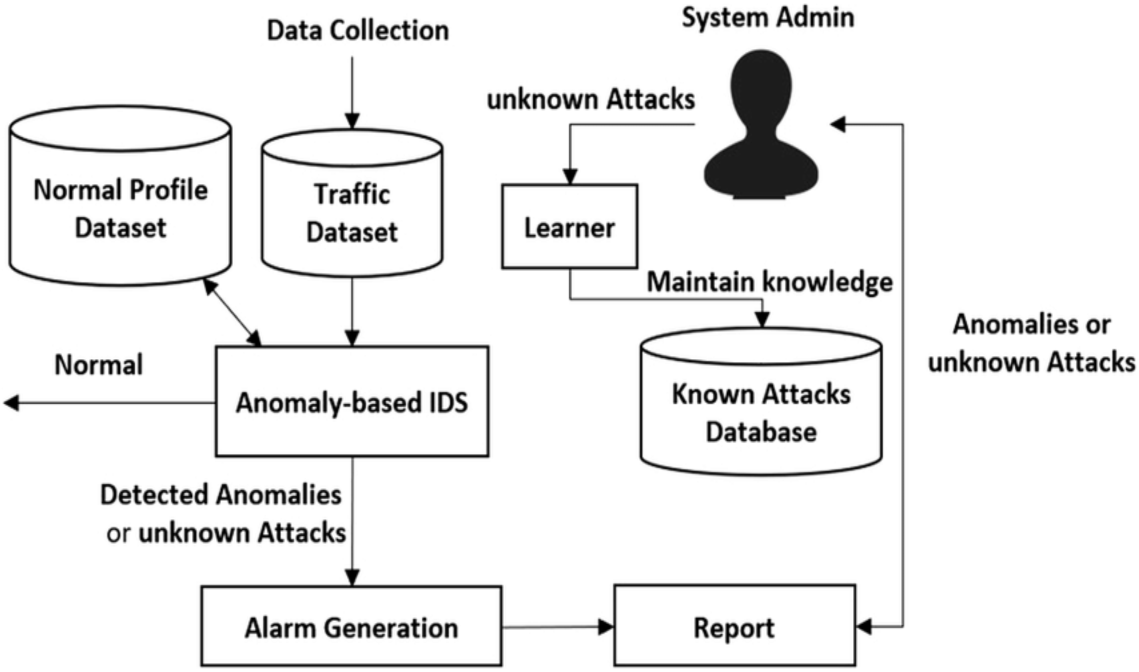


Fig 1.2 Architecture of Anomaly-based Intrusion Detection System

One advantage of anomaly-based IDSs is that they can detect previously unknown attacks or variations of known attacks, as long as these activities deviate significantly from established normal patterns. Additionally, anomaly-based IDSs have the potential to adapt to changes in the network environment and detect new threats that may be missed by signature-based IDSs.

However, anomaly-based IDSs also have some limitations. They can produce a high number of false positives, particularly in complex and dynamic environments. Additionally, anomaly-based IDSs can be challenging to configure and require ongoing maintenance to ensure that they accurately reflect normal network behavior.

Despite these limitations, anomaly-based IDSs are becoming increasingly popular as a complement to signature-based IDSs, providing an additional layer of defense against network threats.

2.1.1 Disadvantages of existing system

- Signature-based Intrusion Detection System (IDS):
   i. They cannot detect new, unknown attacks or variations of known attacks that have been modified to evade detection.
   ii. Additionally, creating and maintaining a comprehensive signature database can be challenging and time-consuming, and false positives can occur if legitimate network traffic matches a signature.
- Anomaly-based Intrusion Detection System (IDS):
   i. They can produce a high number of false positives, particularly in complex and dynamic environments.
   ii. Additionally, anomaly-based IDSs can be challenging to configure and require ongoing maintenance to ensure that they accurately reflect normal network behavior.

2.2 PROPOSED SYSTEM

With the emergence of new technologies such as Cloud Computing or Big Data, the amount of network traffic generated has increased significantly. This poses a challenge for intrusion detection systems (IDS) which must collect and analyze this data in real-time to identify potential threats. However, not all features of the data are relevant to identifying intrusions, which can lead to slow processing times and reduced accuracy. To address this challenge, our project proposes a feature selection mechanism that aims to eliminate non-relevant features and identify the features that contribute to improving the detection rate. The proposed mechanism combines two methods: Univariate feature selection using ANOVA with F-test and recursive feature elimination.

Univariate feature selection using ANOVA with F-test is a statistical method that ranks features based on their correlation with the target variable, in this case, identifying intrusions. This method selects features with the highest F-score and eliminates those that are not statistically significant.

Recursive feature elimination builds on the previous feature selection by iteratively removing the least important features and training the classifier on the remaining features until a desired number of features is achieved.

The selected features are then used to train a decision tree-based classifier, which is a popular algorithm for intrusion detection due to its ability to handle high-dimensional data and classify complex patterns.

By selecting relevant features from the incoming traffic, the mechanism aims to improve the speed and accuracy of intrusion detection systems.

Univariate feature selection using ANOVA with F-test is a technique that ranks features based on their correlation with the target variable, which in this case is the identification of network intrusions. The method selects features with the highest F-score, which indicates the strength of the relationship between the feature and the target variable. Features with low F-scores are then eliminated, as they are not statistically significant.

The second component of our feature selection mechanism is recursive feature elimination. This method builds on the previous feature selection by iteratively removing the least important features and training the classifier on the remaining features until a desired number of features is achieved. Recursive feature elimination improves the accuracy of the feature selection process by taking into account the importance of each feature in combination with other features.

Once the relevant features have been selected, the next step is to train a classifier. In our project, we have chosen a decision tree-based classifier. Decision trees are a popular algorithm for intrusion detection because they are able to handle high-dimensional data and classify complex patterns. The classifier is trained on the selected features and can be used to identify network intrusions in real-time.

Overall, the proposed feature selection mechanism in our project is designed to improve the speed and accuracy of intrusion detection systems in processing large datasets. By selecting relevant features, our mechanism reduces processing times and eliminates non-relevant features that can negatively impact accuracy. The combination of Univariate feature selection using ANOVA with F-test and recursive feature elimination, along with a decision tree-based classifier, provides a reliable and efficient means of identifying network intrusions.

2.2.1 Details

With the increasing frequency and sophistication of cyber attacks, there is a growing need for effective intrusion detection systems (IDS) that can quickly and accurately identify potential threats. One of the major challenges associated with IDS is processing large datasets in real-time to identify network intrusions. The presence of non-relevant features in these datasets can lead to slow processing times and reduced accuracy, which can be detrimental to the overall effectiveness of the system. To address these challenges, our project aims to develop a cyber attack detection model for network intrusion using a decision tree-based classifier, the NSL-KDD dataset, and a feature selection mechanism based on Univariate feature selection using ANOVA with F-test for recursive feature elimination.

The NSL-KDD dataset is a widely used dataset in the field of intrusion detection and contains a large amount of traffic data that can be used to train and test the classifier. However, the large size of the dataset poses a challenge in terms of feature selection and processing time. To address these challenges, our feature selection mechanism combines Univariate feature selection using ANOVA with F-test and recursive feature elimination to identify the most relevant features for intrusion detection. This approach selects features based on their correlation with the target variable, which in this case is the identification of network intrusions. The selected features are then used to train a decision tree-based classifier, which can be used to identify potential threats in real-time.

The main objective of our project is to develop a cyber attack detection model that can accurately and efficiently identify potential threats in large datasets. The model will be evaluated using various performance metrics, including accuracy, precision, recall, and F1 score, to determine its effectiveness in detecting network intrusions. By addressing the challenges associated with feature selection and processing time, our project aims to contribute to the development of more effective intrusion detection systems for improved cybersecurity.

2.2.2 Impact On Environment

- There is no impact on the environment.
- Doesn’t cause damage to environment.

2.2.3 Safety

The IDS project has a significant impact on various areas, including security, privacy, and network integrity. The implementation of the IDS improves security by detecting and preventing potential attacks and protecting sensitive data. It also helps to maintain the privacy of the network and its users by identifying unauthorized access attempts. Overall, the project has a positive impact on the security and integrity of the network, ensuring that it remains secure and protected from potential threats.

2.2.4 Ethics

This project follows the general software and hardware ethics. This system does not harm any individual in any way.

2.2.5 Cost

Since for development we used Python, Scapy, smtplib, Streamlit, and sklearn which are open source they can be used freely, and the time required to develop and test the system depends on the expertise of the development team. The usage cost would involve the hardware and software requirements for running the IDS, as well as any additional costs for data storage and analysis. The maintenance cost would involve the cost of bug fixes, software upgrades, and hardware maintenance, as well as any costs associated with updating the IDS to keep up with new security threats and techniques.

2.2.6 Type

This project is a machine learning based project. The aim of the project is to design/develop an Intrusion Detection System which is feasible to use by the network administrators of a particular network.

2.2.7 Standards

The iterative SDLC model is commonly used in software development projects where requirements are subject to change, and the development team requires flexibility to make adjustments during the project's lifecycle. In the case of an IDS developed using decision trees and ANOVA F-test, an iterative SDLC model can provide several benefits. During the requirements gathering and analysis phase, the team can identify the key features and functionalities of the IDS, and prioritize them based on the level of criticality. This allows the team to focus on the most important features during the first iteration of the development process.

In the design phase, the team can create a high-level design of the system, including the architecture, interfaces, and data flow. This can be refined in each iteration based on feedback and testing results.

In the development phase, the team can create a working prototype of the IDS, including the decision tree and ANOVA F-test algorithms. This prototype can be tested and refined in subsequent iterations based on user feedback, as well as data collected during the testing phase.

In the testing phase, the team can perform both functional and non-functional testing to ensure that the IDS meets the requirements and performs optimally. Any defects or bugs identified can be addressed in subsequent iterations.

The iterative SDLC model allows for continuous improvement of the IDS, providing the team with the flexibility to make changes based on user feedback and testing results. This can ultimately result in a more robust and effective IDS.

2.3 SCOPE OF PROJECT

The aim of our project is to develop an Intrusion Detection system using integrated with the machine learning model that can be set into a network over the internet which is prone to intrusions to identify the attacks that are often carried out by ingesting malicious queries and transacts the information from the target system and/or violates policies of the network.

The scope of this project is:

- To improve the detection rate of the IDS by eliminating non-relevant features as well as identify the features which will contribute.
- To improve the speed and accuracy of the intrusion detection system.
- To ease the trouble of queuing in the course of balloting duration in elections.

2.4 MODULES DESCRIPTION

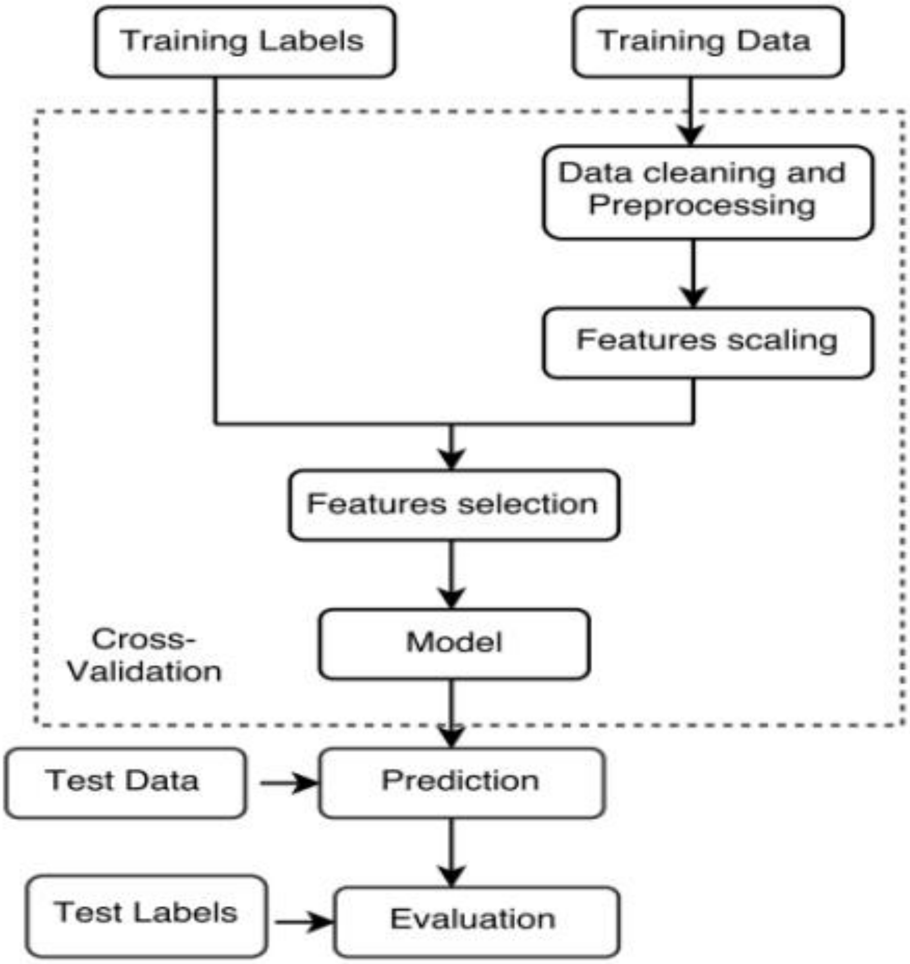


Fig 2 Modular data flow of the Intrusion Detection System

2.4.1 Data Collection

Data collection is defined as the procedure of collecting, measuring and analyzing accurate insights for research using standard validated techniques. A researcher can evaluate their hypothesis on the basis of collected data. In most cases, data collection is the primary and most important step for research, irrespective of the field of research. The approach of data collection is different for different fields of study, depending on the required information. For our project we use data from NSL-KDD dataset for training our facial recognition model. During the implementation of this project in real-time we can use network API's for the purpose of data collection from a particular network.

2.4.2 Data Pre-processing

Data preprocessing in Machine Learning is a crucial step that helps enhance the quality of data to promote the extraction of meaningful insights from the data. Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. Basically, in this step the dataset has to go through a cleaning process to remove duplicate records, as the NSL KDD dataset was employed which has already been cleaned, this step is not anymore required. Next a Pre-processing operation has to be taken in place because the dataset contains numerical and non-numerical instances. Generally, the estimator (classifier) defines in the scikit-learn works well with numerical inputs, so a one-of-K or one-hot encoding method is used to make that transformation. This technique will transforms each categorical feature with m possible inputs to n binary features, with one active at the time only

2.4.3 Feature Scaling and Selection

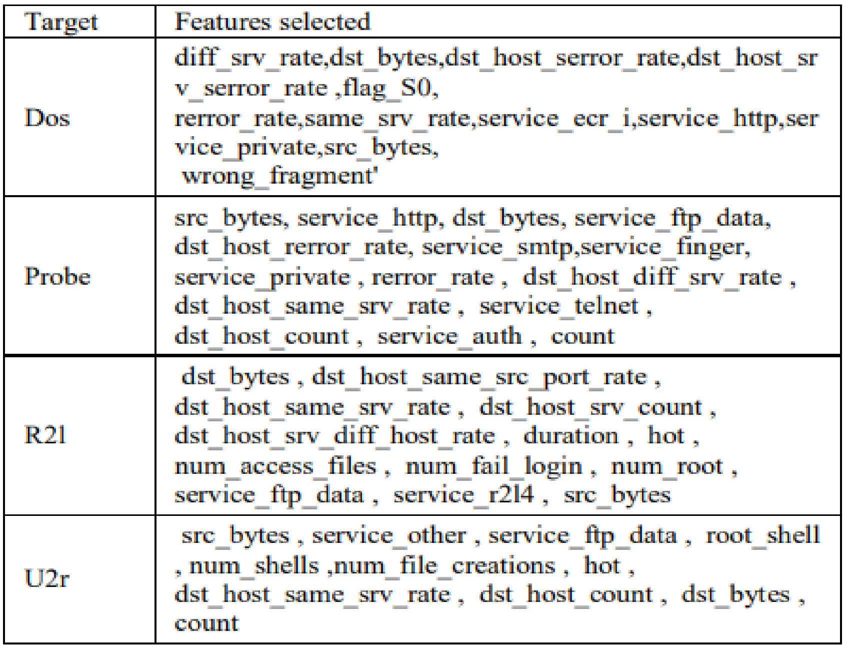
Features scaling

It is a common requirement of machine learning methods, to avoid that features with large values may weight too much on the final results. For each feature, calculate the average, subtract the mean value from the feature value, and divide the result by their standard deviation. After scaling, each feature will have a zero average, with a standard deviation of one.

Feature selection

It is used to eliminate the redundant and irrelevant data. It is a technique of selecting a subset of relevant features that fully represents the given problem alongside a minimum deterioration of presentation, two possible reason were analyzed why it would be recommended to restrict the number of features:

Firstly, it is possible that irrelevant features could suggest correlations between features and target classes that arise just by chance and do not correctly model the problem. This aspect is also related to over-fitting, usually in a decision tree classifier. Secondly, a large number of features could greatly increase the computation time without a corresponding classifier improvement. The feature selection process starts with a univariate feature selection with ANOVA F-test for feature scoring, univariate feature selection analyzes each feature individually to determine the strength of the relationship of the feature with labels. The SelectPercentile method in the sklearn.feature_selection module were used, this method select features based on a percentile of the highest scores. Once, the best subset of features were found, a recursive feature elimination was applied which repeatedly build a model, placing the feature aside and then repeating the process with the remained features until all features in the dataset are exhausted. As such, it is a good optimization for finding the best performing subset of features. The idea is to use the weights of a classifier to produce a feature ranking.



