e-ISSN: 2582-5208

International Research Journal of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

"NID-PROX"- INTRUSTION DETECTION FOR PROBE, DOS, U2L, R2U

Divya Dharshini. G[*1], Prathyusha. N[*2], Kughan. D.S.S[*3], Durga Devi. P[*4]

[*1,2,3,4]Department Of Computer Science Engineering, SRM Institute, Chennai, Tamil Nadu, India.

ABSTRACT

Network intrusion detection systems that use machine learning rely on flow attributes gathered from protocols, like NetFlow for their operation effectively. Modern ML based intrusion detection systems operate under the assumption that they can extract flow related data from every packet within a given flow instance. However the reality is that flow exporters are often implemented in devices where packet sampling's unavoidable. As such ML driven intrusion detection systems may encounter challenges when dealing with sampled data sets than streams of flow information. In our research study we investigate how sampling packets can impact the performance of ML based NIDS (Network Intrusion Detection Systems). Unlike studies where the parameters of the flow export stage played a role, in the evaluation process here they are not a factor affecting our assessment process. This means that NIDS performance can still be accurately evaluated with packet sampling in place. Our experimental results demonstrate that fraudulent flows containing smaller packets sizes could possibly go unnoticed when using sampling rates low as 1 in 10 or 1 in 100. By employing an evaluation methodology we delved into how various sampling strategies influence both the detection and false alarm rates of NIDS. The ability to promptly and reliably detect vulnerabilities, within network systems has become increasingly crucial today. This system will undergo training using machine learning techniques to identify network packet assaults.

Keywords: Network Intrusion Detection System (NIDS), Network Security Strategies, Dynamic NIDS Response.

I. INTRODUCTION

For several years, computer networks have played a crucial role in communication, enabling connected devices to share information with other systems and individuals. Through these networks, businesses, organizations, colleges, and universities can provide accessible services. As a result, the networking industry has grown rapidly, fueled by increased interest in internet accessibility. Consequently, information security has become one of today's most pressing concerns. The data we plan to transmit must be safeguarded to prevent unauthorized access.

In the realm of information security, three primary aspects are critical: confidentiality, integrity, and availability. Confidentiality ensures privacy by permitting only authorized users to access a system over the internet, identifying them through processes that incorporate accountability. Integrity, the second core principle, guarantees the accuracy of information, assuring users that data has not been altered by unauthorized parties.

Intrusion Detection Systems (IDS) are designed to identify malicious activities within a network, classifying network traffic records as either attack or normal based on preliminary monitoring. When unclassified records appear, IDS can assign them to either category. This system serves as an alert mechanism, notifying users of suspicious activity. The detection rate indicates IDS accuracy, and strong performance ensures reliable detection. Some IDS solutions are even capable of preventing attacks, helping to protect organizations from potential harm.

1.1 OBJECTIVE

This report serves to concisely explore the application of machine learning in network intrusion detection. By distilling key principles, methodologies, and recent advancements, we aim to equip readers with actionable insights into leveraging machine learning for bolstering cybersecurity defenses. Through an eight-line examination, we strive to elucidate the efficacy and relevance of machine learning in mitigating evolving cyber threats, fostering informed decision-making, and fostering a proactive approach to safeguarding digital infrastructures. In recent years, the proliferation of sophisticated cyber threats has necessitated the evolution of defensive strategies. Machine learning, with its ability to detect patterns and anomalies within vast datasets, presents a promising avenue for enhancing network security. By leveraging algorithms capable of learning from historical data, organizations can detect and respond to intrusions in real-time, minimizing the potential impact of cyber-attacks. Moreover, the adaptive nature of machine learning algorithms enables them to continually improve their accuracy over time, staying ahead of emerging threats. This proactive approach not only strengthens defenses but also reduces reliance on reactive measures, leading to more robust cybersecurity postures. As the cybersecurity landscape continues to evolve, embracing machine learning holds immense potential for staying one step ahead of malicious actors and preserving the integrity of digital infrastructure.

1.1.1 SCOPE

The scope of this study encompasses an in-depth exploration of network intrusion detection and its vital role in cybersecurity. It begins by introducing the importance of detecting and preventing unauthorized access to secure networks, setting the stage for a review of traditional intrusion detection methods, such as signature-based and anomaly-based approaches, along with their inherent limitations in adaptability and accuracy. This background leads into an explanation of essential machine learning (ML) concepts, emphasizing how these advanced techniques address the shortcomings of traditional methods. The study then delves into various ML algorithms specifically applied in intrusion detection, such as decision trees, support vector machines, and neural networks, to understand their unique advantages in identifying complex attack patterns. Real-world applications and case studies are examined to highlight the effectiveness of ML-based intrusion detection systems, showcasing instances where these systems have significantly improved detection rates and response times. Additionally, challenges and considerations in implementing ML for intrusion detection—such as data quality, computational requirements, and evolving threat landscapes—are discussed to underscore the practical complexities involved. Finally, the study provides recommendations for integrating ML into existing cybersecurity frameworks, offering insights into enhanced threat detection, response strategies, and overall mitigation of security risks in modern networks.

1.2 LITERATURE SURVEY

"Detect selective forwarding attacks in wireless sensor networks through support vector machines,"

The proposed approach presents several merits and demerits in the context of intrusion detection in Wireless Sensor Networks (WSNs). Among its advantages, it addresses a significant security issue inherent in WSNs and utilizes Support Vector Machines (SVMs) effectively for detecting intrusions. The implementation of sliding windows allows the system to adapt to dynamic network changes, enhancing its responsiveness to evolving threats. Additionally, the approach provides a comprehensive evaluation against conventional schemes, demonstrating its effectiveness in improving intrusion detection capabilities. However, it also has notable demerits; it relies on a centralized architecture, which may introduce scalability and reliability issues as the network expands. Furthermore, it does not account for collaborative attacks, which limits its applicability in more realistic scenarios where multiple attackers may coordinate their efforts.

"Detect blackhole attack on aodv-based mobile ad hoc networks through dynamic learning method."

The proposed method for intrusion detection in Mobile Ad Hoc Networks (MANETs) presents several notable merits and demerits. Among its advantages, it effectively addresses a significant security issue in MANETs and implements a dynamic learning method that enhances adaptability to changing network conditions. Additionally, the approach conducts a thorough evaluation against conventional methods, showcasing its potential to improve detection capabilities. However, there are also several drawbacks to consider; the method assumes network homogeneity, which may not accurately reflect real-world scenarios where network conditions can vary widely. It also does not account for collaborative attacks, which limits its applicability in situations where multiple attackers may coordinate their efforts. Furthermore, the lack of countermeasures for detected attacks reduces the overall effectiveness of the system in providing comprehensive security solutions.