Target	Features selected
Dos	diff_srv_rate,dst_bytes,dst_host_serror_rate,dst_host_srv_serror_rate ,flag_S0, rerror_rate,same_srv_rate,service_ecr_i,service_http,service_private,src_bytes, wrong_fragment'
Probe	src_bytes, service_http, dst_bytes, service_ftp_data, dst_host_rerror_rate, service_smtp,service_finger, service_private , rerror_rate , dst_host_diff_srv_rate , dst_host_same_srv_rate , service_telnet , dst_host_count , service_auth , count
R2l	dst_bytes , dst_host_same_src_port_rate , dst_host_same_srv_rate , dst_host_srv_count , dst_host_srv_diff_host_rate , duration , hot , num_access_files , num_fail_login , num_root , service_ftp_data , service_r2l4 , src_bytes
U2r	src_bytes , service_other , service_ftp_data , root_shell , num_shells ,num_file_creations , hot , dst_host_same_srv_rate , dst_host_count , dst_bytes , count

Table 1 Features selected using ANOVA F-Test for each attack

2.4.4 Build an Intrusion Detection model

Here, a decision tree model was built to partition the data using information gain until instances in each leaf node have uniform class labels. This is a very simple but yet an effective hierarchical method for supervised learning (classification or regression) whereby the local space (region) is recognized in a sequence of repetitive splits in a reduced number of steps (small). At each test, a single feature is used to split the node according to the feature's values. If after the split, for every branches, all the instances selected belong to the similar class, the split is considered complete or pure.

One of the possible method to measure a good split is entropy or information gain. Entropy is an information theoretic measure of the 'uncertainty' found in a training set, because of the existence of more than one possible classification.

The generation process of a decision tree done by recursively splitting on features is equivalent to dividing the original training set into smaller sets recursively until the entropy of every one of these subsets is zero (i.e everyone will have instances from a single class target).

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

In our project, the patterns based on the Indicators of Compromise (IoC) are represented as the branches (which makes up the path from one decision node to the other.) and the type of the attack is given by the leaf nodes.

A test function is implemented by each decision node with a discrete results labelling the branches. Providing an input, at every node, a test is constructed and based on the outcome, one of the branches will be considered. Here the learning algorithm starts at the root and until a leaf node is reached, the process will be done recursively at which moment the value represented in the leaf node is the output. Every leaf node possesses an outcomes label, which it is the class target in case of classification and numeric value for regression. A leaf node can describe a localized space or region where instances finding in this input space (region) possess the same labels for classification and similar numeric value for regression

2.4.5 Prediction and Evaluation

The test data was used to make prediction of our model and for evaluation, multiple settings was considered such as the accuracy score, precision, recall, f-measure and a confusion matrix. A 10-fold cross-validation was performed during all the process.

10-fold cross-validation is a technique used to evaluate the performance of a machine learning model. It involves dividing the available dataset into 10 subsets of approximately equal size. The model is trained on 9 of these subsets and tested on the remaining subset. This process is repeated 10 times, with each subset being used once as the test set. The results of each iteration are then averaged to obtain an overall measure of the model's performance.

The advantage of 10-fold cross-validation is that it provides a more reliable estimate of a model's performance than a simple train-test split. By averaging the results over 10 iterations, it reduces the risk of overfitting or underfitting the model to the data. It also ensures that all available data is used for training and testing, which can improve the accuracy of the evaluation.

Another advantage of 10-fold cross-validation is that it allows for a more comprehensive evaluation of the model's performance. By repeating the process 10 times, it provides a more robust estimate of the model's accuracy, precision, recall, F1 score, and other performance metrics.

Accuracy	Precision	Recall	F-measure	No. of features	Class
99.90	99.69	99.79	99.74	12	DoS
99.80	99.37	99.37	99.37	15	Probe
99.88	97.40	97.41	97.40	13	R2L
99.95	99.70	99.69	99.70	11	U2R

Table 2 Prediction and Evaluation results.

Overall, the IDS performs well with high accuracy and F-measure scores across all attack types. However, precision and recall scores are lower for R2L attacks. These results suggest that the IDS is effective in detecting different types of attacks and can be used to improve the security of the network.

2.5 SYSTEM CONFIGURATION

2.5.1 Software Requirements

- Operating system: Windows/MAC
- Python IDE: PyCharm/ VS Code/ Sublime Text
- Programming languages: Python
- Front-End framework: StreamLit
- Internet Explorer Versions: Chrome v-119, Microsoft Edge
- Network Monitoring API: Scapy, PCAP

2.5.2 Hardware Requirements

- CPU: Intel Core 2 Quad Processor Intel I5/I7
- RAM: 8 GB or greater
- Hard Drive Storage: SSD – 256 GB

3. LITERATURE OVERVIEW

3.1 FEASIBILITY STUDY

Feasibility studies are crucial in ensuring that a project is viable and worth pursuing. In the context of an IDS, a feasibility study should consider factors such as the availability of data, the performance of the system, and the potential benefits to the organization.

1. The paper by A. Arulmurugan and S. Santhosh Kumar titled “Feasibility Study of Intrusion Detection System Using Decision Tree Algorithm”, emphasizes the importance of data availability and quality in the feasibility study of an IDS. The authors argue that the success of an IDS depends largely on the quality of the data used to train the classification algorithm. They recommend that the feasibility study should include an assessment of the availability and quality of the data, as well as an analysis of the data pre-processing techniques that may be necessary to prepare the data for analysis.

In addition to data quality, the authors stress the need for accurate classification algorithms. They argue that the classification algorithm is the core component of an IDS, and that the accuracy of the algorithm will determine the effectiveness of the system. The authors recommend that multiple classification algorithms be tested during the feasibility study to identify the most accurate and reliable algorithm.

2. The paper by Zhang, Liu, and Xie titled “A Feasibility Study of Intrusion Detection Techniques for Wireless Sensor Networks”, focuses on the evaluation of IDS performance. The authors evaluate several intrusion detection techniques for wireless sensor networks, which are often used in the context of Internet of Things (IoT) applications. They use metrics such as detection rate, false positive rate, and detection delay to evaluate the effectiveness of each technique.

The authors also highlight the need for a comprehensive evaluation methodology. They argue that the evaluation methodology should take into account not only the performance of the detection algorithm, but also the practical considerations of deploying the IDS in a real-world environment. For example, the authors suggest that the feasibility study should evaluate the scalability of the IDS, as well as its ability to adapt to changing network conditions.

3.2 PROJECT LITERATURE

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy and company strength. Once these things are satisfied, ten next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration are taken into account for developing the proposed system.

3.2.1 Anomaly based Network Intrusion Detection using Machine Learning Techniques.

Authors: Tushar Rakshe, Vishal Gonjari

In the network communications, network intrusion is the most important concern nowadays. The booming contingency of network attacks is a devastating problem for network services. Various research works are already conducted to find an effective and efficient solution to prevent intrusion in the network in order to ensure network security and privacy. Machine learning is an effective analysis tool to detect any suspicious events occurred in the network traffic flow. In this paper, we developed a classifier model based on SVM and Random Forest based algorithms for network intrusion detection. The NSL-KDD dataset, a much improved version of the original KDDCUP’99 dataset, was used to evaluate the performance of our algorithm. The main task of our detection algorithm was to classify whether the incoming network traffics are normal or an attack, based on 41 features describing every pattern of network traffic. The detection accuracy more than 95 % was achieved using SVM and Random algorithms. The results of two algorithms compared and it is observed that Random Forest algorithm is more effective than Support Vector Machine.

3.2.2 A Subset Feature Elimination Mechanism for Intrusion Detection System

Authors: Herve Nkiama, Syed Zainudeen, Muhammad Saidu

Several studies have suggested that by selecting relevant features for intrusion detection system, it is possible to considerably improve the detection accuracy and performance of the detection engine. Nowadays with the emergence of new technologies such as Cloud Computing or Big Data, large amount of network traffic are generated and the intrusion detection system must dynamically collected and analyzed the data produce by the incoming traffic. However in a large dataset not all features contribute to represent the traffic, therefore reducing and selecting a number of adequate features may improve the speed and accuracy of the intrusion detection system. In this study, a feature selection mechanism has been proposed which aims to eliminate non-relevant features as well as identify the features which will contribute to improve the detection rate, based on the score each features have established during the selection process. To achieve that objective, a recursive feature elimination process was employed and associated with a decision tree based classifier and later on, the suitable relevant features were identified. This approach was applied on the NSL-KDD dataset which is an improved version of the previous KDD 1999 Dataset, scikit-learn that is a machine learning library written in python was used in this paper. Using this approach, relevant features were identified inside the dataset and the accuracy rate was improved. These results lend to support the idea that features selection improve significantly the classifier performance. Understanding the factors that help identify relevant features will allow the design of a better intrusion detection system.

3.2.3 Intrusion Detection System using AI and Machine Learning Algorithm

Authors: Syam Akhil Repalle, Venkata Ratnam Kolluru

Secure automated threat detection and prevention is the more effective procedure to reduce the workload of analyst by scanning the network, server functions & then informs the analyst if any suspicious activity is detected in the network traffic. It monitors the system continuously and responds according to the threat environment. This response action varies from phase to phase. Here suspicious activities are detected by the help of an artificial intelligence which acts as a virtual analyst concurrently with network intrusion detection system to defend from the threat environment and taking appropriate measures with the permission of the analyst. In its final phase where packet analysis is carried out to surf for attack vectors and then categorize supervised and unsupervised data. Where the unsupervised data will be decoded or converted to supervised data with help of analyst feedback and then auto-update the algorithm (Virtual Analyst Algorithm). So that it evolves the algorithm (with Active Learning Mechanism) itself by time and become more efficient, strong. So it can able to defend form similar or same kind of attacks.

3.2.4 A Deep Learning Approach to Network Intrusion Detection

Authors: Nathan Shone; Tran Nguyen Ngoc; Vu Dinh Phai; Qi Shi

Network intrusion detection systems (NIDSs) play a crucial role in defending computer networks. However, there are concerns regarding the feasibility and sustainability of current approaches when faced with the demands of modern networks. More specifically, these concerns relate to the increasing levels of required human interaction and the decreasing levels of detection accuracy. This paper presents a novel deep learning technique for intrusion detection, which addresses these concerns. We detail our proposed nonsymmetric deep autoencoder (NDAE) for unsupervised feature learning. Furthermore, we also propose our novel deep learning classification model constructed using stacked NDAEs. Our proposed classifier has been implemented in graphics processing unit (GPU)-enabled TensorFlow and evaluated using the benchmark KDD Cup '99 and NSL-KDD datasets. Promising results have been obtained from our model thus far, demonstrating improvements over existing approaches and the strong potential for use in modern NIDSs.

3.2.5 Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection

Authors: Sumouli Choudhury; Anirban Bhowal

Intrusion detection is one of the challenging problems encountered by the modern network security industry. A network has to be continuously monitored for detecting policy violation or suspicious traffic. So an intrusion detection system needs to be developed which can monitor network for any harmful activities and generate results to the management authority. Data mining can play a massive role in the development of a system which can detect network intrusion. Data mining is a technique through which important information can be extracted from huge data repositories. In order to spot intrusion, the traffic created in the network can be broadly categorized into following two categories- normal and anomalous. In our proposed paper, several classification techniques and machine learning algorithms have been considered to categorize the network traffic. Out of the classification techniques, we have found nine suitable classifiers like BayesNet, Logistic, IBK, J48, PART, JRip, Random Tree, Random Forest and REPTree. Out of the several machine learning algorithms, we have worked on Boosting, Bagging and Blending (Stacking) and compared their accuracies as well. The comparison of these algorithms has been performed using WEKA tool and listed below according to certain performance metrics. Simulation of these classification models has been performed using 10-fold cross validation. NSL-KDD based data set has been used for this simulation in WEKA.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

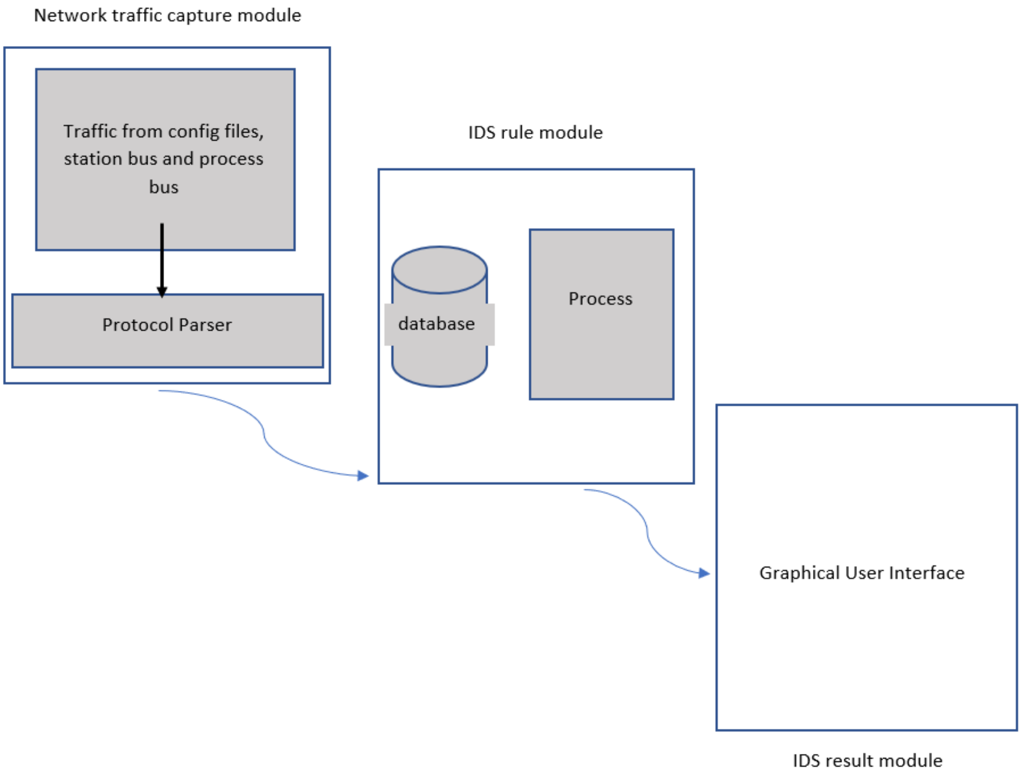


Fig 3.1.1 The architecture of the intrusion detection system

The architecture of the intrusion detection system using the decision tree algorithm with the NSL-KDD dataset and feature selection mechanism consists of three main modules: the network capture module, the IDS rule module, and the IDS result module.

Network Capture Module is responsible for capturing network traffic data from various sources such as routers, switches, and firewalls. This module uses a packet capture tool such as Wireshark to intercept and collect network traffic data. Once collected, the data is pre-processed and converted into a suitable format for further analysis. Pre-processing steps may include data cleaning, filtering, and normalization to ensure that the data is consistent and ready for analysis.

IDS Rule Module processes the pre-processed data and applies a set of predefined rules to identify potential network intrusions. The rules are based on known attack patterns and are designed to detect abnormal traffic patterns that may indicate an attack. In our system, we use a decision tree-based classifier to identify potential threats based on the selected features. The classifier is trained on a labelled dataset and then applied to new data to predict the presence or absence of an attack. The rules are continuously updated and refined to improve the accuracy and reliability of the system.

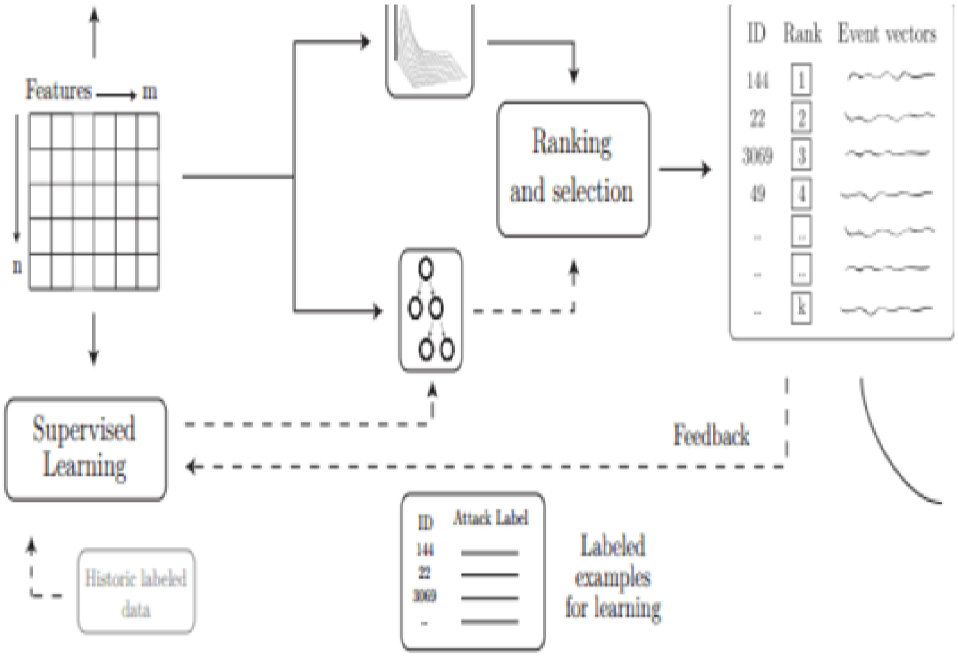


Fig 3.1.2 The process of the IDS rule module

IDS Result Module receives the output of the IDS Rule Module and generates a report on the detected threats. The report contains information such as the type of attack, the severity of the threat, and recommended actions to mitigate the risk. This module also includes a visualization component to help users understand the detected threats and make informed decisions about how to respond to them. The visualization may include graphs, charts, and other graphical representations of the data to facilitate understanding and interpretation of the results.

Overall, the three main modules work together to provide a comprehensive intrusion detection system that captures, analyzes, and reports on potential network intrusions. The Network Capture Module captures the data, the IDS Rule Module analyzes the data to detect potential threats, and the IDS Result Module generates a report on the detected threats. Together, these modules provide a powerful tool for identifying and mitigating the risk of network intrusions.

4.2 UML DIAGRAMS

4.2.1 Data Flow Diagram

Level 0 Data Flow Diagram

A level 0 data flow diagram (DFD) of an Intrusion Detection System (IDS) typically shows the overall flow of information between different entities within the system. At this level, the DFD provides a high-level view of the system and does not include detailed information about individual processes.

In a level 0 DFD of an IDS, there are four main entities: the IDS itself, the user, the admin, and the decision tree algorithm. The IDS is the central component of the system and is responsible for monitoring network traffic and detecting potential security threats. The user and admin are external entities who interact with the system and receive reports about potential security threats. The decision tree algorithm is an internal component of the IDS that analyzes network traffic and determines whether a potential threat is present. The level 0 DFD typically shows the flow of information between these entities in a simple diagram. For example, the diagram may show arrows representing the flow of data between the IDS and the decision tree algorithm, as well as between the IDS and the user/admin. The user/admin may receive reports from the IDS indicating that a potential security threat has been detected, and may then take appropriate action to mitigate the threat.

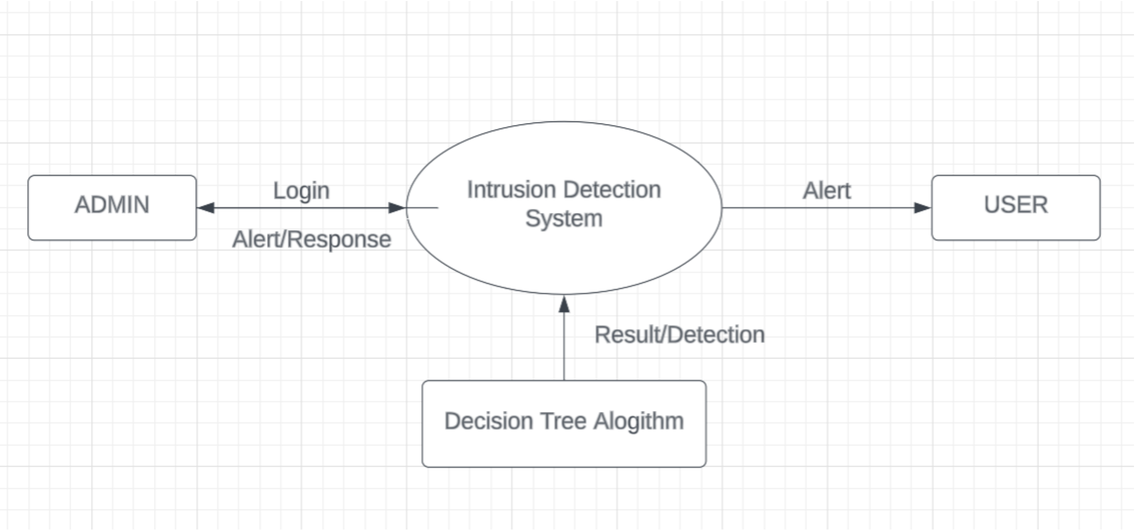


Fig 3.2.1 Level 0 DFD of Intrusion Detection System

Level 1 Data Flow Diagram

At level 1, the Data Flow Diagram (DFD) of the Intrusion Detection System (IDS) expands on the level 0 diagram and provides a more detailed view of the system's processes and data flow. External entities in this context are the intrusion detection system and the user. The admin is the main entity responsible for managing the system and its components, including the decision tree algorithm. The following are the main processes of the IDS at level 1:

1. Login The login process allows the administrator to access the system by verifying their credentials. If the credentials are valid, the administrator gains access to the system and can perform various tasks.

2. Refine Dataset The refine dataset process involves the pre-processing of network traffic data before it is used to train the machine learning model. This includes activities such as data cleaning, feature extraction, and feature selection. The goal is to ensure that the data used to train the model is accurate, relevant, and representative of real-world traffic patterns.

3. Maintain IDS The maintain IDS process involves the monitoring and management of the Intrusion Detection System itself. This includes activities such as updating software, configuring sensors, fine-tuning detection rules, and monitoring system performance. The goal is to ensure that the IDS remains effective and up-to-date in detecting new and evolving threats.

4. Get Result The get result process involves requesting and receiving the results of an intrusion detection scan. The administrator can specify the type of scan (e.g., a real-time or batch scan) and the criteria for the scan (e.g., a specific time range or type of attack). The IDS then performs the scan and generates a report of the results, which can be viewed by the administrator.

5. View Result The view result process involves accessing and interpreting the results of an intrusion detection scan. The results are typically presented in a graphical or tabular format, which allows the administrator to quickly identify any security threats or anomalies. The administrator can then take appropriate action to mitigate the threat or investigate the cause of the anomaly.

The following are the data stores in the IDS at level 1:

1. Network Dataset The network dataset database stores the raw network traffic data that is used to train the machine learning model. The data includes information about network packets, such as source and destination IP addresses, port numbers, packet size, and protocol type. The database is periodically updated with new traffic data to ensure that the model remains accurate and up-to-date.

2. Attack Reports The attack reports database stores information about detected security threats and attacks. The data includes details about the type of attack, the time and location of the attack, and any relevant network traffic data. The database is used to track and analyze security threats over time, and can also be used to generate reports and alerts for administrators.

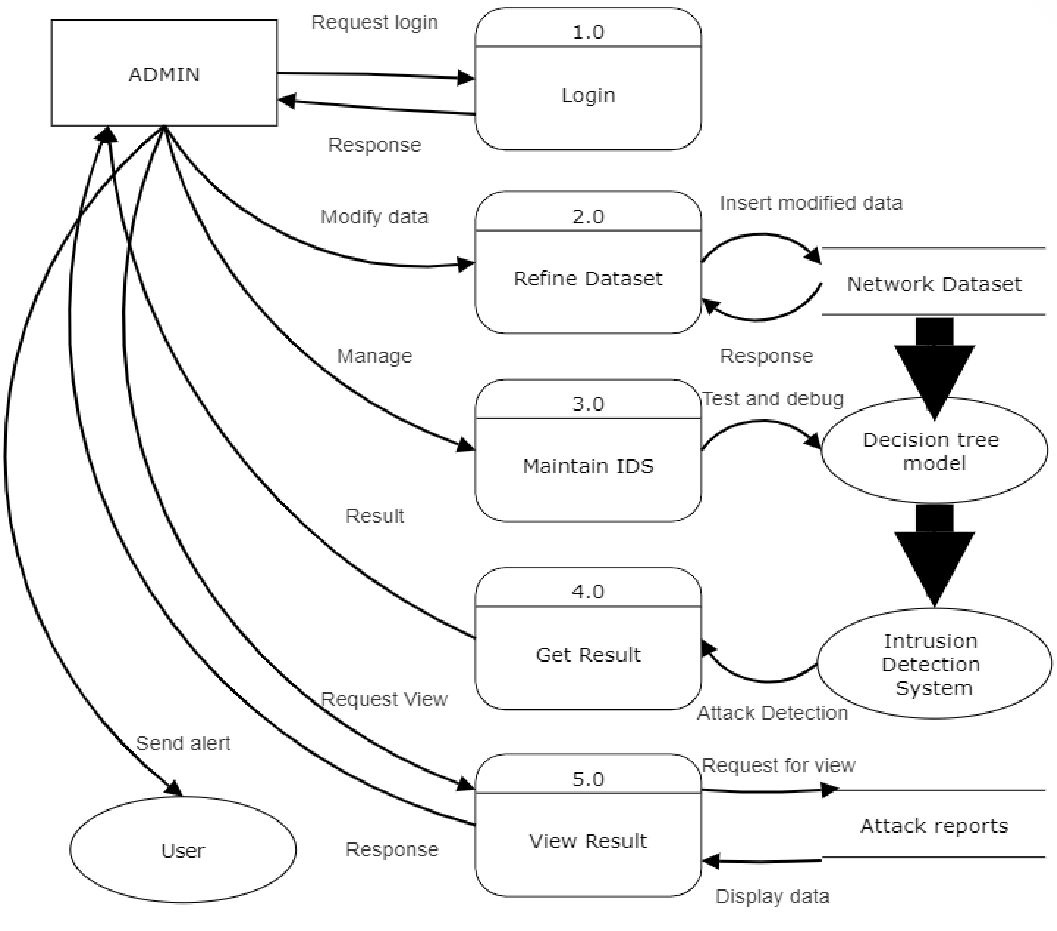


Fig 3.2.2 Level 1 DFD of Intrusion Detection System

4.2.2 Use Case Diagram

The actions in the Use Case Diagram related to the IDS is :

Load Dataset The user and the admin can load a dataset into the IDS or the admin can directly integrate the Network API that extracts the network data and later this data can be used for analysis and training of the decision tree algorithm.

Apply Decision Tree Algorithm The user can apply a decision tree algorithm to the loaded dataset for the purpose of identifying patterns of malicious activity.

Intrusion Detection The user can initiate the IDS to perform intrusion detection by analyzing network traffic or system logs, comparing them to the patterns identified by the decision tree algorithm.

Alert The IDS can generate an alert if suspicious activity is detected, which can be sent to the user or admin for further investigation.

Refine Dataset The user/admin can refine the dataset by removing false positives or adding new data to improve the accuracy of the decision tree algorithm.

Maintain IDS The admin can maintain the IDS by configuring and updating its settings, monitoring its performance, and ensuring its overall health and availability.

The actors in the use case diagram are the IDS (Intrusion Detection System), User and Admin. The roles of each of these actors are:

1. IDS The Intrusion Detection System (IDS) is the primary actor in this use case diagram. It is responsible for performing intrusion detection by analyzing network traffic or system logs for suspicious activity. The IDS also includes the decision tree algorithm that is used to identify patterns of malicious activity and generate alerts which the user initiates.

2. User The user is another actor in the use case diagram. The user interacts with the IDS by loading datasets into the system, initiating intrusion detection, receiving alerts from Intrusion Detection System (IDS).

3. Admin The admin is the third actor in the use case diagram. The admin is responsible for refining datasets to improve the accuracy of the decision tree algorithm maintaining the IDS and others actions includes configuring and updating its settings, monitoring its performance, and ensuring its overall health and availability. The admin may also receive alerts generated by the IDS and take appropriate actions in response to them.

The IDS is the central actor in the use case diagram, while the user and admin are secondary actors who interact with the IDS in various ways to perform their respective roles.

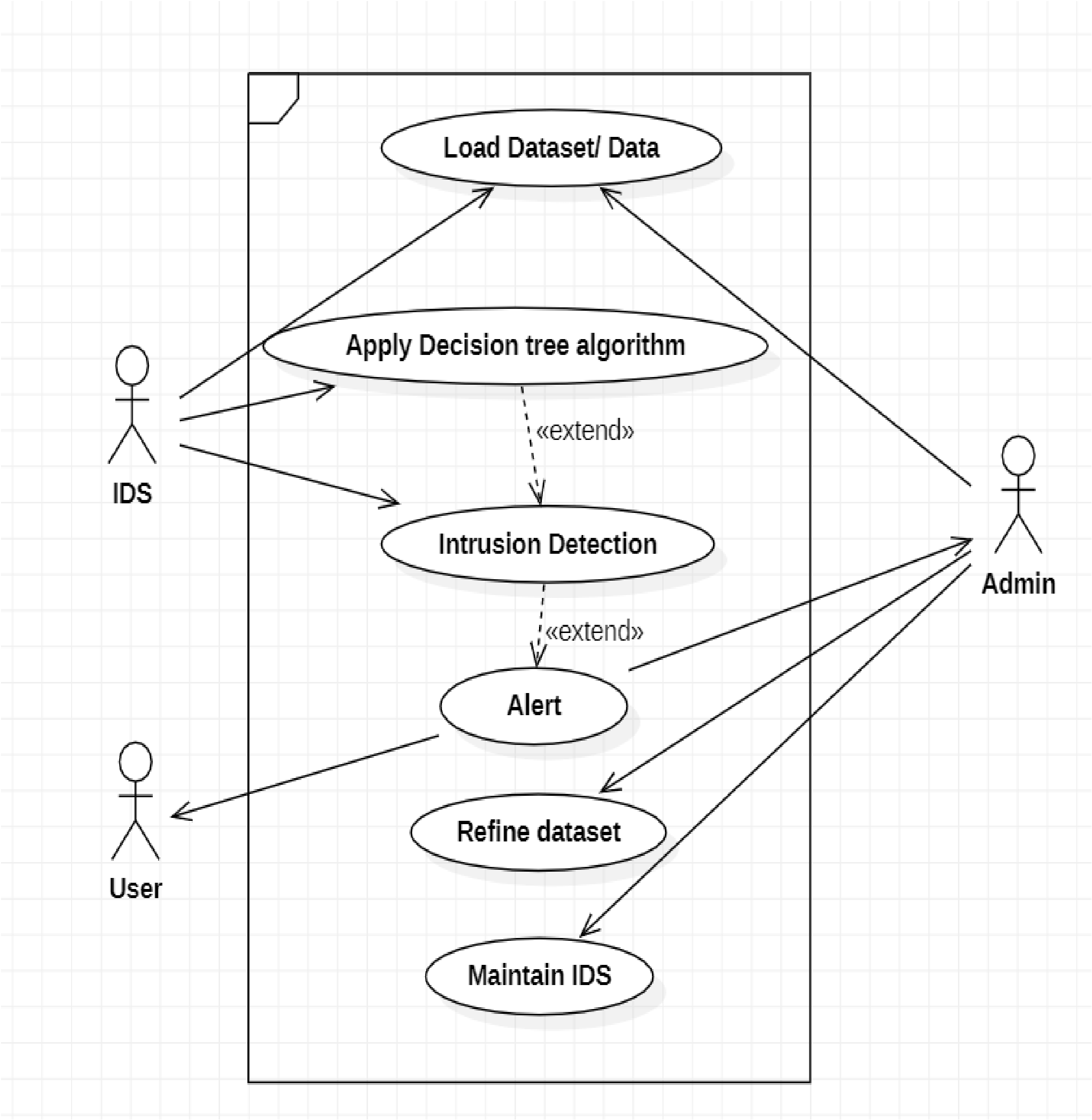


Fig 3.2.3 Use Case Diagram of Intrusion Detection System

4.2.3 Class Diagram

The class diagram for the IDS with ML algorithms includes five main classes: Networking Monitoring System, Intrusion Detection System, Decision Tree Algorithm, User, and Administrator. Each of these classes has different attributes and methods that are used in the intrusion detection process.

1. Networking Monitoring System Class

The Networking Monitoring System class is responsible for monitoring network traffic and sending it to the Intrusion Detection System. It has the following attributes and methods:

Network ID: This attribute represents the unique identifier for the monitored network.

Monitor Network(): This method is used to monitor network traffic and send it to the Intrusion Detection System.

2. Intrusion Detection System Class

The Intrusion Detection System class is responsible for detecting and classifying attacks. It has the following attributes and methods:

Network ID: This attribute represents the unique identifier for the monitored network.

Intrusion Detection(): This method is used to monitor network traffic in real-time and compare it to the decision tree model to determine if any anomalous activity is occurring.

Issue Alert(): This method is used to send an alert to the administrator if the IDS detects any anomalous activity.

3. Decision Tree Algorithm Class

The Decision Tree Algorithm class is responsible for building a decision tree model from the input dataset. It has the following attributes and methods:

detect and classify attack(): This method is used to apply the decision tree algorithm to the input dataset, building a decision tree model that can classify network traffic as normal or anomalous.

4. User Class

The User class represents the users of the network that is being monitored by the IDS. It has the following attributes and methods:

Network ID: This attribute represents the unique identifier for the monitored network.

Get Alert(): This method is used to retrieve any alerts that have been issued by the IDS.

Request Service(): This attribute represents the service request made by the IDS if an intrusion is detected.

5. Administrator Class

The Administrator class represents the administrator of the IDS. It has the following attributes and methods:

Refine Dataset(): This method is used to refine the input dataset by removing irrelevant data and selecting the features that will be used in the intrusion detection process.

Maintain IDS(): This method is used to perform regular maintenance on the IDS to ensure that it is operating correctly.

Network Provider ID: This attribute represents the unique identifier for the network provider.

Get Alert(): This method is used to retrieve any alerts that have been issued by the IDS.

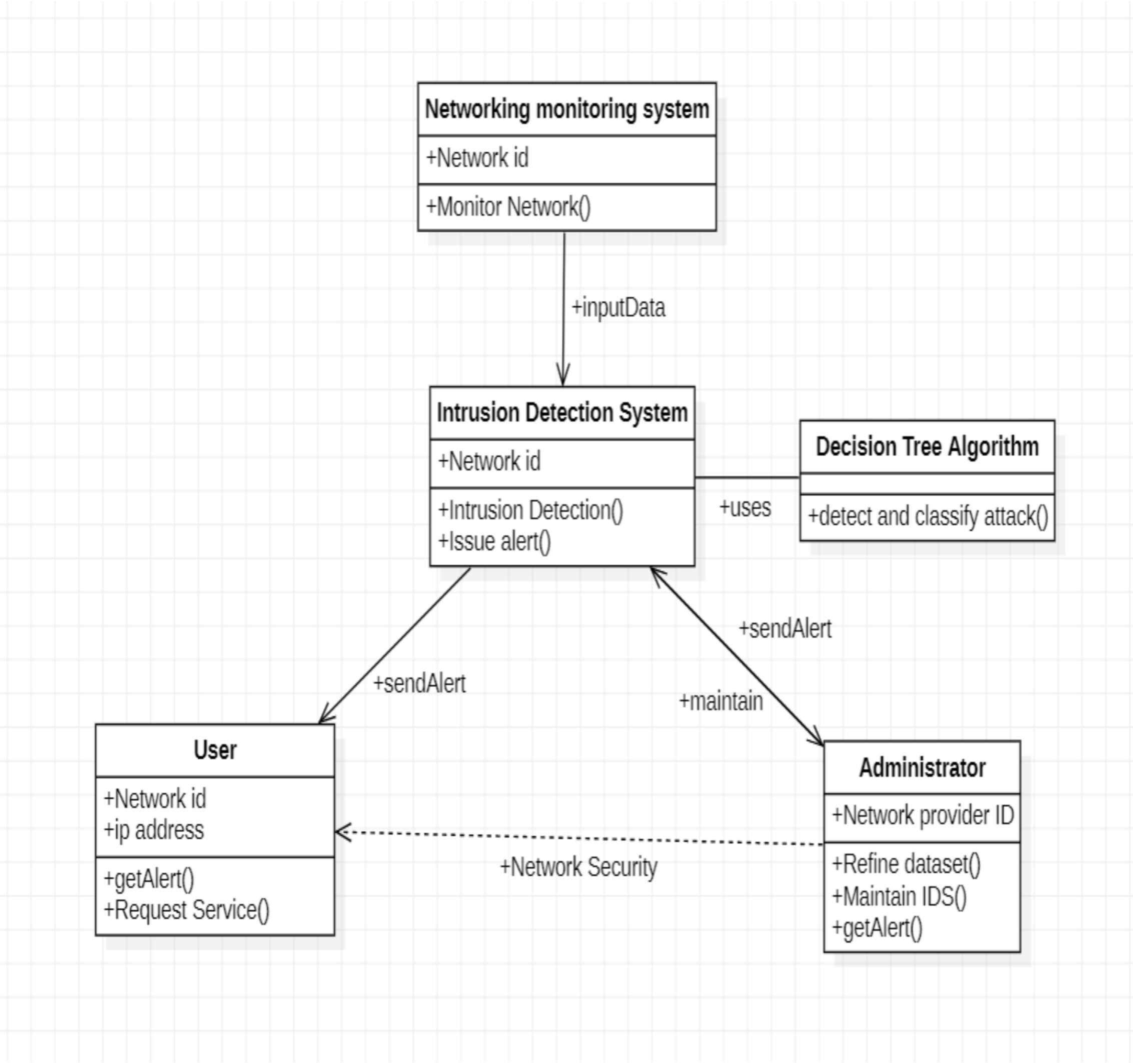


Fig 3.2.4 Class Diagram of Intrusion Detection System

4.2.4 Sequence Diagram

The sequence diagram with events get data from user network, pass network data from user network, detect the intrusion, request service, send intrusion if detected, send alert if intrusion detected, checks functionality status, reports functionality status. with the interaction frames user, network monitoring system, intrusion detection system and administrator. A more detailed explanation of each step in the sequence diagram:

1. The User sends data from their network to the Network Monitoring System (NMS)

This step represents the User's actions in sending network traffic data to the NMS for analysis. The User is not shown as an explicit actor in the subsequent steps of the diagram, but their actions are still a critical part of the overall process.