"The main Cross-feature analysis for detecting ad-hoc routing anomalies,"

The proposed approach for intrusion detection in Mobile Ad Hoc Networks (MANETs) offers several significant merits and demerits. On the positive side, it addresses a critical security issue prevalent in MANETs and introduces a novel data mining technique that enhances anomaly detection capabilities. Furthermore, the method conducts a thorough evaluation against conventional techniques, demonstrating its potential to improve detection accuracy and responsiveness. However, there are notable drawbacks; the approach assumes network homogeneity, which may limit its applicability in diverse real-world environments where network conditions vary. Additionally, it does not account for collaborative attacks, thereby reducing the robustness of the system in scenarios where multiple attackers coordinate their efforts. Lastly, the lack of countermeasures for detected attacks impacts the overall effectiveness of the approach, as it fails to provide comprehensive security solutions to mitigate identified threats.

“Denial-of-service (dos) detection through practical entropy estimation at hierarchical sensor networks,”

The proposed approach for detecting Denial of Service (DoS) attacks in hierarchical sensor networks presents several important merits and demerits. On the merits side, it effectively addresses a significant security issue within these networks and introduces a novel technique that utilizes entropy-based detection for enhanced identification of DoS attacks. Additionally, the method conducts a thorough evaluation against conventional techniques, showcasing its potential to improve detection accuracy and responsiveness. However, there are also notable demerits; the approach assumes network homogeneity, which may limit its applicability in real-world scenarios where network conditions can vary widely. Furthermore, it does not take into account collaborative attacks, which reduces the robustness of the system in situations where multiple attackers coordinate their efforts. Lastly, the lack of countermeasures for detected attacks impacts the overall effectiveness of the approach, as it does not provide comprehensive solutions to mitigate identified threats.

II. METHODOLOGY

2.1 SYSTEM ANALYSIS

2.1.1 EXISTING SYSTEM

Numerous intrusion detection strategies, tactics, and algorithms aid in identifying those various attacks. The primary goal of the current work is to present a thorough analysis of intrusion detection, including types of intrusion detection methods, types of assaults, various tools and approaches, research needs, and problems. Lastly, the IDS Tool for Research Purposes is developed. These tools are able to identify and stop the intruder's intrusion. Existing work in network intrusion detection using machine learning encompasses various approaches, including deep learning, ensemble methods, anomaly detection, and adversarial machine learning. Researchers leverage deep learning techniques like CNNs and RNNs for automatic feature learning from raw network data. Ensemble methods combine multiple classifiers to improve detection performance, while anomaly detection techniques, such as clustering and autoencoders, detect deviations from normal behavior. Feature selection and dimensionality reduction methods enhance efficiency, while adversarial machine learning and transfer learning improve robustness and adaptability. Online learning enables real-time adaptation to evolving threats, and explainable AI techniques aid in understanding model decisions. Collectively, these approaches contribute to more effective and adaptive intrusion detection systems, capable of combating evolving cyber threats in networked environments. These approaches aim to enhance detection accuracy, but challenges like interpretability, computational complexity, and vulnerability to adversarial attacks persist. Balancing performance and complexity are crucial, alongside addressing issues like data imbalance and overfitting. Future research may focus on robustness, scalability, and real-time adaptation to emerging threats.

There are few disadvantages of the proposed approach of machine learning, with its advantages. Among such limitations, one major limit is limited interpretability- deep learning models such as CNNs and RNNs make it very tough to understand why a given model is making predictions. High computational complexity is the other problem since most deep learning and ensemble methods need many computational resources for both training and inference. This limits their scalability and applicability in environments where resources are constrained. Complex models also suffer from overfitting, in which they might memorize noise or irrelevant patterns in the training data, thus generalizing poorly on unseen data. Overfitting can increase false positives and false negatives in intrusion detection, thus weakening overall model reliability. Another challenge is data imbalance, commonly seen in network intrusion detection datasets, where normal instances heavily outweigh malicious ones. This imbalance biases the learning process and reduces the model's effectiveness in identifying rare or novel attacks. Furthermore, machine learning models are vulnerable to adversarial attacks, where minor perturbations in input data can lead to misclassification or evasion of detection, compromising the reliability and effectiveness of existing intrusion detection systems.

2.1.2 PROPOSED SYSTEM

With the exponential growth of computer networks and the proliferation of applications running atop them, network security has become increasingly paramount. The rise in security vulnerabilities across systems coupled with the potential economic impact of attacks underscores the urgent need for accurate and real-time detection of vulnerabilities within network systems. Traditional methods have proven insufficient in the face of evolving threats, necessitating innovative approaches like machine learning (ML) algorithms. These algorithms offer the promise of autonomously identifying patterns indicative of cyber threats within network traffic, enabling rapid detection and response. By continuously analyzing network data in real-time, ML-based intrusion detection systems can adapt to emerging threats, providing a proactive defense mechanism against cyber-attacks. Moreover, ML models can distinguish between benign network activity and malicious behavior, reducing false positives and minimizing disruptions to legitimate network operations. Integrating ML algorithms into network security frameworks not only enhances organizational security but also aids in compliance with regulatory requirements and industry standards governing data protection and privacy. As cyber threats continue to escalate in volume and sophistication, investing in ML- powered network security solutions becomes imperative for organizations looking to mitigate risks and fortify their defenses against cyber-attacks. The proposed machine learning algorithm for network security operates through a systematic process starting with data collection, wherein diverse network data is gathered including packet headers and flow information. Subsequently, preprocessing techniques are applied to refine the data before it undergoes training. During training, the algorithm learns to discern patterns indicative of benign or malicious network behavior through supervised or unsupervised learning methods. Feature selection is employed to identify pertinent attributes contributing to the algorithm's efficacy. Following model construction, evaluation metrics are utilized to assess its performance in accurately detecting threats while minimizing false positives and negatives.

The proposed machine learning algorithm offers several advantages in network intrusion detection, primarily through enhanced accuracy in identifying network vulnerabilities and malicious activities by employing advanced pattern recognition techniques. By analyzing network data in real time, the algorithm enables rapid detection and mitigation of potential threats, effectively reducing cyberattack risks and minimizing their impact on network operations. Its adaptability to evolving cyber threats further strengthens the network’s security posture, ensuring resilience against emerging attack vectors. Additionally, rigorous feature selection and evaluation processes reduce false positives, resulting in more reliable and actionable security alerts that prevent unnecessary disruptions to legitimate network activities. The automated nature of this detection process optimizes resource utilization, focusing security efforts on high-risk areas and thereby improving operational efficiency. Finally, implementing a machine learning-based intrusion detection system assists organizations in adhering to regulatory and compliance standards for data protection and privacy, bolstering overall regulatory compliance.

2.2 SYSTEM REQUIREMENT SPECIFICATION

The System Requirement Specification (SRS) document is a comprehensive blueprint for software development, detailing both functional and non-functional requirements. It begins with an introduction, providing an overview of the system's purpose and scope. Functional requirements outline the system's features and behaviors, while non-functional requirements specify performance, security, usability, and reliability aspects.