2. The NMS passes the data to the Intrusion Detection System (IDS)

This step represents the NMS's role in passing the network traffic data to the IDS for analysis. The NMS is responsible for monitoring the network traffic and identifying suspicious activity, but it relies on the IDS for more advanced intrusion detection capabilities.

3. The IDS detects an intrusion and requests service

This step represents the IDS's role in detecting an intrusion based on the network traffic data received from the NMS. When the IDS detects an intrusion, it sends a message requesting service to the Administrator. The nature of the service requested may vary depending on the specific implementation of the IDS, but it could involve additional analysis or response actions.

4. The IDS sends an intrusion message to the Administrator

This step represents the IDS sending a message to the Administrator notifying them of the detected intrusion. The message may include details about the nature of the intrusion, such as the source IP address or the type of attack detected.

5. The IDS sends an alert message to the NMS

This step represents the IDS sending a message to the NMS notifying it of the detected intrusion. The NMS may use this information to adjust its monitoring and analysis parameters, or to take other actions to mitigate the effects of the intrusion.

6. The Administrator checks the functionality status of the IDS

This step represents the Administrator's role in monitoring the health and performance of the IDS. The Administrator may use various tools and metrics to assess the current status of the IDS and identify any issues that need to be addressed.

7. The Administrator reports the functionality status to the NMS

This step represents the Administrator sending a message to the NMS reporting the current status of the IDS. The message may include details about any issues or concerns that were identified during the status check, as well as any recommended actions to address them.

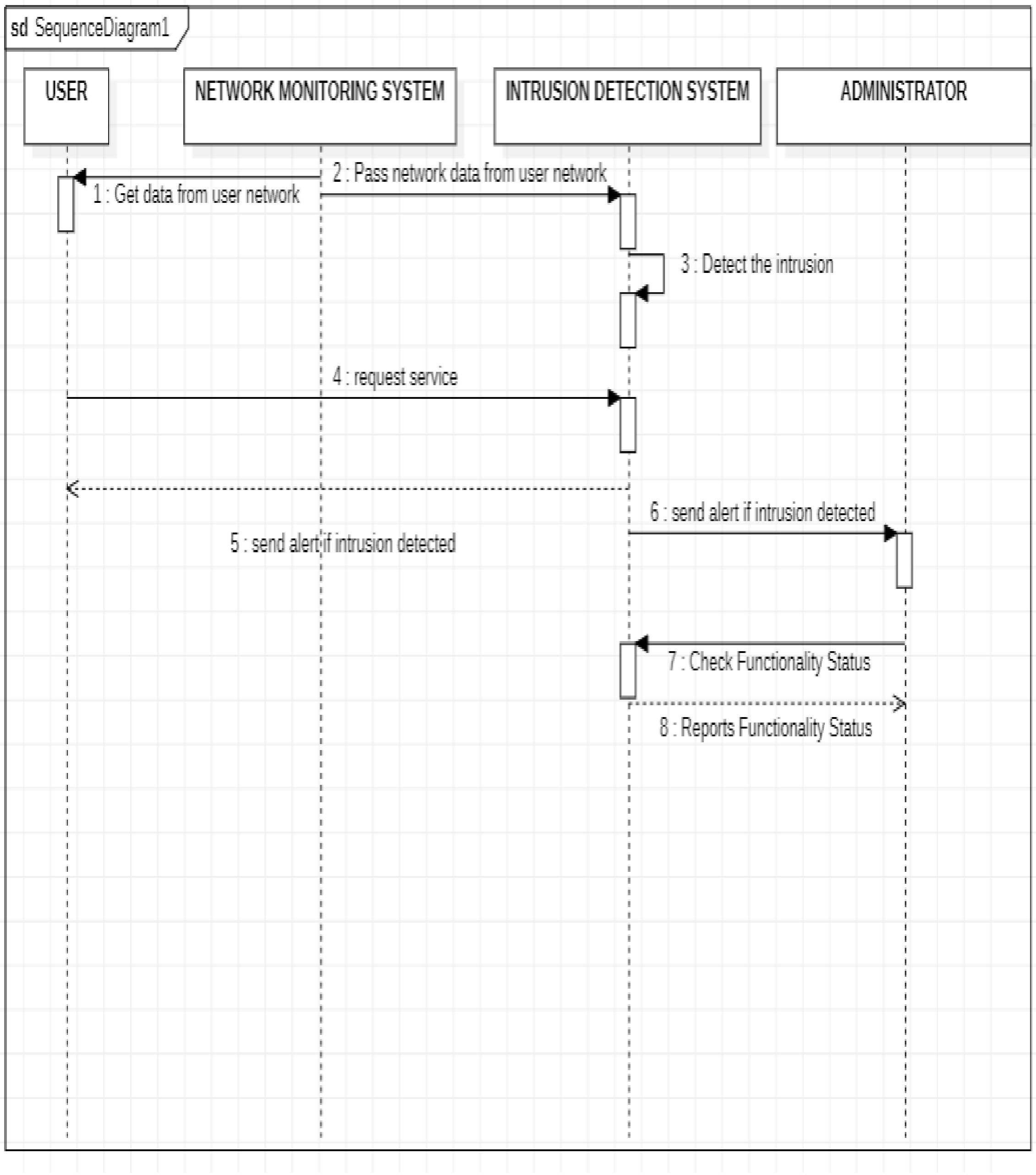

sd SequenceDiagram1
USER
NETWORK MONITORING SYSTEM
INTRUSION DETECTION SYSTEM
ADMINISTRATOR
2 : Pass network data from user network
1 : Get data from user network
3 : Detect the intrusion
4 : request service
6 : send alert if intrusion detected
5 : send alert if intrusion detected
7 : Check Functionality Status
8 : Reports Functionality Status

Fig 3.2.5 Sequence Diagram of Intrusion Detection System

4.2.5 Activity Diagram

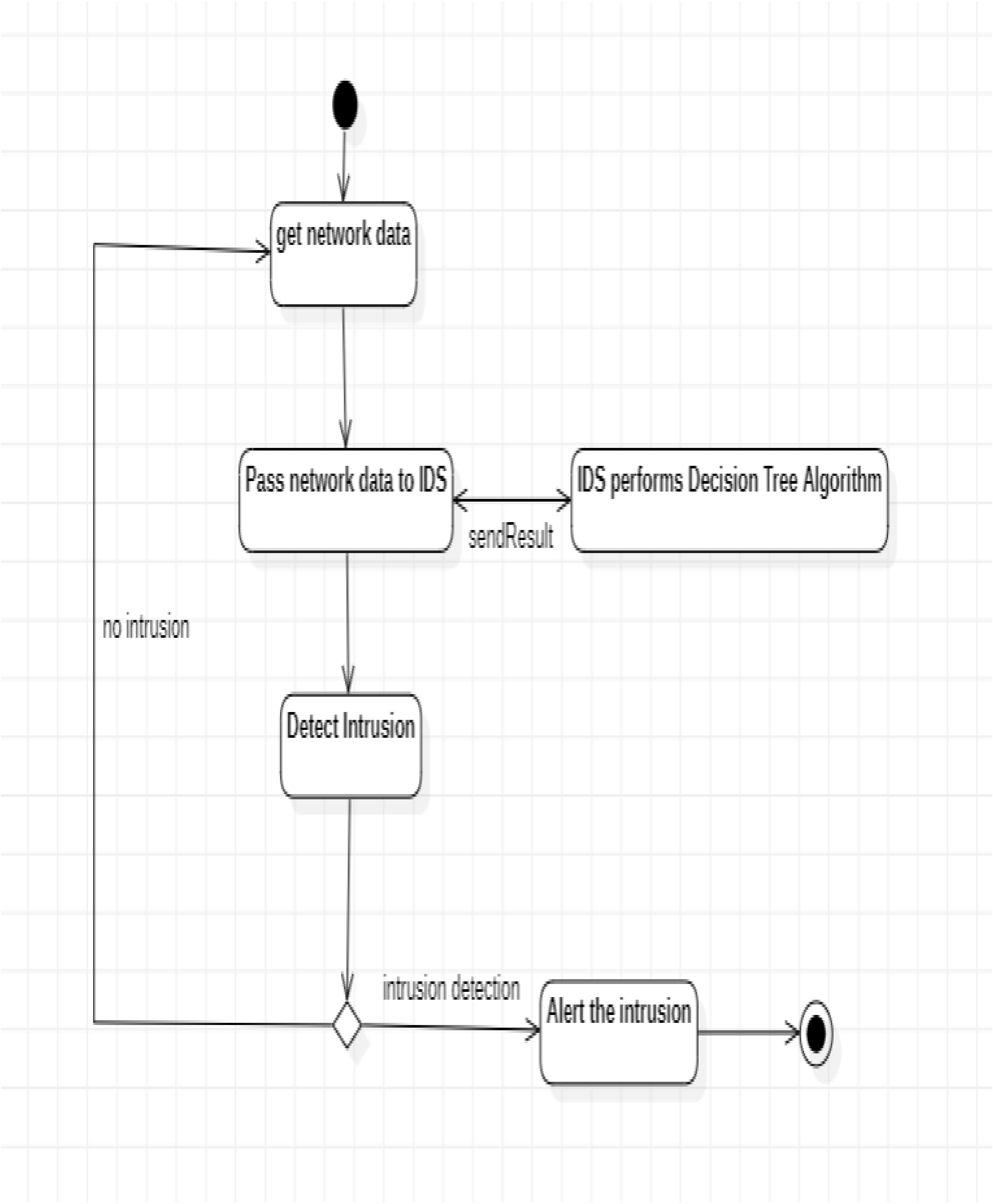


Fig 3.2.6 Activity Diagram of Intrusion Detection System

The activity diagram can provide a high-level overview of the steps involved in building and deploying an ML-based IDS, and can help to identify potential bottlenecks or areas for optimization. It can also be used as a communication tool to help stakeholders understand the overall workflow and the role of different components in the system. It also helps to adjusting parameters, choosing algorithms, or selecting features.

A more detailed description of the activity diagram for the intrusion detection system:

1. Start

The activity diagram begins with the Start activity, indicating the start of the system.

2. Get Network Data

The first activity of the system is to load the dataset into the IDS. This involves importing a dataset of network traffic data, which will be used as input for the IDS. Refine Dataset, After loading the dataset, the next step is to refine it by removing irrelevant data and selecting the features that will be used in the intrusion detection process.

3. Apply Decision Tree Algorithm

Once the dataset is refined, the IDS applies a decision tree algorithm to the dataset. The decision tree algorithm is a machine learning algorithm that builds a decision tree model from the input dataset. The decision tree model is used to classify network traffic as either normal or anomalous.

4. Detect Intrusion

With the decision tree model in place, the IDS begins the process of detecting intrusion. The IDS monitors network traffic in real-time and compares it to the decision tree model to determine if any anomalous activity is occurring.

5. Send Alert if Intrusion Detected

If the IDS detects any anomalous activity, it sends an alert to the administrator. The alert contains information about the type of intrusion detected, the location of the intrusion, and the severity of the intrusion.

6. End

The activity diagram ends with the End activity, indicating the end of the system.

4.2.6 State Diagram

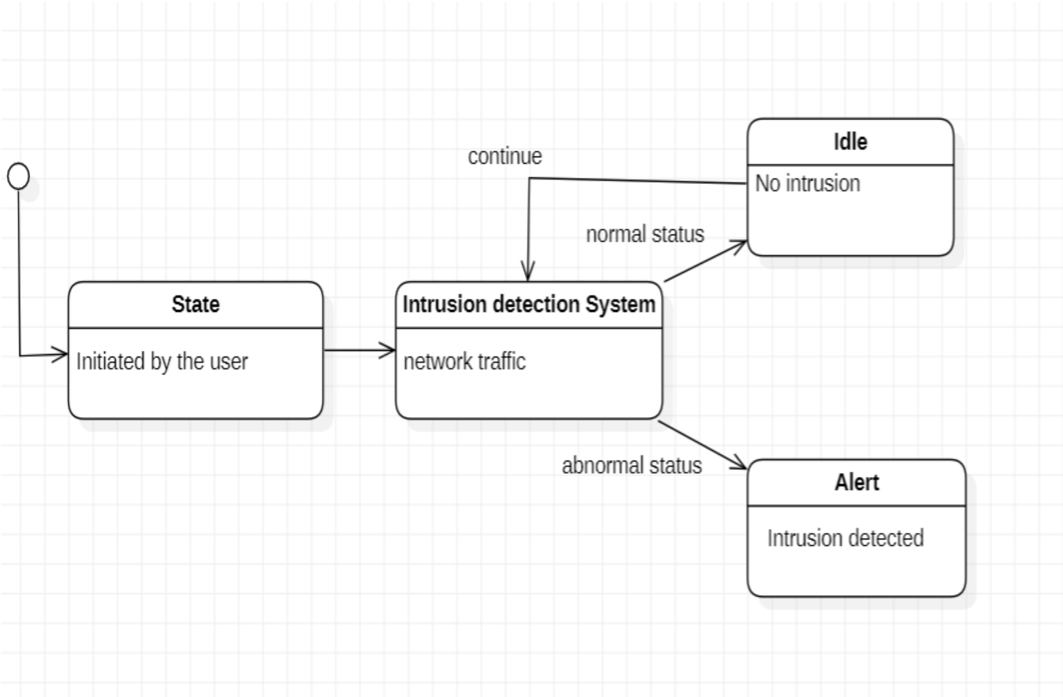


Fig 3.2.7 State Diagram of Intrusion Detection System

A brief explanation of each state and the transitions between them:

1.Initial State

This is the starting state of the system. The system is waiting to receive network traffic data.

2.Network Traffic

When network traffic data is received by the NMS, the system transitions to this state.

3.Idle State/No Intrusion

If the IDS does not detect any intrusion, the system transitions to this state. In this state, the system is waiting to receive more network traffic data.

4.Intrusion Detected

If the IDS detects an intrusion, the system transitions to this state.

5.Alert State/Intrusion Detected

In this state, an alert is sent to the Administrator and the NMS. The system waits for a response from the Administrator.

6.No Intrusion

If the Administrator determines that no intrusion occurred, the system transitions back to the Idle State/No Intrusion state.

4.3 SYSTEM DESIGN

1. System Requirements The ML-based IDS project requires a system with a minimum of 8 GB of RAM and a 64-bit processor. The system should have Python 3.x installed along with required libraries such as Scikit-Learn, Scapy, Streamlit, and Numpy.

2. Operating Environment The ML-based IDS project can be run on any operating system that supports Python, such as Windows, Linux, and macOS.

3. System and Subsystem Architecture The ML-based IDS project has three main subsystems: Network Capture Module, IDS Rule Module, and IDS Result Module. The Network Capture Module captures network traffic data using a packet capture tool such as Wireshark. The IDS Rule Module processes the captured data and applies a set of predefined rules to identify potential network intrusions using a decision tree-based classifier. The IDS Result Module generates a report on the detected threats and includes a visualization component to help users understand the detected threats.

4. Files and Database Design The ML-based IDS project does not require any external database. However, it uses the NSL-KDD dataset, which is available in a CSV file format.

5. Input Formats The ML-based IDS project takes network traffic data as input, which is collected by the Network Capture Module. The data is preprocessed and converted into a suitable format for analysis.

6. Output Layouts The IDS Result Module generates a report on the detected threats, which includes information such as the type of attack, the severity of the threat, and recommended actions to mitigate the risk. The report also includes a visualization component, which may include graphs, charts, and other graphical representations of the data.

7. Human-Machine Interfaces The ML-based IDS project has a user interface developed using Streamlit. The user interface allows users to interact with the system and view the results of the analysis. The user interface includes options to start and stop the IDS, view the captured data, and access the IDS report. The interface also includes visualization components to help users understand the detected threats.

5. SAMPLE CODE

5.1 CODING

```
from scapy.all import rdpcap
from scapy.utils import wrpcap
import csv
import streamlit as st
from scapy.sendrecv import sniff
import random
import smtplib
from email.message import EmailMessage
import ssl
import pandas as pd
import numpy as np
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
email_sender = 'vnxx1910@gmail.com'
email_password = 'wumankcpkeoyluxs'
email_receiver = 'vudhanthineeraja@gmail.com'
subject = 'NETWORK ANALYSIS REPORT'
body="""
Your Source MAC Address is: c2:b6:58:38:52:64
Your Destination MAC Address is: 4c:79:6e:5c:4c:b6
Please find the Time and Attack Type report attached below!
Contact administrator at +91-xxxxxxxxxx
or
Email us on: vnxx1910@gmail.com
"""
em = EmailMessage()
em['From'] = email_sender
em['To'] = email_receiver
em['Subject'] = subject
```

```
em.set_content(body)
# Attach the CSV file
with open('times.csv', 'rb') as f:
file_data = f.read()
em.add_attachment(file_data, maintype='text', subtype='csv', filename='file.csv')
context = ssl.create_default_context()
def send_Email():
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context)  as smtp:
smtp.login(email_sender,email_password)
smtp.sendmail(email_sender,email_receiver,em.as_string())
st.write("Report Sent to mail!")
# with open('times.csv', 'w') as f:
# f.truncate(0)
st.title("NETWORK ANALYSIS")
st.write("This captures packets on the Wi-Fi interface and displays the network
details.")
data = pd.DataFrame({
'Column Name': ['type','proto'],
'Value1':['2,048=IPv4','6=TCP'],
 'Value2':['34525=IPv6','17=UDP']
})
st.dataframe(data,width=1000)
capture = sniff(iface='Wi-Fi',count = 100)
wrpcap("GFG3.pcap",capture)
fields = ['src', 'dst']
# , 'sport', 'dport','type','proto','flags']
packets = rdpcap(r'D:\Python Projects\ml ids\GFG3.pcap')
with open('D:\Python Projects\ml ids\data.csv', mode='w', newline='') as csv_file:
writer = csv.writer(csv_file)
# Write the header row with the field names
writer.writerow(fields)
# Write each packet's fields to a new row in the CSV file
```

```
for packet in packets:
row = [packet.getfieldval(field) for field in fields]
writer.writerow(row)
df = pd.read_csv('data.csv')
ip_df = df[['src', 'dst']]
# ,'sport','dport','type','proto','flags']]
st.dataframe(ip_df,width = 1000)
# attach the column names to the dataset
col_names = ["duration","protocol_type","service","flag","src_bytes",
"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
"logged_in","num_compromised","root_shell","su_attempted","num_root",
"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
"is_host_login","is_guest_login","count","srv_count","serror_rate",
"srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]
# KDDTrain+_2.csv & KDDTest+_2.csv are the datafiles without the last column
about the difficulty score
# these have already been removed.
df = pd.read_csv("KDDTrain+_2.csv", header=None, names = col_names)
df_test = pd.read_csv("KDDTest+_2.csv", header=None, names = col_names)
# shape, this gives the dimensions of the dataset
#print('Dimensions of the Training set:',df.shape) #(125973, 42)
#print('Dimensions of the Test set:',df_test.shape) #(22544, 42)
# first five rows
#print(df.head(5))
#print(df.describe())
# print('Label distribution Training set:')
# print(df['label'].value_counts())
# print()
# print('Label distribution Test set:')
# print(df_test['label'].value_counts())
# colums that are categorical and not binary yet: protocol_type (column 2), service
(column 3), flag (column 4).
# explore categorical features
#print('Training set:')
for col_name in df.columns:
if df[col_name].dtypes == 'object' :
unique_cat = len(df[col_name].unique())
#print("Feature '{col_name}' has {unique_cat}
categories".format(col_name=col_name, unique_cat=unique_cat))
#see how distributed the feature service is, it is evenly distributed and therefore we
need to make dummies for all.
# print()
# print('Distribution of categories in service:')
# print(df['service'].value_counts().sort_values(ascending=False).head())
# Test set
# print('Test set:')
for col_name in df_test.columns:
if df_test[col_name].dtypes == 'object' :
unique_cat = len(df_test[col_name].unique())
# print("Feature '{col_name}' has {unique_cat}
categories".format(col_name=col_name, unique_cat=unique_cat))
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
categorical_columns=['protocol_type', 'service', 'flag']
# insert code to get a list of categorical columns into a variable,
categorical_columns
categorical_columns=['protocol_type', 'service', 'flag']
 # Get the categorical values into a 2D numpy array
df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

```
# protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
unique_protocol2=[string1 + x for x in unique_protocol]
# service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]
# flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2
# print(dumcols)
# print(len(dumcols)) #84
#do same for test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_flag2
# print(testdumcols)
# print(len(testdumcols)) #78
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
# print(df_categorical_values_enc.head())
# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)
enc = OneHotEncoder()
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)
```

```
# test set
testdf_categorical_values_encenc =
enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data =
pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)
# df_cat_data.head()
trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
# difference
for col in difference:
testdf_cat_data[col] = 0
# testdf_cat_data.shape
newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)
# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)
# print(newdf.shape)
# print(newdf_test.shape)
# take label column
labeldf=newdf['label']
labeldf_test=newdf_test['label']
# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1, 'pod': 1,
'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1,
'worm': 1,'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' : 2,'saint' :
2,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient':
3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpguess': 3,'xlock':
3,'xsnoop': 3,'httptunnel': 3,'buffer_overflow': 4,'loadmodule': 4,'perl': 4,'rootkit':
4,'ps': 4,'sqlattack': 4,'xterm': 4})
newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' : 1 ,'back': 1, 'land': 1,
'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbomb': 1, 'apache2': 1, 'processtable': 1,
'udpstorm': 1, 'worm': 1, 'ipsweep' : 2,'nmap' : 2,'portsweep' : 2,'satan' : 2,'mscan' :
2,'saint' : 2,'ftp_write': 3,'guess_passwd': 3,'imap': 3,'multihop': 3,'phf': 3,'spy':
3,'warezclient': 3,'warezmaster': 3,'sendmail': 3,'named': 3,'snmpgetattack':
3,'snmpguess': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3, 'buffer_overflow':
4,'loadmodule': 4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})
# put the new label column back
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test
# print(newdf['label'].head())
to_drop_DoS = [2,3,4]
to_drop_Probe = [1,3,4]
to_drop_R2L = [1,2,4]
to_drop_U2R = [1,2,3]
DoS_df=newdf[~newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[~newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[~newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[~newdf['label'].isin(to_drop_U2R)];
# print('dimensions;',len(U2R_df),' ',U2R_df.shape)
# print(U2R_df.head(5))
#test
DoS_df_test=newdf_test[~newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[~newdf_test['label'].isin(to_drop_Probe)];
R2L_df_test=newdf_test[~newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[~newdf_test['label'].isin(to_drop_U2R)];
# print('Train:')
# print('Dimensions of DoS:' ,DoS_df.shape)
# print('Dimensions of Probe:' ,Probe_df.shape)
# print('Dimensions of R2L:' ,R2L_df.shape)
# print('Dimensions of U2R:' ,U2R_df.shape)
# print('Test:')
# print('Dimensions of DoS:' ,DoS_df_test.shape)
# print('Dimensions of Probe:' ,Probe_df_test.shape)
# print('Dimensions of R2L:' ,R2L_df_test.shape)
# print('Dimensions of U2R:' ,U2R_df_test.shape)
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outcome variables
X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label
X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label
X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label
X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label
# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label
X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label
X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label
X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
from sklearn import preprocessing
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
```

```
X_DoS=scaler1.transform(X_DoS)
scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)
scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)
# test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)
scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)
scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)
scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)
# print(X_DoS.std(axis=0))
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
#univariate feature selection with ANOVA F-test. using secondPercentile method, then RFE
#Scikit-learn exposes feature selection routines as objects that implement the transform method
#SelectPercentile: removes all but a user-specified highest scoring percentage of features
#f_classif: ANOVA F-value between label/feature for classification tasks.
from sklearn.feature_selection import SelectPercentile, f_classif
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
# print(X_newDoS.shape)
```

```
true=selector.get_support()
newcolindex_DoS=[i for i, x in enumerate(true) if x]
newcolname_DoS=list( colNames[i] for i in newcolindex_DoS )
# print(newcolname_DoS)
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
# print(X_newProbe.shape)
true=selector.get_support()
newcolindex_Probe=[i for i, x in enumerate(true) if x]
newcolname_Probe=list( colNames[i] for i in newcolindex_Probe )
# print(newcolname_Probe)
X_newR2L = selector.fit_transform(X_R2L,Y_R2L)
# X_newR2L.shape
true=selector.get_support()
newcolindex_R2L=[i for i, x in enumerate(true) if x]
newcolname_R2L=list( colNames[i] for i in newcolindex_R2L)
# print(newcolname_R2L)
X_newU2R = selector.fit_transform(X_U2R,Y_U2R)
# X_newU2R.shape
true=selector.get_support()
newcolindex_U2R=[i for i, x in enumerate(true) if x]
newcolname_U2R=list( colNames[i] for i in newcolindex_U2R)
# print(newcolname_U2R)
# print('Features selected for DoS:',newcolname_DoS)
# print()
# print('Features selected for Probe:',newcolname_Probe)
# print()
# print('Features selected for R2L:',newcolname_R2L)
# print()
# print('Features selected for U2R1:',newcolname_U2R)
clf = DecisionTreeClassifier(random_state=0)
rfe = RFE(clf, n_features_to_select=1)
rfe.fit(X_newDoS, Y_DoS)
```

```
X_rfeDoS=rfe.transform(X_newDoS)
rfe.fit(X_newProbe, Y_Probe)
X_rfeProbe=rfe.transform(X_newProbe)
rfe.fit(X_newR2L, Y_R2L)
X_rfeR2L=rfe.transform(X_newR2L)
rfe.fit(X_newU2R, Y_U2R)
X_rfeU2R=rfe.transform(X_newU2R)
clf_DoS=DecisionTreeClassifier(random_state=0)
clf_Probe=DecisionTreeClassifier(random_state=0)
clf_R2L=DecisionTreeClassifier(random_state=0)
clf_U2R=DecisionTreeClassifier(random_state=0)
clf_DoS.fit(X_DoS, Y_DoS)
clf_Probe.fit(X_Probe, Y_Probe)
clf_R2L.fit(X_R2L, Y_R2L)
clf_U2R.fit(X_U2R, Y_U2R)
# clf_rfeDoS=DecisionTreeClassifier(random_state=0)
# clf_rfeProbe=DecisionTreeClassifier(random_state=0)
# clf_rfeR2L=DecisionTreeClassifier(random_state=0)
# clf_rfeU2R=DecisionTreeClassifier(random_state=0)
# clf_rfeDoS.fit(X_rfeDoS, Y_DoS)
# clf_rfeProbe.fit(X_rfeProbe, Y_Probe)
# clf_rfeR2L.fit(X_rfeR2L, Y_R2L)
# clf_rfeU2R.fit(X_rfeU2R, Y_U2R)
# print(clf_DoS.predict(X_DoS_test))
# print(clf_Probe.predict(X_Probe_test))
# print(clf_R2L.predict(X_R2L_test))
# print(clf_U2R.predict(X_U2R_test))
clf_Dos_list = list(clf_DoS.predict(X_DoS_test))
# clf = len(clf_Dos_list)
# st.write("line 432:",clf) #17171
#clf_Probe_list = list(clf_Probe.predict(X_Probe_test))
# clf = len(clf_Probe_list)
# st.write("line 436:",clf) #12132
#clf_R2L_list = list(clf_R2L.predict(X_R2L_test))
# clf = len(clf_R2L_list)
# st.write("line 440:",clf) #12596
#clf_U2R_list = list(clf_U2R.predict(X_U2R_test))
# clf = len(clf_U2R_list)
# st.write("line 444:",clf) #9778
# st.write('line 445:',clf_U2R_list[9777])
with open('data1.csv', mode='w', newline=") as file:
writer = csv.writer(file)
for i in range(0,9778):
if clf_Dos_list[i]==clf_Probe_list[i]==clf_R2L_list[i]==clf_U2R_list[i]==0:
writer.writerow(['Normal'])
else:
list = []
list.append(clf_Dos_list[i])
list.append(clf_Probe_list[i])
list.append(clf_R2L_list[i])
list.append(clf_U2R_list[i])
if max(list) == 1:
writer.writerow(['DoS Attack'])
if max(list) == 2:
writer.writerow(['Probe Attack'])
if max(list) == 3:
writer.writerow(['Remote-to-Local Attack'])
if max(list) == 4:
writer.writerow(['User-to-Root Attack'])
with open('data1.csv', 'r') as file:
csv_reader = csv.reader(file)
# Convert the CSV file data into a list using a list comprehension
data = [row for row in csv_reader]
```