2.2.1 SOFTWARE REQUIREMENTS

• Operating System(pc)	:	Windows 7,8,10 (64 bit)
• Software	:	Python
• Tools	:	Anaconda (Jupyter notebook IDE)

2.2.3 HARDWARE REQUIREMENTS

• Hard Disk	:	500GB and above
• RAM	:	4GB and above
• Processor	:	I5 and above

2.3 SOFTWARE DESCRIPTIONS REQUIREMENT ANALYSIS

Requirement analysis is a critical phase in the project's development process. It involves gathering, documenting, and understanding the specific needs and expectations of stakeholders for the weather prediction application. This phase focuses on identifying the functional and non-functional requirements, user expectations, system constraints, and desired features. By analyzing the requirements, the project team can define a clear scope and roadmap for the application's development, ensuring that it meets the users' needs and aligns with the project's objectives.

2.3.1 PYTHON:

Python is an interpreted, high-level, dynamically-typed, free, and open-source programming language. It supports multiple programming paradigms, including procedural and object-oriented styles, without requiring variable type declarations. For example, if a variable x is assigned with x=10 , x can later be reassigned as a string or another type, reflecting Python's flexibility. Similar to PERL, Python has become highly popular due to its readability and compatibility across various operating systems, such as UNIX-based systems, macOS, MS-DOS, OS/2, and many versions of Microsoft Windows. This versatility, combined with its easy-to-learn syntax, makes Python a practical choice for many developers.

Python was created by Guido van Rossum, who named it after his favorite comedy show, *Monty Python's Flying Circus*. Its source code is publicly accessible, allowing for modification and reuse, which has broadened its application across multiple industries.

Key Features of Python:

1. **Free and Open Source**: Python’s source code is accessible to everyone, encouraging collaborative development.
2. **Object-Oriented Language**: Python supports object-oriented programming, enabling modular and reusable code structures.
3. **High-Level Language**: It provides an easy-to-understand syntax and abstracts complex system details, suitable for both beginners and experts.
4. **Support for GUI Programming**: Python includes libraries for developing graphical user interfaces.
5. **Portability**: Python code can run across different platforms without modification.
6. **Integrated Language**: Python can seamlessly integrate with other programming languages, making it highly versatile.
7. **Interpreted Language**: Python code is executed line-by-line, which aids in debugging and accelerates development.

2.3.2 ANACONDA

The Anaconda installation includes over 250 pre-installed packages, along with Anaconda's package and virtual environment manager, which allows access to more than 7,500 additional open-source packages from PyPI. Anaconda also provides a graphical interface, Anaconda Navigator, for package management, making it user-friendly compared to command-line tools.

A key distinction between Anaconda and pip, another popular package manager, is how they handle package dependencies. Pip installs each package along with its dependencies without checking for potential conflicts with previously installed packages. This can lead to compatibility issues, especially in data science, where different versions of libraries like numpy may be required for various tools. For instance, installing a package through pip that uses a different version of numpy could disrupt an existing installation of TensorFlow, potentially causing errors or inconsistent results.

Anaconda, however, checks the entire environment and any version constraints (e.g., TensorFlow 2.0 or higher) before installing packages. If Anaconda detects an incompatibility, it issues a warning, helping prevent conflicts. Users can install packages individually from Anaconda repositories, Anaconda Cloud, or custom mirrors using the conda install command. Packages are compiled by Anaconda, Inc., with support for macOS (64-bit), Linux (64-bit), and Windows (32-64 bit).

Although pip can still be used within an Anaconda environment to install packages from PyPI, Anaconda manages both conda and pip installations. For users interested in building custom packages, the conda build command enables creation and sharing on platforms like PyPI and Anaconda Cloud. Anaconda2 installs Python 2.7 by default, while Anaconda3 installs Python 3.7, but either version can be used to create new environments with different Python versions as needed.

2.3.2.1 ANACONDA NAVIGATOR:

Anaconda Navigator is a desktop GUI included with the Anaconda distribution that allows users to launch applications and manage packages, environments, and channels without using command-line commands. Through Navigator, users can search for packages on Anaconda Cloud or a local repository, install them into an environment, and run or update them as needed. This tool is compatible with Linux, macOS, and Windows, making it accessible across major operating systems.

By default, the following applications are available in Navigator:

- JupyterLab
- Jupyter Notebook
- Qt Console
- Spyder
- Glue
- Orange
- RStudio Visual Studio Code

2.3.3 JUPYTER NOTEBOOK

Jupyter Notebook is a web-based interactive computational environment used for creating and sharing documents in the Jupyter format. The term "notebook" may refer to the Jupyter web application, the Jupyter Python web server, or the specific document format. A Jupyter Notebook document is a JSON file that adheres to a versioned schema and contains an ordered list of input/output cells. These documents, typically saved with the ".ipynb" extension, can include code, text formatted with Markdown, mathematical expressions, visualizations, and multimedia content.

There are many kernels that can be used with the Jupyter Notebook, enabling developers to produce multilingual content. IPython kernel is already included by default with the Jupyter Notebook. As of 2.3 release [11][12] October 2014), 49 Jupyter compatible kernels exist for quite a range of computer programming languages that include Python, R, Julia, Haskell.

Feasibility study This phase involves the evaluation of the project's viability and the submission of a business proposal with a general plan of the project plus some cost estimates. System analysis involves a feasibility study about the proposed system. This is done to make sure that the proposed method will not burden the business. For feasibility study, the basic understanding of the primary requirements of the system must be known.

This would mean that the feasibility analysis would consider three essential factors such as Financial feasibility, Technical feasibility, Social feasibility.

Financial viability

The objective of this research is to evaluate the system's potential financial effect on the organization. The business can only spend a specific amount of money on the research and development of the system. It has to be justified for the costs. Since most of the technologies used were free, the designed system could also fit in the budget because only the customized items have to be purchased.

Technical feasibility

This is to assess technical requirements of the system or even technical feasibility. Every design system should not seek or demand much use of available technical resources. Many available technical resources will thus get stretched because of this. The client will then bear much strain because of that. Since this system requires minimal or non-minor changes for implementation, the requirements of its design must be low.

Social practicability One of the purposes of research is to determine the level of user acceptability to the system and this is through training the users on the proper use of the system. The techniques applied in educating and exposing the user to the systems are the only factors determining the extent of adoption by the users. This is because he is the last user of the system, who should be put at ease since he might provide some incisive critique, which is accepted.

III. MODELING AND ANALYSIS

3.1 ARCHITECTURE DIAGRAM:

The Intrusion Detection System (IDS) architecture includes network sensors and host agents that monitor network and host activities, forwarding data to a central data collection engine. This engine processes the data for analysis by the detection engine, which identifies potential threats using various techniques. Detected threats trigger alerts logged in the system and sent to administrators through the alerting and logging system. Administrators manage and configure the IDS through the management console, which also provides reporting and visualization capabilities.

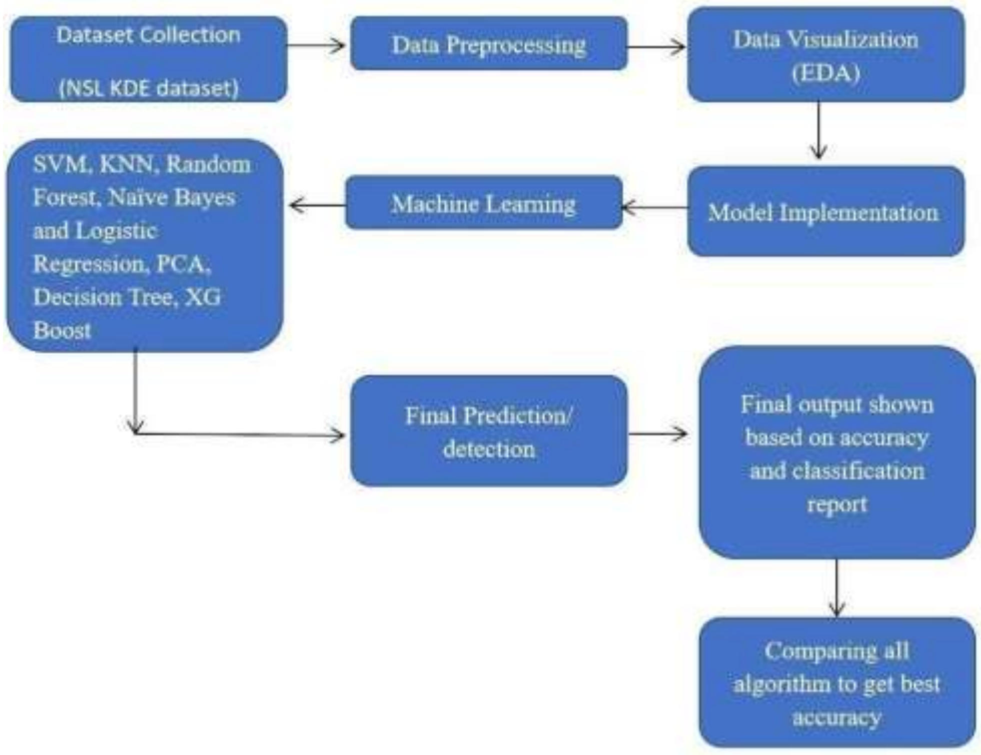

Dataset Collection
(NSL KDE dataset)
Data Preprocessing
Data Visualization (EDA)
Model Implementation
Machine Learning
SVM, KNN, Random Forest, Naive Bayes and Logistic Regression, PCA, Decision Tree, XG Boost
Final Prediction/ detection
Final output shown based on accuracy and classification report
Comparing all algorithm to get best accuracy

Fig 3.1:

3.2. Use Case Diagram:

A use case diagram for a Network Intrusion Detection System (NIDS) can be visualized with two main actors: the Network Administrator and the NIDS itself. The administrator interacts with the NIDS through a use case called "Manage System." This allows them to configure detection methods (signature-based or anomaly-based), update threat databases, and set alert thresholds.

The NIDS has two primary use cases. One is "Monitor Network Traffic" where it continuously captures and analyzes network packets. The other is "Detect Intrusion" where it identifies suspicious activity based on its configured methods. If an intrusion is detected, the NIDS initiates the "Generate Alert" use case, notifying the administrator through various channels like email or a dashboard. The administrator can then take action through the "Respond to Alert" use case, such as blocking the attacker or investigating further.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

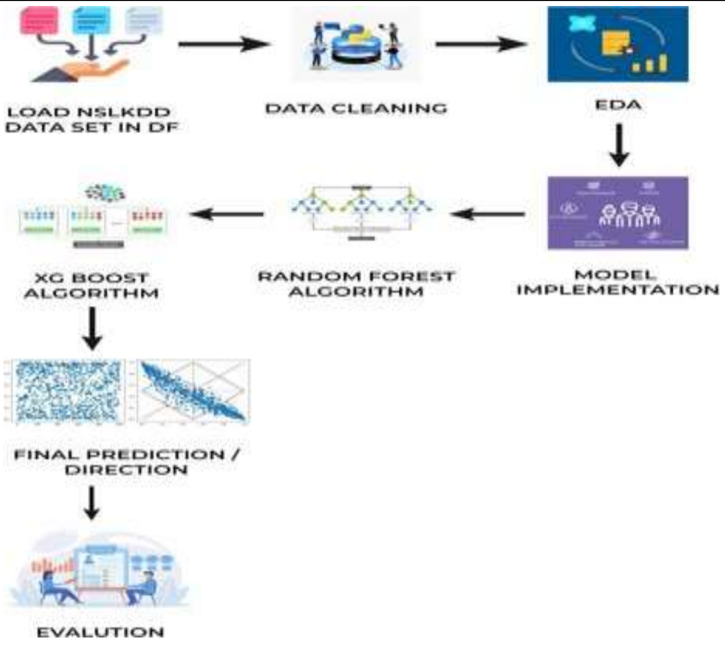


Fig 3.2:

3.3 SEQUENCE DIAGRAM:

Once validated, the Deployment Module integrates the model into the The "Network Security Intrusion Detection Sequence Diagram" illustrates the step-by-step process involving various components in detecting and responding to network intrusions. It begins with the Network Sensor forwarding collected data to the Preprocessing Module, which prepares the data for analysis. Subsequently, the data is passed to the Machine Learning Model, where it is examined for signs of normal or malicious behavior. The Feature Selection Component optimum production environment for real-time monitoring. Any detected suspicious activity triggers the Response System, which takes appropriate actions to address the threats, ensuring the network's security and integrity.

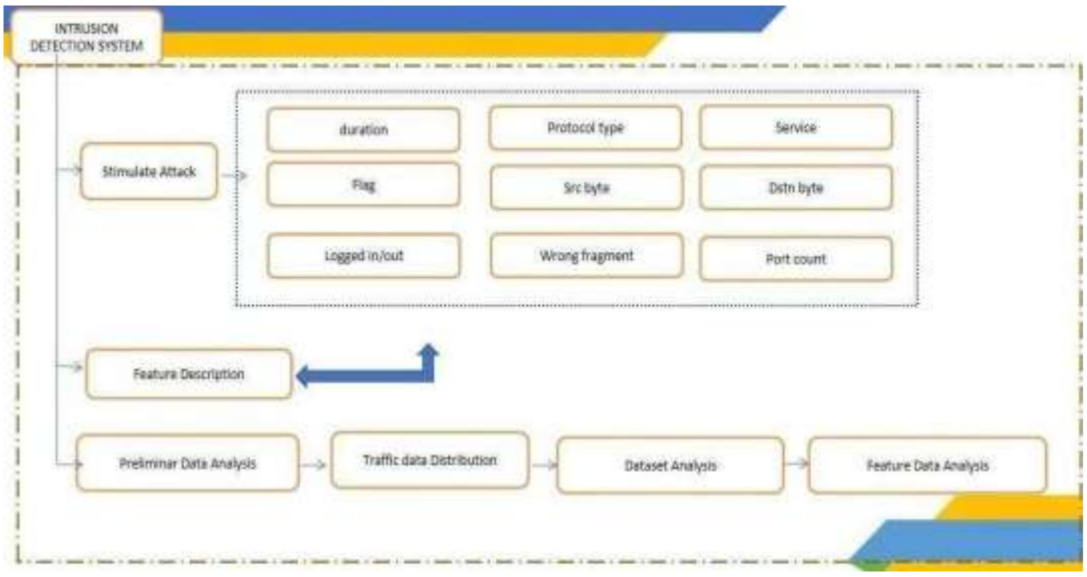


Fig 3.3:

The diagram outlines how we protect networks. Data is collected and cleaned, then checked for problems by a smart system. If anything looks off, action is taken to fix it and keep the network safe.