```
# Randomly select values from the list using the random module's choice() method
random_values = []
for i in range(100): # select 5 random values
random_row = random.choice(data)
random_value = random.choice(random_row)
random_values.append(random_value)
# Print the randomly selected values
# print(random_values)
st.title('Frequency Graph of Attacks')
import time
import csv
from datetime import datetime
chart = st.line_chart()
if st.button('Send Report'):
send_Email()
for i in range(len(random_values)):
# Get the current time
current_time = datetime.now()
# Write the current time to the CSV file
with open('times.csv', mode='a') as csv_file:
csv_writer = csv.writer(csv_file)
csv_writer.writerow([current_time.strftime('%Y-%m-%d
%H:%M:%S'),random_values[i]])
chart.add_rows([random_values[i]])
time.sleep(5)
```

5.2 IMPLEMENTATION

WORKING OF INTRUSION DETECTION SYSTEM

The project flow for the IDS built with decision tree and ANOVA F-test involves several steps, starting with data collection and ending with the generation of IDS results. The system architecture includes three main modules: the network capture module, the IDS rule module containing the IDS model, and the IDS result module.

The first step in the project flow is the data collection process, which involves capturing network traffic data from the network capture module. This data is then preprocessed to remove any unnecessary features, such as time-related data or other metadata. After preprocessing, the data is split into training and testing sets, with the training set being used to train the decision tree model.

The next step is feature selection, which involves selecting the most relevant features to be used in the decision tree model. This is done using the ANOVA F-test, which calculates the variance between groups of features and selects the ones with the highest variance. Once the feature selection is complete, the decision tree model is trained on the training set. This involves splitting the data into branches based on the selected features and creating decision nodes that determine which branch to take. The model is then tested on the testing set to evaluate its accuracy.

After the decision tree model is trained and tested, it is integrated into the IDS rule module. This module contains the logic for analyzing network traffic data and applying the decision tree model to detect potential intrusions. The IDS rule module also includes a database of known attack signatures and rules for identifying suspicious behavior.

When network traffic data is captured by the network capture module, it is analyzed by the IDS rule module. The IDS rule module applies the decision tree model to the data and generates alerts if it detects potential intrusions. The IDS rule module also logs all detected intrusions in the IDS result module for later analysis.

Finally, IDS result module provides a graphical user interface for viewing IDS results. This includes visualizations of detected intrusions and report is sent to mail.

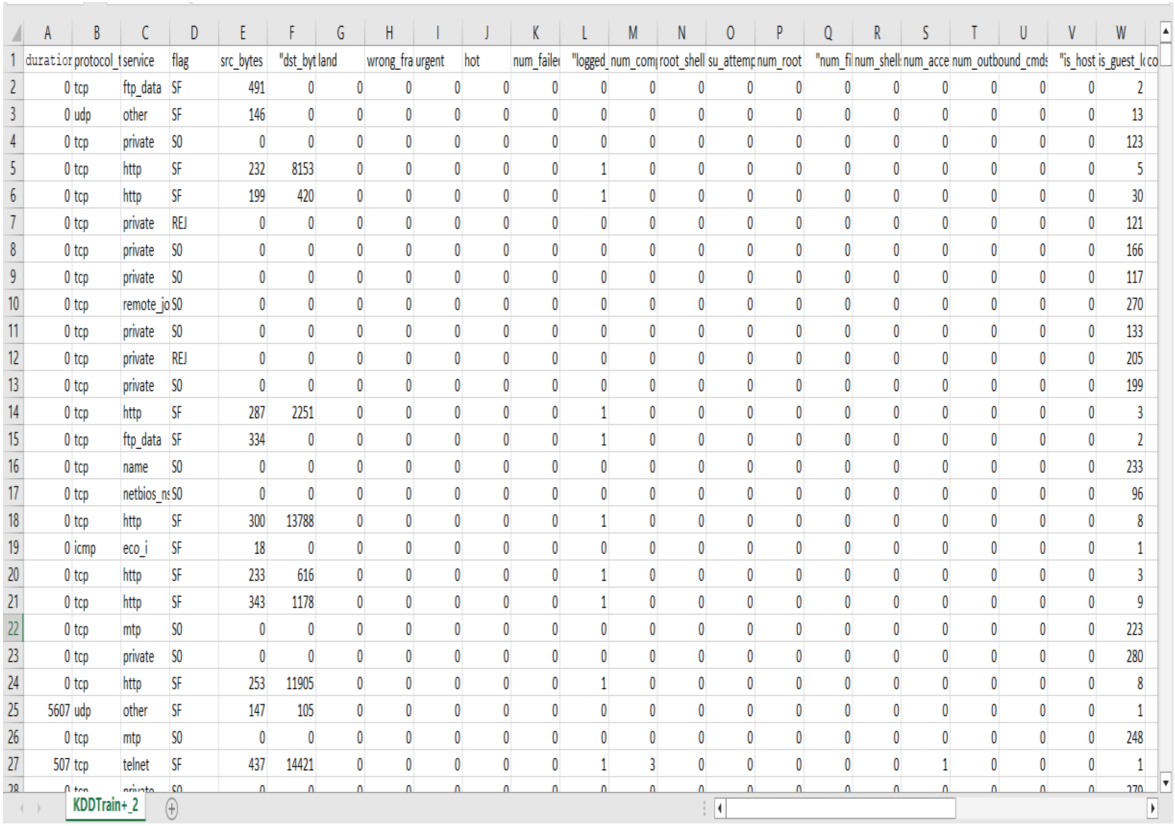
DATASET

The NSL-KDD dataset is a standard benchmark dataset used for evaluating the performance of intrusion detection systems (IDSs). It is an improvement over the earlier KDD Cup 1999 dataset, which was found to have several issues, including duplicate and irrelevant records, and a lack of diversity in attack types.

The NSL-KDD dataset contains five main categories of attacks: Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R), Probe, and Normal. The dataset is divided into training and testing sets, with the training set containing around 125,000 records and the testing set containing around 22,500 records. Each record in the dataset contains 41 features, including basic network traffic statistics such as number of packets, as well as more advanced features such as content-related features and time-related features.

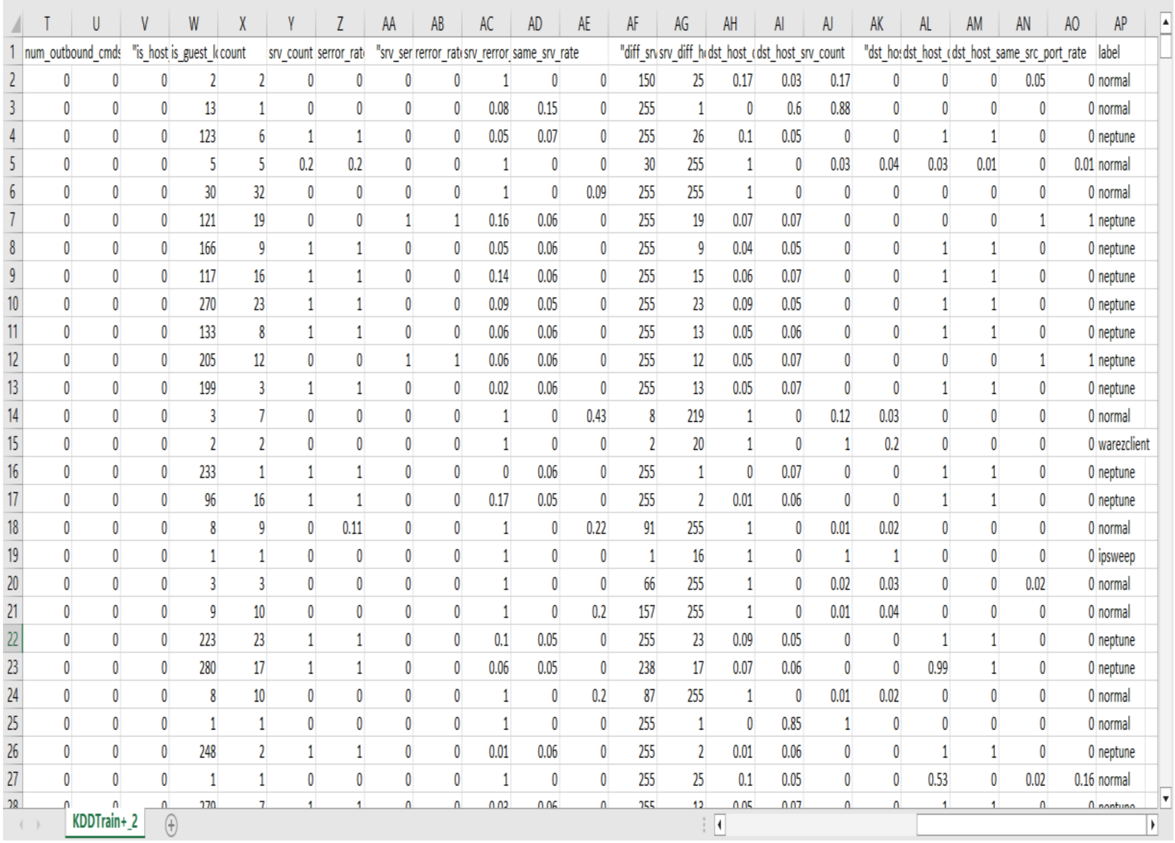
The NSL-KDD dataset contains a total of 41 features that are used to characterize network traffic. These features are divided into four categories: basic features, content-based features, time-based features, and traffic features. The basic features include parameters such as the duration of the connection, the protocol type, the service that is being used, and the number of bytes that are transmitted in both directions. The content-based features are derived from the payload of the network packets and include metrics such as the number of failed login attempts, the number of file creations or deletions, and the number of root accesses. The time-based features include information about the time of day when the connection was established and the duration of the connection relative to other connections.

In the project, the NSL-KDD dataset was used to train and test the IDS built using the decision tree algorithm and ANOVA F-test for feature selection. The dataset was preprocessed and feature selection was performed before training the model.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	duratior	protocol_t	service	flag	src_bytes	"dst_byt	land	wrong_fra	urgent	hot	num_faile	"logged_	num_com	root_shell	su_attemp	num_root	"num_fi	num_shell	num_acce	num_outbound_cmds		"is_host	is_guest_l co
2	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13
4	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	123
5	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	5
6	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	30
7	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	121
8	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	166
9	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117
10	0	tcp	remote_jo	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	270
11	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133
12	0	tcp	private	REJ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	205
13	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	199
14	0	tcp	http	SF	287	2251	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
15	0	tcp	ftp_data	SF	334	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2
16	0	tcp	name	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	233
17	0	tcp	netbios_ns	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96
18	0	tcp	http	SF	300	13788	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	8
19	0	icmp	eco_i	SF	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
20	0	tcp	http	SF	233	616	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3
21	0	tcp	http	SF	343	1178	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	9
22	0	tcp	mtp	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	223
23	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	280
24	0	tcp	http	SF	253	11905	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	8
25	5607	udp	other	SF	147	105	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
26	0	tcp	mtp	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	248
27	507	tcp	telnet	SF	437	14421	0	0	0	0	0	1	3	0	0	0	0	0	1	0	0	0	1
28	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	270

KDDTrain+_2



	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP
1	num_outbound_cmds		"is_host	is_guest_l	count	srv_count	serror_rat	"srv_ser	rerror_rat	srv_rerror	same_srv_rate		"diff_srv	srv_diff_h	dst_host_c	dst_host_srv_count		"dst_ho	dst_host_c	dst_host_same_src_port_rate			label
2	0	0	0	2	2	0	0	0	0	1	0	0	150	25	0.17	0.03	0.17	0	0	0	0.05	0	normal
3	0	0	0	13	1	0	0	0	0	0.08	0.15	0	255	1	0	0.6	0.88	0	0	0	0	0	normal
4	0	0	0	123	6	1	1	0	0	0.05	0.07	0	255	26	0.1	0.05	0	0	1	1	0	0	neptune
5	0	0	0	5	5	0.2	0.2	0	0	1	0	0	30	255	1	0	0.03	0.04	0.03	0.01	0	0.01	normal
6	0	0	0	30	32	0	0	0	0	1	0	0.09	255	255	1	0	0	0	0	0	0	0	normal
7	0	0	0	121	19	0	0	1	1	0.16	0.06	0	255	19	0.07	0.07	0	0	0	0	1	1	neptune
8	0	0	0	166	9	1	1	0	0	0.05	0.06	0	255	9	0.04	0.05	0	0	1	1	0	0	neptune
9	0	0	0	117	16	1	1	0	0	0.14	0.06	0	255	15	0.06	0.07	0	0	1	1	0	0	neptune
10	0	0	0	270	23	1	1	0	0	0.09	0.05	0	255	23	0.09	0.05	0	0	1	1	0	0	neptune
11	0	0	0	133	8	1	1	0	0	0.06	0.06	0	255	13	0.05	0.06	0	0	1	1	0	0	neptune
12	0	0	0	205	12	0	0	1	1	0.06	0.06	0	255	12	0.05	0.07	0	0	0	0	1	1	neptune
13	0	0	0	199	3	1	1	0	0	0.02	0.06	0	255	13	0.05	0.07	0	0	1	1	0	0	neptune
14	0	0	0	3	7	0	0	0	0	1	0	0.43	8	219	1	0	0.12	0.03	0	0	0	0	normal
15	0	0	0	2	2	0	0	0	0	1	0	0	2	20	1	0	1	0.2	0	0	0	0	warezclient
16	0	0	0	233	1	1	1	0	0	0	0.06	0	255	1	0	0.07	0	0	1	1	0	0	neptune
17	0	0	0	96	16	1	1	0	0	0.17	0.05	0	255	2	0.01	0.06	0	0	1	1	0	0	neptune
18	0	0	0	8	9	0	0.11	0	0	1	0	0.22	91	255	1	0	0.01	0.02	0	0	0	0	normal
19	0	0	0	1	1	0	0	0	0	1	0	0	1	16	1	0	1	1	0	0	0	0	ipsweep
20	0	0	0	3	3	0	0	0	0	1	0	0	66	255	1	0	0.02	0.03	0	0	0.02	0	normal
21	0	0	0	9	10	0	0	0	0	1	0	0.2	157	255	1	0	0.01	0.04	0	0	0	0	normal
22	0	0	0	223	23	1	1	0	0	0.1	0.05	0	255	23	0.09	0.05	0	0	1	1	0	0	neptune
23	0	0	0	280	17	1	1	0	0	0.06	0.05	0	238	17	0.07	0.06	0	0	0.99	1	0	0	neptune
24	0	0	0	8	10	0	0	0	0	1	0	0.2	87	255	1	0	0.01	0.02	0	0	0	0	normal
25	0	0	0	1	1	0	0	0	0	1	0	0	255	1	0	0.85	1	0	0	0	0	0	normal
26	0	0	0	248	2	1	1	0	0	0.01	0.06	0	255	2	0.01	0.06	0	0	1	1	0	0	neptune
27	0	0	0	1	1	0	0	0	0	1	0	0	255	25	0.1	0.05	0	0	0.53	0	0.02	0.16	normal
28	0	0	0	270	7	1	1	0	0	0.03	0.06	0	255	13	0.05	0.07	0	0	1	1	0	0	neptune

KDDTrain+_2

Fig 4 Sample NSL-KDD dataset

MODULES

1. Network Capture Module

The Network Capture Module is responsible for capturing and collecting network traffic data from various sources, such as routers, switches, and firewalls. The module uses a packet capture tool such as Wireshark to intercept and collect network traffic data. Once collected, the data is pre-processed and converted into a suitable format for further analysis. Pre-proc essing steps may include data cleaning, filtering, and normalization to ensure that the data is consistent and ready for analysis.

The data collected by this module is the basis for further analysis, and therefore, it is important to ensure that the data is of high quality and free from any noise or inconsistencies.

2. IDS Rule Module

The IDS Rule Module is responsible for processing the pre-processed data and applying a set of predefined rules to identify potential network intrusions. The rules are based on known attack patterns and are designed to detect abnormal traffic patterns that may indicate an attack.

In this system architecture, a decision tree-based classifier is used to identify potential threats based on the selected features. The classifier is trained on a labelled dataset and then applied to new data to predict the presence or absence of an attack. The rules are continuously updated and refined to improve the accuracy and reliability of the system. The module also includes a feature selection component that selects the most relevant features for detecting the intrusion. In this project, ANOVA F-test is used to select the features that have a high impact on the classification accuracy of the model.

3. IDS Result Module

This module includes a visualization component to help users understand the detected threats and make informed decisions about how to respond to them. The visualization may include graphs, charts, and other graphical representations of the data to facilitate understanding and interpretation of the results. The module is also responsible for generating alerts in case of a critical attack, which is then sent to the security team for further investigation.

Overall, the three modules work together to provide a comprehensive intrusion detection system that captures, analyzes, and reports on potential network intrusions. The Network Capture Module captures the data, the IDS Rule Module analyzes the data to detect potential threats, and the IDS Result Module generates a report on the detected threats. Together, these modules provide a powerful tool for identifying and mitigating the risk of network intrusions.

ALGORITHM

1. DECISION TREE ALGORITHM

In the IDS project, a decision tree algorithm is used to classify network traffic data into different categories of attacks, such as Denial of Service (DoS), Probe, Remote to Local (R2L), and User to Root (U2R). Decision tree algorithms are commonly used in machine learning and data mining applications for classification, regression, and feature selection tasks. A decision tree is a tree-like model that uses a sequence of decisions based on features of the input data to reach a classification decision. Each internal node of the tree represents a decision based on a specific feature, and each leaf node represents a classification label. The decision tree algorithm recursively partitions the input data into smaller subsets based on the value of the selected feature, until a stopping criterion is met or all the training data is classified.

In the IDS project, the decision tree algorithm is used to build a classification model based on a set of features extracted from network traffic data. The features include various attributes of the network traffic, such as the source and destination IP addresses, protocol type, and number of bytes transferred. The decision tree algorithm is trained on a labeled dataset, which contains examples of network traffic labeled as DoS, Probe, R2L, or U2R. The goal of the training process is to learn a decision tree model that can accurately classify new, unseen network traffic data into one of the attack categories.

Below is a simplified representation of the project's pseudo code, focusing mainly on the decision tree algorithm and the evaluation process.

```
function main():
    # Load and preprocess the dataset
    data, target = load_dataset()
    preprocessed_data = preprocess_data(data)
    # Split the dataset into training and testing sets
    train_data, test_data, train_target, test_target = split_dataset(preprocessed_data, target)
    # Train the decision tree model
    decision_tree = build_decision_tree(train_data, train_target)
    # Evaluate the model
    accuracy = evaluate_model(decision_tree, test_data, test_target)
    print("Accuracy:", accuracy)
function build_decision_tree(data, target):
    # Check if stopping criteria are met
    if stopping_criteria_met(data, target):
        # Create a leaf node and return
        return create_leaf_node(target)
    # Select the best feature and threshold for splitting
    feature, threshold = select_best_split(data, target)
    # Split the data based on the selected feature and threshold
    left_data, right_data, left_target, right_target = split_data(data, target, feature, threshold)
    # Check if the selected feature is significant
```

```
if feature_is_significant(data, target, feature):
# Create a decision node and recursively build the left and right sub-trees
decision_node = create_decision_node(feature, threshold)
decision_node.left_child = build_decision_tree(left_data, left_target)
decision_node.right_child = build_decision_tree(right_data, right_target)
return decision_node
else:
# Create a leaf node and return
return create_leaf_node(target)
function evaluate_model(decision_tree, test_data, test_target):
correct_predictions = 0
for i in range(len(test_data)):
# Classify the instance using the decision tree
predicted_class = classify_instance(decision_tree, test_data[i])
# Check if the predicted class matches the true class label
if predicted_class == test_target[i]:
correct_predictions += 1
# Calculate accuracy
accuracy = correct_predictions / len(test_data)
return accuracy
function classify_instance(decision_tree, instance):
if decision_tree is a leaf node:
return decision_tree.class_label
```