3.4 COLLABORATION DIAGRAM

This collaborative network detection system tackles threats without relying on traditional sensors. Network devices, like routers and switches, act as actors, continuously exporting network flow data (source, destination, traffic type) to a central Security Information and Event Management (SIEM) system. The SIEM analyzes this collective data, searching for abnormalities and potential threats based on pre-defined rules or machine learning. If a threat is identified, the SIEM shares this intelligence (think threat signatures or suspicious IP addresses) with participating devices.

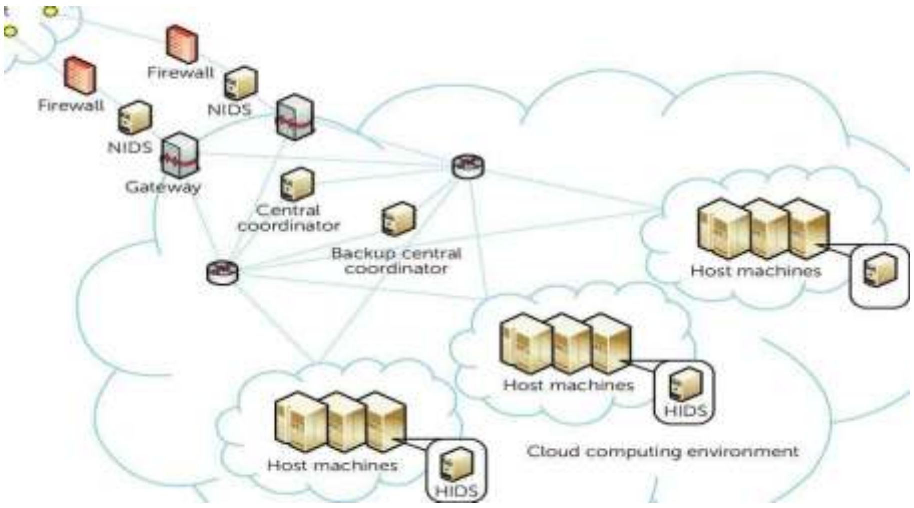


Fig 3.4:

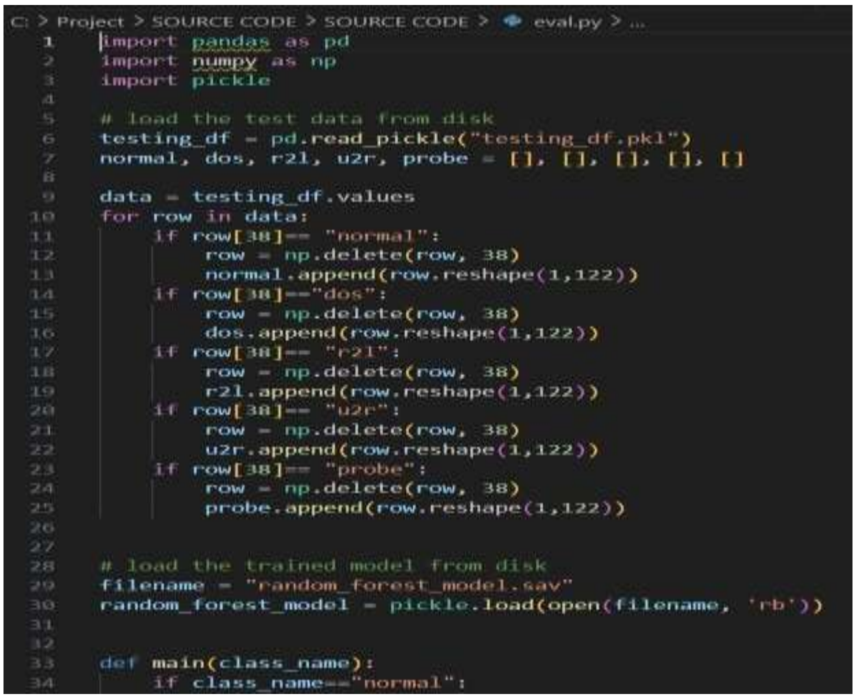
IV. MODULES LIST

Module 1: Creation of Random Forest model Module 2: Testing Model

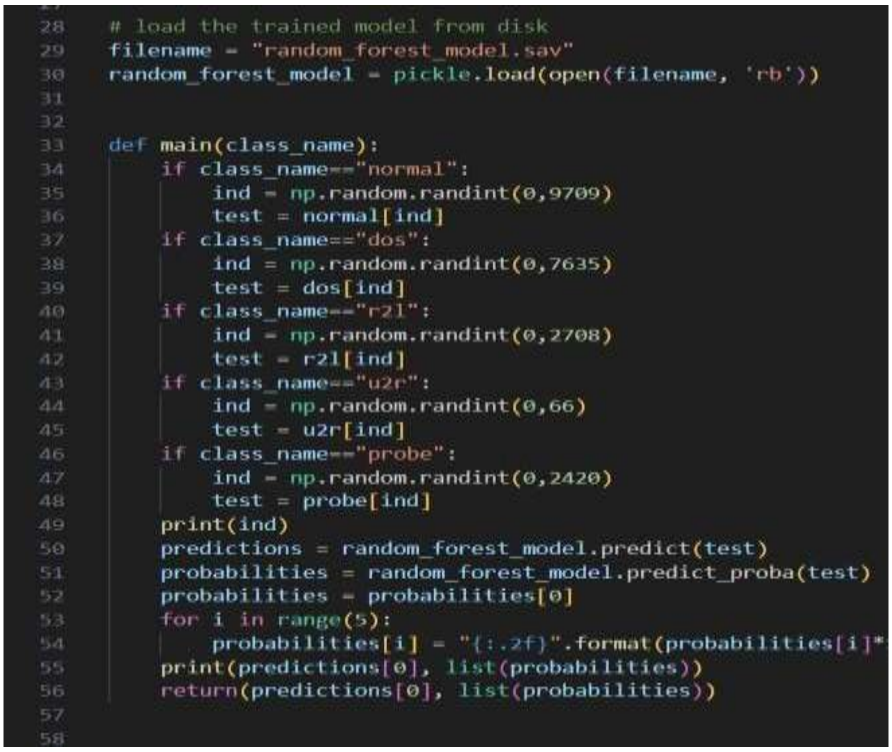
Module 3: Creating Flask Framework

4.1 MODULE DESCRIPTION

To build a Random Forest model for classification tasks, data collection and preprocessing should be conducted, ensuring that diverse and representative examples for each class are included. The data is then cleaned by removing duplicates and handling any missing values to create a reliable dataset. Following this, feature engineering is performed to identify important features that help distinguish between the different classes. The Random Forest model is trained on a split datasets for training and validation, and its performance is evaluated using metrics such as accuracy and precision. Finally, hyperparameter tuning techniques, such as grid search, are applied to identify the optimal parameters, maximizing the model's performance.


C: > Project > SOURCE CODE > SOURCE CODE > eval.py > ...

```
import pandas as pd
import numpy as np
import pickle

# load the test data from disk
testing_df = pd.read_pickle("testing_df.pkl")
normal, dos, r2l, u2r, probe = [], [], [], [], []

data = testing_df.values
for row in data:
    if row[38]== "normal":
        row = np.delete(row, 38)
        normal.append(row.reshape(1,122))
    if row[38]=="dos":
        row = np.delete(row, 38)
        dos.append(row.reshape(1,122))
    if row[38]== "r2l":
        row = np.delete(row, 38)
        r2l.append(row.reshape(1,122))
    if row[38]== "u2r":
        row = np.delete(row, 38)
        u2r.append(row.reshape(1,122))
    if row[38]== "probe":
        row = np.delete(row, 38)
        probe.append(row.reshape(1,122))


# load the trained model from disk
filename = "random_forest_model.sav"
random_forest_model = pickle.load(open(filename, 'rb'))


def main(class_name):
    if class_name=="normal":
```

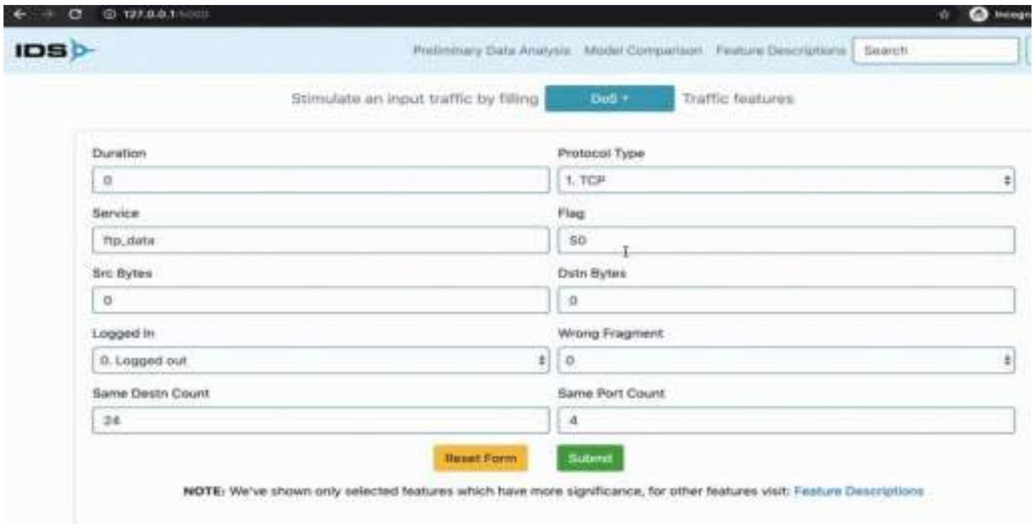


```
# load the trained model from disk
filename = "random_forest_model.sav"
random_forest_model = pickle.load(open(filename, 'rb'))


def main(class_name):
    if class_name=="normal":
        ind = np.random.randint(0,9709)
        test = normal[ind]
    if class_name=="dos":
        ind = np.random.randint(0,7635)
        test = dos[ind]
    if class_name=="r2l":
        ind = np.random.randint(0,2708)
        test = r2l[ind]
    if class_name=="u2r":
        ind = np.random.randint(0,66)
        test = u2r[ind]
    if class_name=="probe":
        ind = np.random.randint(0,2420)
        test = probe[ind]
    print(ind)
    predictions = random_forest_model.predict(test)
    probabilities = random_forest_model.predict_proba(test)
    probabilities = probabilities[0]
    for i in range(5):
        probabilities[i] = "{:.2f}".format(probabilities[i]*
    print(predictions[0], list(probabilities))
    return(predictions[0], list(probabilities))
```

Fig 4.1: Training Model (Sample Code)

4.2.1 Testing Model

Testing a random forest model in a network intrusion detection system (NIDS) using machine learning is a prudent approach. To develop an effective network intrusion detection system, begin by collecting and preprocessing labeled network traffic data, which includes cleaning, transforming, and normalizing features to create a consistent dataset. Next, perform feature selection to identify the most relevant attributes that differentiate between normal and malicious traffic, a process that reduces dimensionality and enhances model efficiency. Following this, train a random forest classifier on the processed data, using evaluation metrics such as accuracy, precision, recall, F1-score, and ROC curve analysis to measure its performance. Hyperparameter tuning is then essential to refine the random forest model, with techniques like grid search, random search, or Bayesian optimization employed to achieve optimal results. Finally, test the model on a separate dataset to validate its accuracy and reliability in real-world scenarios. Once validated, deploy the model in a live environment and establish a monitoring system for continuous updates and improvements, ensuring sustained effectiveness in detecting network intrusions.

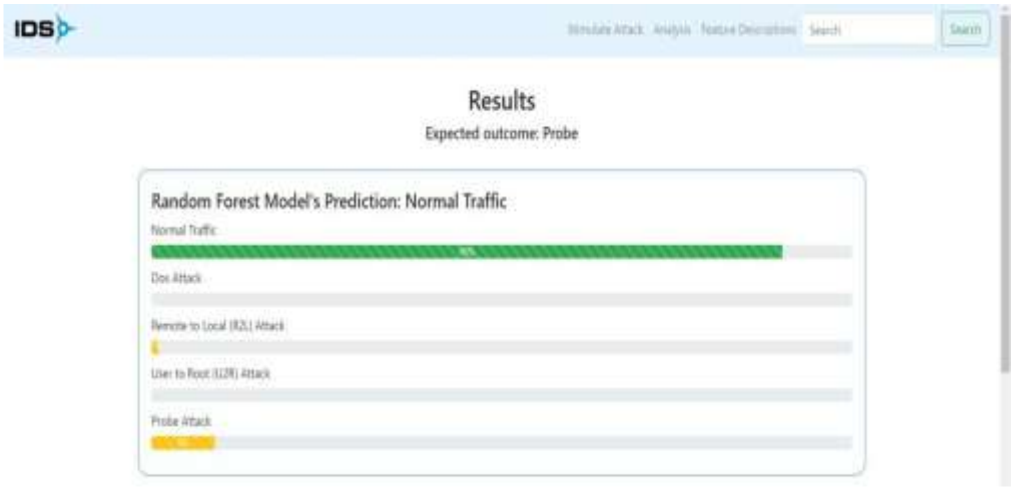


Stimulate an input traffic by filling DoS Traffic features

Duration: 0	Protocol Type: 1. TCP
Service: ftp_data	Flag: S0
Src Bytes: 0	Dstn Bytes: 0
Logged In: 0. Logged out	Wrong Fragment: 0
Same Destn Count: 24	Same Port Count: 4