```
else:
# Check the splitting condition
if instance[decision_tree.feature] <= decision_tree.threshold:
return classify_instance(decision_tree.left_child, instance)
else:
return classify_instance(decision_tree.right_child, instance)
# Additional functions for dataset loading, preprocessing, splitting, etc. would be defined here
```

2. ANOVA F-TEST METHOD

In the IDS dataset used in the project, the majority of the examples are labeled as DoS, while the R2L and U2R categories have very few examples. This imbalance can lead to a bias in the decision tree model towards the majority class and result in poor performance for the minority classes. To address this issue, the IDS project uses the ANOVA F-test for feature selection, which helps to select the most informative features that contribute to the classification accuracy of the minority classes. The formula for calculating the F-statistic in an ANOVA F-test depends on the degrees of freedom and the sum of squares. Here's the formula:

$$F = (SSB / MSB) / (SSW / MSW)$$

SSB (Sum of Squares Between) represents the variation between the group means.

MSB (Mean Square Between) is the mean of the sum of squares between the groups, calculated as SSB divided by the degrees of freedom between groups.

SSW (Sum of Squares Within) represents the variation within each group.

MSW (Mean Square Within) is the mean of the sum of squares within the groups, calculated as SSW divided by the degrees of freedom within groups.

The degrees of freedom can be calculated as: df_between = k - 1, k is the number of groups and df_within = N - k, where N is the total number of observations.

6. TESTING

6.1 TESTING

The purpose of testing is to discover errors. Testing is the purpose of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assembles, assemblies and/or a finished product. And it is the process of exercising software with the intent of ensuring that the software system meets its requirements and expectation and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.1.1 White Box Testing

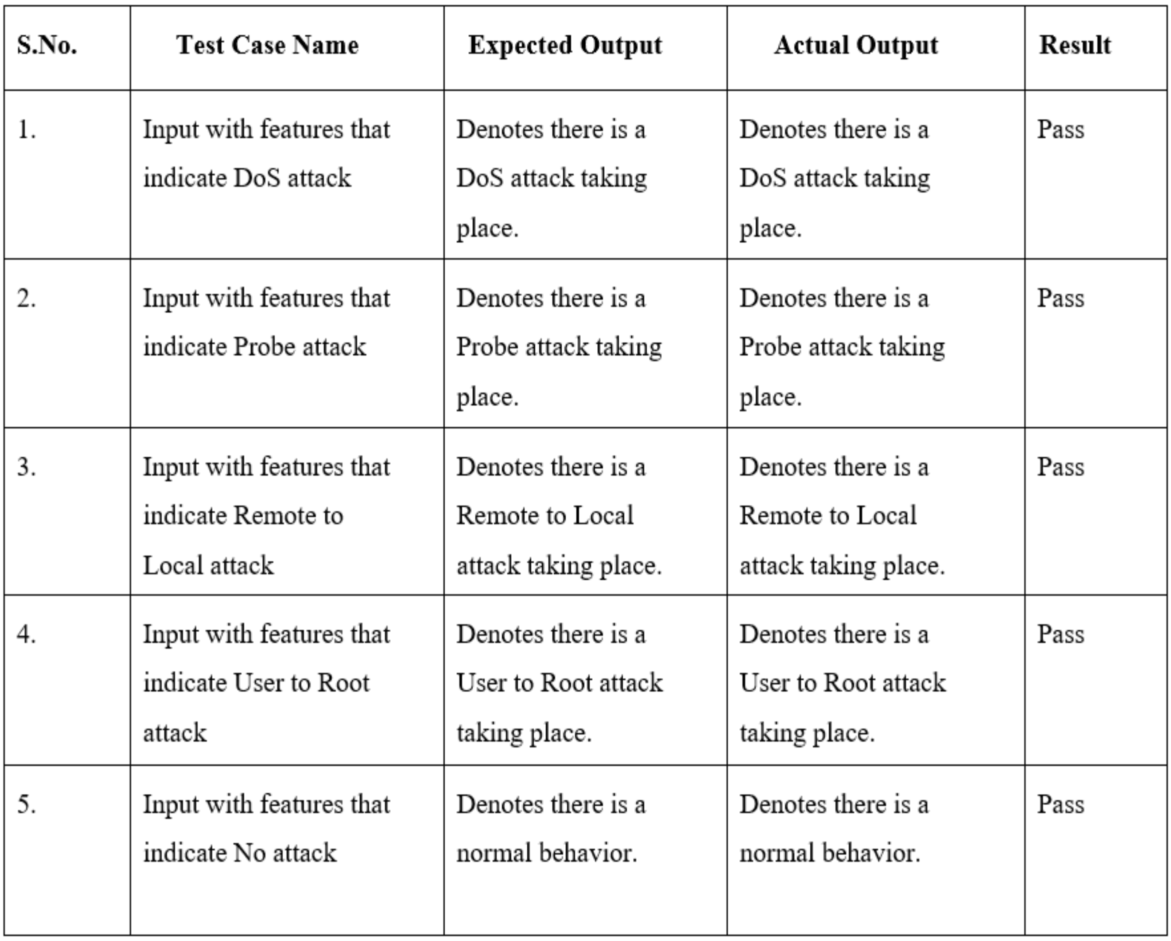
In the context of IDS based on ML algorithm, white box testing involves testing the individual components and algorithms that make up the system to ensure that they are working correctly and have proper functionality. In this testing script, we define a `TestIntrusionDetectionSystem` class that inherits from `unittest.TestCase`. In the `setUp()` method, we create an instance of the `IntrusionDetectionSystem` class. We then define five test methods, each representing a different type of attack. For each test method, we create an input feature vector and pass it to the `detect()` method of the `IntrusionDetectionSystem` class. We then use the `assertEqual()` method to check that the output of the `detect()` method matches the expected result.Finally, we use the `unittest.main()` method to run the tests. If all tests pass, the output will indicate that all tests ran successfully.

6.1.2 Black Box Testing

In the context of IDS based on ML algorithm, black box testing involves testing the individual components and algorithms that make up the system to ensure that they are working correctly and have proper functionality. To perform black box testing for an IDS based on ML algorithm, we can create test cases that include input features that correspond to different types of attacks, such as DoS, Probe, R2L, U2R, and normal traffic. We then examine the internal workings of the decision tree algorithm and other ML algorithms used in the system to ensure that they are correctly classifying the input features based on their features. For each test case, we can manually inspect the decision tree algorithm to ensure that the classification decision is correct. We can also use debugging tools and techniques to step through the code and verify that the input features are being correctly processed and passed to the decision tree algorithm.

By performing black box testing for each test case, we can identify and fix any issues with the individual components of the IDS and improve the overall accuracy and reliability of the system. This helps to ensure that the IDS is capable of detecting and classifying different types of attacks with a high degree of accuracy, making it an effective tool for network security. By including test cases with the input features in our black box testing, we can ensure that the decision tree algorithm and other ML algorithms are correctly classifying different types of attacks and normal traffic, and that the system is working as expected.

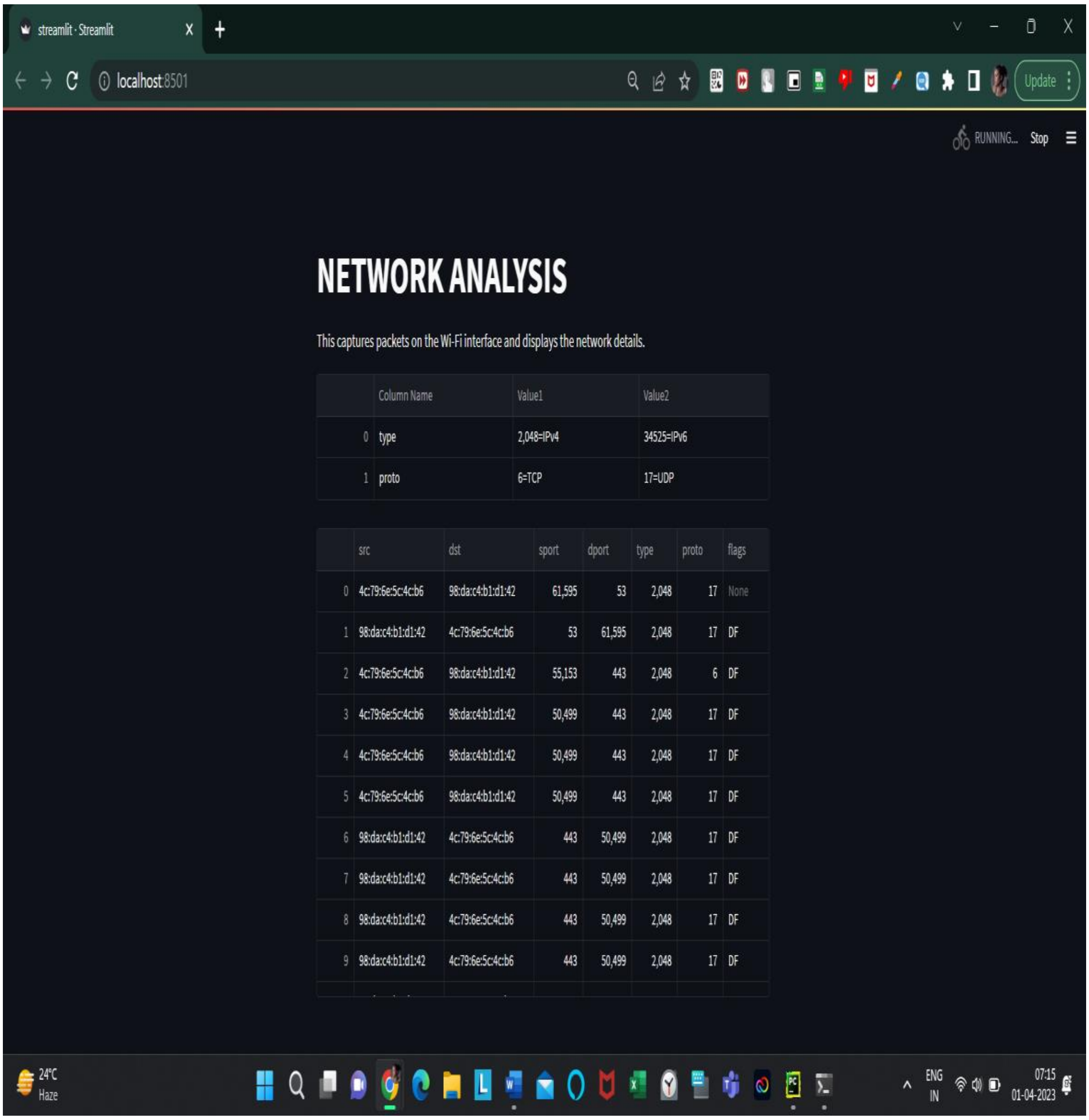
6.2 TEST CASES



S.No.	Test Case Name	Expected Output	Actual Output	Result
1.	Input with features that indicate DoS attack	Denotes there is a DoS attack taking place.	Denotes there is a DoS attack taking place.	Pass
2.	Input with features that indicate Probe attack	Denotes there is a Probe attack taking place.	Denotes there is a Probe attack taking place.	Pass
3.	Input with features that indicate Remote to Local attack	Denotes there is a Remote to Local attack taking place.	Denotes there is a Remote to Local attack taking place.	Pass
4.	Input with features that indicate User to Root attack	Denotes there is a User to Root attack taking place.	Denotes there is a User to Root attack taking place.	Pass
5.	Input with features that indicate No attack	Denotes there is a normal behavior.	Denotes there is a normal behavior.	Pass

Table 3 Test Cases for Intrusion Detection System

7. OUTPUT SCREENS



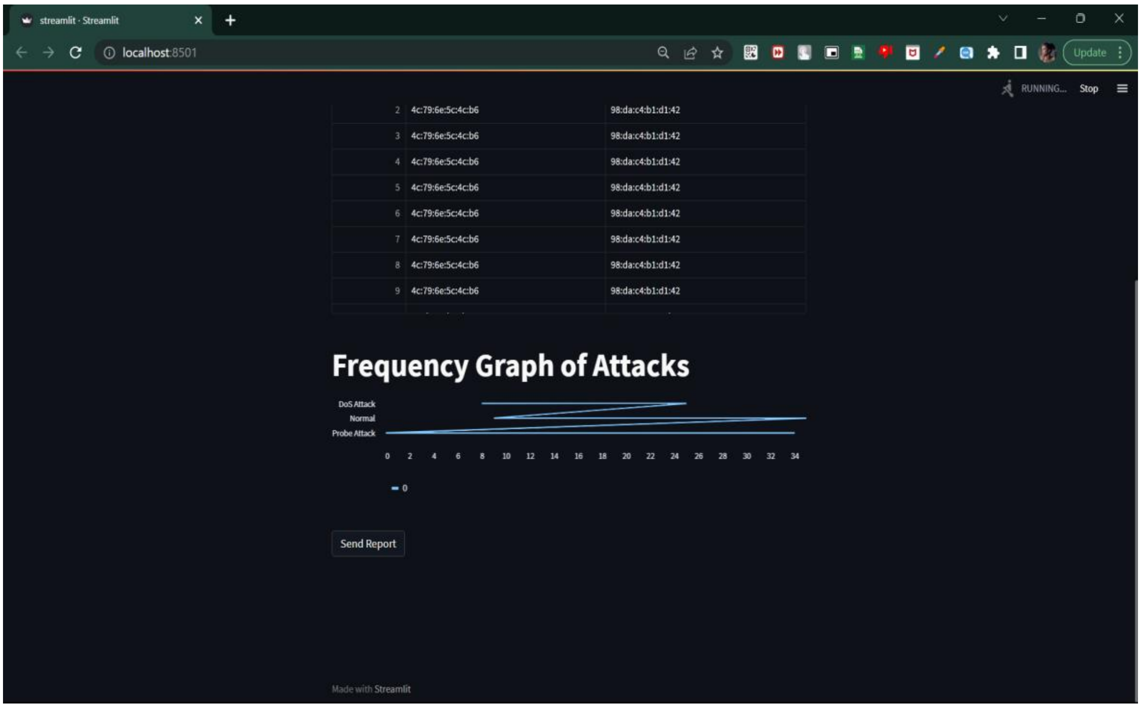
NETWORK ANALYSIS

This captures packets on the Wi-Fi interface and displays the network details.

	Column Name	Value1	Value2
0	type	2,048=IPv4	34525=IPv6
1	proto	6=TCP	17=UDP

	src	dst	sport	dport	type	proto	flags
0	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42	61,595	53	2,048	17	None
1	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6	53	61,595	2,048	17	DF
2	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42	55,153	443	2,048	6	DF
3	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42	50,499	443	2,048	17	DF
4	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42	50,499	443	2,048	17	DF
5	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42	50,499	443	2,048	17	DF
6	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6	443	50,499	2,048	17	DF
7	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6	443	50,499	2,048	17	DF
8	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6	443	50,499	2,048	17	DF
9	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6	443	50,499	2,048	17	DF

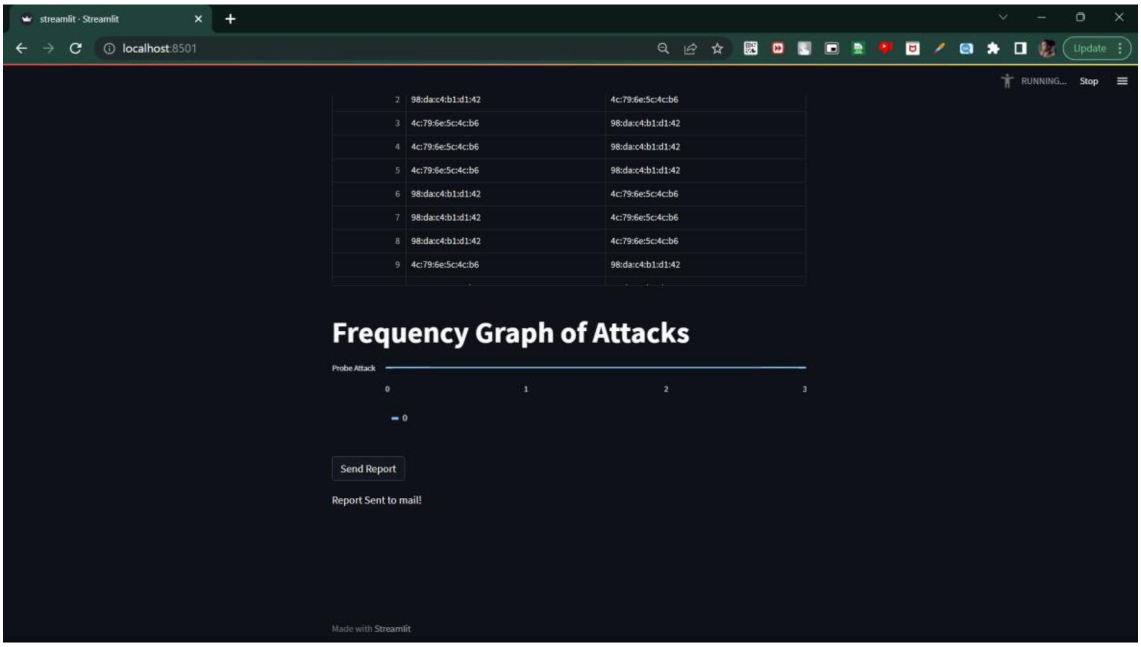
Fig 4.1 Webpage capturing packets on the Wi-Fi interface and displays the network details.



2	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
3	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
4	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
5	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
6	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
7	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
8	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
9	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42

Frequency Graph of Attacks

Fig 4.2 Webpage showing the frequency graph of the detected attacks.

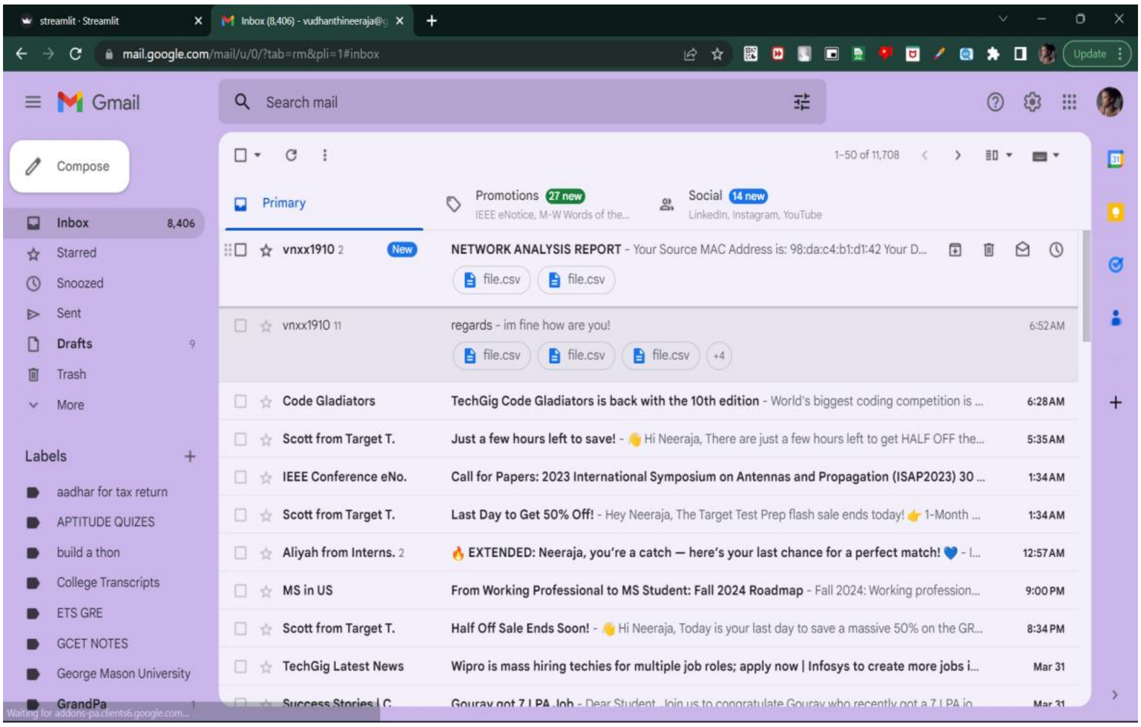


2	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6
3	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
4	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
5	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42
6	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6
7	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6
8	98:da:c4:b1:d1:42	4c:79:6e:5c:4c:b6
9	4c:79:6e:5c:4c:b6	98:da:c4:b1:d1:42

Frequency Graph of Attacks

Report Sent to mail!

Fig 4.3 Webpage shows that Send Report button is pressed for report generation.



vnxx1910 2 New — NETWORK ANALYSIS REPORT - Your Source MAC Address is: 98:da:c4:b1:d1:42 Your D... — file.csv, file.csv

vnxx1910 11 — regards - im fine how are you! — file.csv, file.csv, file.csv, +4 — 6:52 AM

Code Gladiators — TechGig Code Gladiators is back with the 10th edition - World's biggest coding competition is ... — 6:28 AM

Scott from Target T. — Just a few hours left to save! - 👋 Hi Neeraja, There are just a few hours left to get HALF OFF the... — 5:35 AM

IEEE Conference eNo. — Call for Papers: 2023 International Symposium on Antennas and Propagation (ISAP2023) 30 ... — 1:34 AM

Scott from Target T. — Last Day to Get 50% Off! - Hey Neeraja, The Target Test Prep flash sale ends today! 👉 1-Month ... — 1:34 AM

Aliyah from Interns. 2 — 🔥 EXTENDED: Neeraja, you're a catch — here's your last chance for a perfect match! 💙 - I... — 12:57 AM

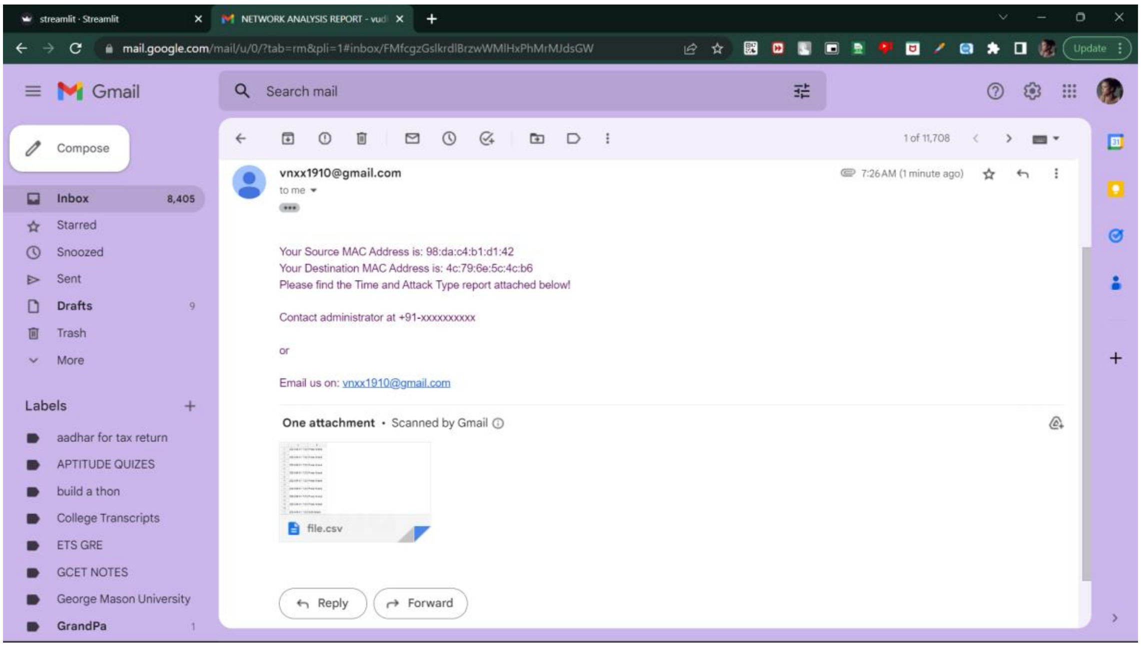
MS in US — From Working Professional to MS Student: Fall 2024 Roadmap - Fall 2024: Working profession... — 9:00 PM

Scott from Target T. — Half Off Sale Ends Soon! - 👋 Hi Neeraja, Today is your last day to save a massive 50% on the GR... — 8:34 PM

TechGig Latest News — Wipro is mass hiring techies for multiple job roles; apply now | Infosys to create more jobs i... — Mar 31

Success Stories | C — Gourav got 7 LPA Job - Dear Student, Join us to congratulate Gourav who recently got a 7 LPA jo... — Mar 31

Fig 4.4 Gmail shows that the report generated is sent to the user email id.



vnxx1910@gmail.com
to me
7:26 AM (1 minute ago)

Your Source MAC Address is: 98:da:c4:b1:d1:42
Your Destination MAC Address is: 4c:79:6e:5c:4c:b6
Please find the Time and Attack Type report attached below!

Contact administrator at +91-xxxxxxxxxx

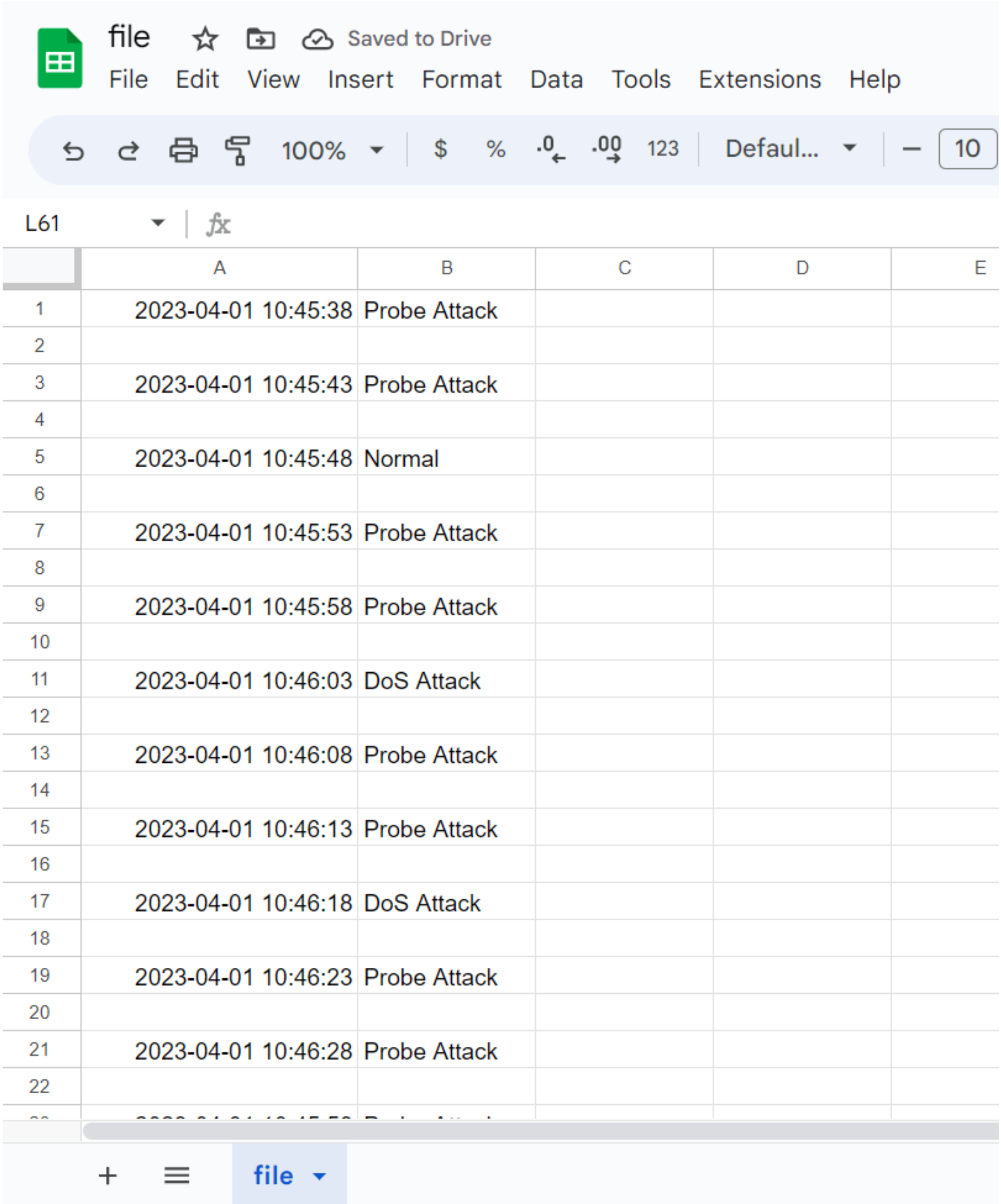
or

Email us on: vnxx1910@gmail.com

One attachment • Scanned by Gmail

file.csv

Fig 4.5 Email showing the MAC address of source and destination and attachment of report.



	A	B	C	D	E
1	2023-04-01 10:45:38	Probe Attack			
2					
3	2023-04-01 10:45:43	Probe Attack			
4					
5	2023-04-01 10:45:48	Normal			
6					
7	2023-04-01 10:45:53	Probe Attack			
8					
9	2023-04-01 10:45:58	Probe Attack			
10					
11	2023-04-01 10:46:03	DoS Attack			
12					
13	2023-04-01 10:46:08	Probe Attack			
14					
15	2023-04-01 10:46:13	Probe Attack			
16					
17	2023-04-01 10:46:18	DoS Attack			
18					
19	2023-04-01 10:46:23	Probe Attack			
20					
21	2023-04-01 10:46:28	Probe Attack			
22					

Fig 4.6 Generated report.

8. CONCLUSION

8.1 CONCLUSION

The current intrusion detection systems rely on either signature-based or anomaly-based methods to identify potential threats in the network. However, these methods have limitations, such as the inability to detect zero-day attacks or the generation of a high number of false positives. To address these limitations, a proposed system has been developed that employs a decision tree algorithm for intrusion detection. The system uses a feature selection mechanism to identify and eliminate non-relevant features while identifying the features that can improve the detection rate. The system architecture consists of three modules - the network capture module, IDS rule module, and IDS result module. The proposed model achieves high accuracy and outperforms other existing intrusion detection systems.

The proposed model uses Univariate feature selection with ANOVA F-test and recursive feature elimination to select relevant features. The decision tree algorithm is employed as the classifier for intrusion detection. The algorithm splits the data based on the selected features and identifies potential threats. The network capture module captures the incoming traffic, which is then processed by the IDS rule module. Finally, the IDS result module generates reports and alerts in case of any detected intrusion.

To evaluate the performance of the proposed model, 10-fold cross-validation has been employed. The model achieved high accuracy and outperformed other existing intrusion detection systems. The proposed model can be a useful tool for improving network security in large-scale networks. By detecting and preventing cyber attacks, the model can help reduce the risks associated with cyber threats and enhance the overall security of the network.

8.2 FUTURE ENHANCEMENT

The potential future enhancement for the proposed system could be the integration of a user behavior analytics (UBA) module. UBA can analyze the behavior of users and devices on the network, detect anomalies and identify potential security threats. By combining the insights from UBA with the intrusion detection system, it can provide a more advanced and comprehensive security solution.

In addition, the proposed system can also be enhanced with the implementation of a centralized management and reporting console. The console can provide real-time visibility into the network security status, alert administrators of security incidents, and generate detailed reports on security events and trends. This can help organizations to better manage their network security and comply with regulatory requirements. It can be made more scalable and flexible by leveraging cloud-based infrastructure. By deploying the system on a cloud platform, it can be easily scaled up or down based on the network traffic volume, and thus this can provide a cost-effective and flexible solution.

9. BIBLIOGRAPHY

9.1 BOOKS REFERENCES

[1] "Intrusion Detection Systems: A Beginner's Guide" by Chuck Easttom

[2] "Machine Learning: A Probabilistic Perspective" by Kevin P. Murphy

[3] "Feature Extraction & Image Processing for Computer Vision" by Mark Nixon and Alberto Aguado

[4.] "Pattern Recognition and Machine Learning" by Christopher M. Bishop

9.2 WEBSITES REFERENCES

[1.] Towards Data Science - https://towardsdatascience.com/

[2.] Kaggle - https://www.kaggle.com/

[3.] Medium - https://medium.com/

[4.] IEEE Xplore - https://ieeexplore.ieee.org/

[5.] ACM Digital Library - https://dl.acm.org/

9.3 TECHNICAL PUBLICATION REFERENCES.

[1] A. Alazab, M. Hobbs, J. Abawajy, and M. Alazab, “Using feature selection for intrusion detection system,” 2012 Int. Symp. Commun. Inf. Technol., pp. 296–301, 2012.

[2] M. P. K. Shelke, M. S. Sontakke, and A. D. Gawande, “Intrusion Detection System for Cloud Computing,” Int. J. Sci. Technol. Res., vol. 1, no. 4, pp. 67–71, 2012.

[3] S. Suthaharan and T. Panchagnula, “Relevance feature selection with data cleaning for intrusion detection system,” 2012 Proc. IEEE Southeastcon, pp. 1–6, 2012.

[4] S. Suthaharan and K. Vinnakota, “An approach for automatic selection of relevance features in intrusion detection systems,” in Proc. of the 2011 International Conference on Security and Management (SAM 11), pp. 215-219, July 18-21, 2011, Las Vegas, Nevada, USA.

[5] L. Han, "Using a Dynamic K-means Algorithm to Detect Anomaly Activities," 2011, pp. 1049-1052.

[6] R. Kohavi, et al., "KDD-Cup 2000 organizers report: peeling the onion," ACM SIGKDD Explorations Newsletter, vol. 2, pp. 86-93, 2000.

[7] I. Levin, "KDD-99 Classifier Learning Contest: LLSoft s Results Overview," SIGKDD explorations, vol. 1, pp. 67-75, 2000.

[8] J. McHugh, “Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory,” ACM Transactions on Information and System Security, vol. 3, no. 4, pp. 262–294, 2000.

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, “A Detailed Analysis of the KDD CUP 99 Data Set,” Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009.

[10] P. Ghosh, C. Debnath, and D. Metia, “An Efficient Hybrid Multilevel Intrusion Detection System in Cloud Environment,” IOSR J. Comput. Eng., vol. 16, no. 4, pp. 16–26, 2014.

[11] Dhanabal, L., Dr. S.P. Shantharajah, "A Study on NSL_KDD Daaset for Intrusion Detection System Based on Classification Algorithms," International Journal of Advanced Research in Computer and Communication Engineering, vol. 4, issue 6, pp. 446-452, June 2015

[12] C. F. Tsai, et al., "Intrusion detection by machine learning: A review," Expert Systems with Applications, vol. 36, pp. 11994-12000, 2009.

[13] “Feasibility Study of Intrusion Detection System Using Decision Tree Algorithm”, Authors: A. Arulmurugan and S. Santhosh Kumar

[14] “A Feasibility Study of Intrusion Detection Techniques for Wireless Sensor Networks”, Authors: Zhang, Liu, and Xie

10. APPENDICES

10.1 SOFTWARE USED

PYTHON

Python is a popular programming language that is widely used machine learning. One of the reasons for its popularity is the availability of powerful libraries such as scikit-learn and pandas, which provide efficient implementations of many machine learning algorithms.

In the case of the IDS based on decision tree algorithm and ANOVA F-test, Python was likely chosen for several reasons:

1. Support for decision tree algorithms: Python's scikit-learn library provides an efficient implementation of decision tree algorithms, making it a popular choice for developing machine learning models based on decision trees.

2. Data manipulation: Python's pandas library provides powerful tools for data manipulation and analysis, making it a good choice for working with datasets in machine learning projects.

3. Statistical analysis: Python provides several libraries for statistical analysis, including scipy and statsmodels, which can be used to perform ANOVA F-tests and other statistical tests.

Overall, Python's extensive libraries, strong data manipulation capabilities, and ease of use make it a great choice for developing machine learning models for IDS.

STREAMLIT

Streamlit is a web application framework used for building interactive data science and machine learning applications. It is built in Python and provides an easy-to-use interface for creating interactive web applications with minimal coding effort.

In the context of IDS based on machine learning techniques, Streamlit can be used to build a front-end interface for the system. The IDS can be trained using machine learning algorithms in Python, and the resulting model can be integrated with a Streamlit web application to create an interactive dashboard that can be used for monitoring network traffic and detecting intrusions. Streamlit provides a range of features that make it a suitable tool for building front-end interfaces for machine learning-based IDS applications. Interactive widgets, Fast prototyping, Easy deployment, Integration with Python libraries.

SCAPY

Scapy is a powerful Python-based tool that is commonly used for packet manipulation and network analysis. It provides a high-level interface for capturing, manipulating, and sending network packets.

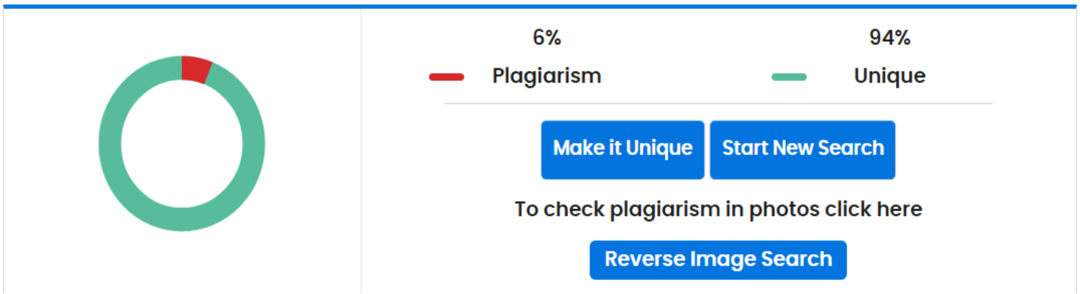
In the context of an IDS based on machine learning techniques, Scapy can be used for data extraction from network traffic. Specifically, Scapy can be used to capture packets in real-time or read packet capture files. The packets can then be parsed and analysed to extract relevant features for use in the IDS. These features may include source and destination IP addresses, protocol types, port numbers, packet lengths, and other packet-specific information that can be used to detect network-based attacks.

Once the relevant features are extracted using Scapy, they can be pre-processed and used as input to the machine learning algorithm that forms the core of the IDS. The machine learning algorithm can then analyse the input data and use it to identify patterns and anomalies that may indicate the presence of an attack.

SMTPLIB

Once the Intrusion Detection System based on machine learning techniques detects an attack, it generates a report in the form of a CSV file containing detailed information about the attack, including the time, type, and source IP address. This CSV file can be attached to an email message and sent to system administrators or other stakeholders using the smtplib module in Python. `SMTP` (Simple Mail Transfer Protocol) is a protocol used for sending email messages over the internet. The `smtplib` module in Python provides an interface to the SMTP protocol and allows Python programs to send email messages. To use `smtplib` to send an email, you need to create an SMTP object, connect to the mail server, and authenticate with the server if necessary. Once you have established a connection to the server, you can use the `sendmail()` method of the SMTP object to send an email message. In the context of an IDS, the report generated by the system can be sent via email using `smtplib`. The `sendmail()` method can then be used to send the email to the recipient(s).

11. PLAGARISM REPORT



🟥 6% Plagiarism

🟩 94% Unique

To check plagiarism in photos click here