NOTE: We've shown only selected features which have more significance, for other features visit: Feature Descriptions

Fig 4.2.1:



Results

Expected outcome: Probe

Random Forest Model's Prediction: Normal Traffic

Normal Traffic

Dos Attack

Remote to Local (R2L) Attack

User to Root (U2R) Attack

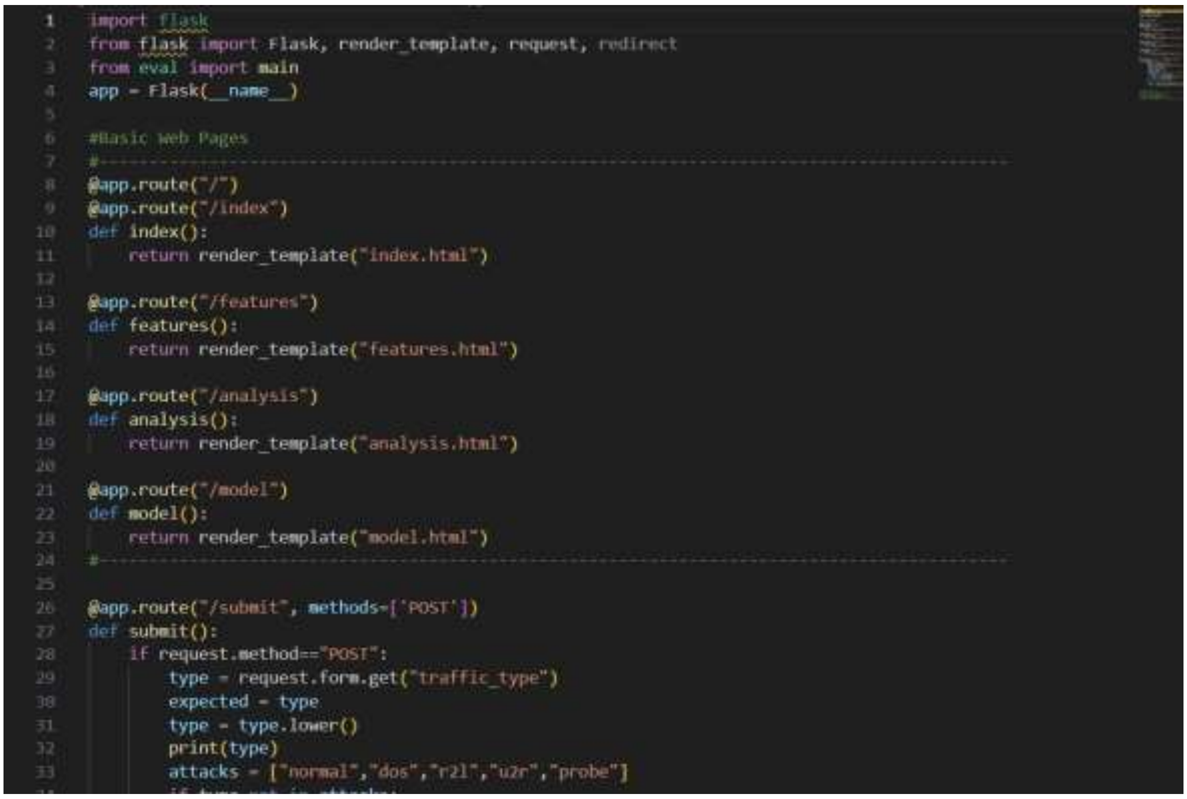
Probe Attack

Fig 4.2.2:

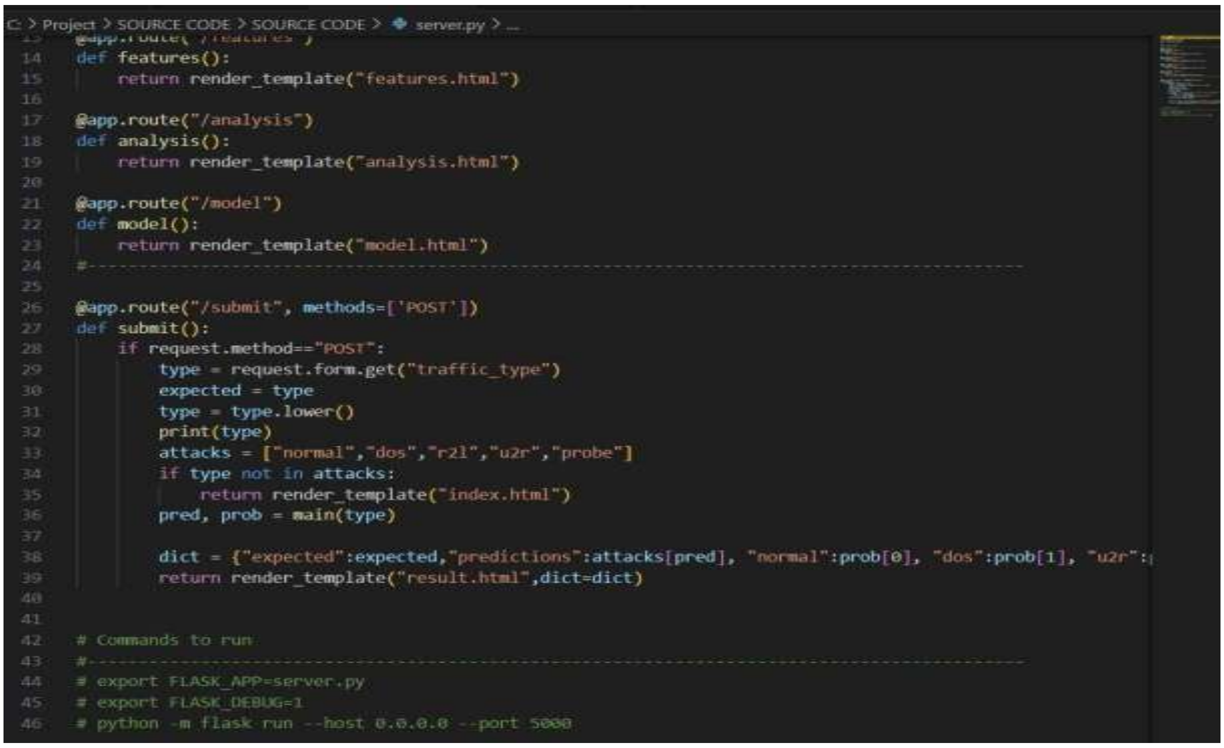
4.2.2 Testing Model

4.2.2 Creating Flask Framework:

Creating a Flask framework involves setting up a lightweight web application using Python's Flask framework. Here's how you can do it in a few points:To set up a Flask application, start by installing Flask through the Anaconda Prompt using the command conda install -c anaconda flask. Once Flask is installed, initialize your application by creating a Python script (e.g., app.py) and setting up a basic Flask app using Flask(_name_). Next, define routes in the application by using the @app.route('<url>') decorator, which maps a specific URL pattern to a corresponding function that handles incoming requests. After setting up your routes, navigate to your project directory in the Anaconda Prompt, and launch the Flask application with python app.py. You can then access the running application by opening a web browser and visiting the specified URL.



```
import flask
from flask import Flask, render_template, request, redirect
from eval import main
app = Flask(__name__)

#Basic Web Pages
#-----------------------------------------------------------------------------
@app.route("/")
@app.route("/index")
def index():
    return render_template("index.html")

@app.route("/features")
def features():
    return render_template("features.html")

@app.route("/analysis")
def analysis():
    return render_template("analysis.html")

@app.route("/model")
def model():
    return render_template("model.html")
#-----------------------------------------------------------------------------

@app.route("/submit", methods=['POST'])
def submit():
    if request.method=="POST":
        type = request.form.get("traffic_type")
        expected = type
        type = type.lower()
        print(type)
        attacks = ["normal","dos","r2l","u2r","probe"]
```



```
C: > Project > SOURCE CODE > SOURCE CODE > server.py > ...
@app.route("/features")
def features():
    return render_template("features.html")

@app.route("/analysis")
def analysis():
    return render_template("analysis.html")

@app.route("/model")
def model():
    return render_template("model.html")
#-----------------------------------------------------------------------------------------------

@app.route("/submit", methods=['POST'])
def submit():
    if request.method=="POST":
        type = request.form.get("traffic_type")
        expected = type
        type = type.lower()
        print(type)
        attacks = ["normal","dos","r2l","u2r","probe"]
        if type not in attacks:
            return render_template("index.html")
        pred, prob = main(type)

        dict = {"expected":expected,"predictions":attacks[pred], "normal":prob[0], "dos":prob[1], "u2r":
        return render_template("result.html",dict=dict)


# Commands to run
#-----------------------------------------------------------------------------------------------
# export FLASK_APP=server.py
# export FLASK_DEBUG=1
# python -m flask run --host 0.0.0.0 --port 5000
```

Fig 4.2.3: Creating Flask Framework

V. RESULTS AND DISCUSSION

System Testing

System testing aims to find errors and ensure that a product functions correctly by testing each component, sub-assembly, and the final product. It verifies that the system meets user expectations and performs without unacceptable failures. Testing types vary, with each addressing specific requirements.

Types of Tests

Unit Testing

Unit testing involves creating test cases to verify that internal program logic operates correctly, and inputs produce expected outputs. This test type is applied to each software component separately after completion, validating code flow, decision branches, and business process paths in line with defined specifications. It is an intrusive structural test, depending on knowledge of the program's construction.

Integration Testing

Integration testing checks if combined software components work seamlessly as a single system. This form of testing evaluates the results from integrated fields or screens, ensuring that while unit testing shows individual component success, integration tests confirm they function together effectively. Its focus is on identifying issues arising from component interactions.

Functional Testing

Functional testing systematically verifies that software features work as specified in technical and business requirements, system documentation, and user manuals. This testing approach includes checking for valid inputs by approving recognized input types and rejecting those deemed invalid. It ensures that all designated functions operate correctly and that expected outputs are generated as defined. Additionally, functional testing activates systems or procedures that interact with each other, confirming that they perform in alignment with the intended design and requirements.

System Testing

System testing validates that the integrated software system fulfills all requirements, focusing on ensuring consistent and predictable outcomes within the configuration. It is based on process flows, concentrating on integration points and process-driven connections.

White Box Testing

White box testing is conducted with knowledge of the program's internal structure, logic, and language, focusing on areas not accessible through black box testing. This approach enables a deeper inspection of program operations and logic.

Black Box Testing

In black box testing, the tester examines software functionality without insight into its internal workings or structure. Tests are based on requirements or specifications, treating the software as a "black box" by focusing solely on inputs and outputs, independent of the underlying code.

VI. CONCLUSION

The first step in the preprocessing procedure involves cleaning, labeling, and classifying the network traffic data set according to several attack types, such as DOS, R2L, U2R, Probe, and Normal. To store a basic feature set for later usage, the outcome and difficulty columns are removed. Now that the data has been prepared, a Random Forest model that has already been trained is imported into a software to forecast the type of traffic attack. In addition to predicting the traffic class, it also gives the likelihood of every potential attack type. This facilitates effective testing and validation of cybersecurity apps using a machine learning model. By combining data preparation, model loading, and prediction, the pipeline demonstrates how to methodically categorize network invasions and has proven to be highly valuable in identifying malicious activity in real-world network environments.

6.1 FUTURE ENHANCEMENTS

To enhance intrusion detection systems (IDS), advanced machine learning (ML) algorithms should be implemented to improve detection accuracy by effectively identifying subtle and evolving threats. Developing ML models capable of real-time detection will allow for immediate responses to security threats as they occur. Additionally, utilizing ML techniques for anomaly detection can help identify deviations from normal network behavior, enabling the detection of previously unknown threats. Integrating automated response mechanisms driven by ML decision-making can streamline actions such as blocking suspicious IP addresses or isolating compromised hosts. Furthermore, implementing adaptive learning in ML models ensures they can learn from new data and emerging threats, maintaining effectiveness in dynamic environments. Reducing false positives is another critical goal, achieved through ML algorithms that accurately distinguish between genuine threats and benign network anomalies. Employing behavioral analysis techniques based on ML will help profile user and network behaviors, improving the detection of insider threats and sophisticated attack patterns. Finally, integrating ML-driven IDS with threat intelligence feeds will enhance overall threat detection capabilities and keep systems updated on emerging cyber threats.

VII. REFERENCES

[1] Y.-a. Huang, W. Fan, W. Lee, and P. S. Yu, "Cross-feature analysis for detecting ad-hoc routing anomalies," in Proc. 23rd IEEE International Conference on Distributed Computing Systems, 2021, pp. 478–487

[2] S. Kaplantzis, A. Shilton, N. Mani, and Y. A. S¸ekercioglu, "Detecting selective forwarding attacks in wireless sensor networks using support vector machines," in 3rd IEEE International Conference on Intelligent Sensors, Sensor Networks and Information, 2021, pp. 335–340

[3] S. Kurosawa, H. Nakayama, N. Kato, A. Jamalipour, and Y. Nemoto, "Detecting blackhole attack on aodv-based mobile ad hoc networks by dynamic learning method." IJ Network Security, vol. 5, no. 3, pp. 338–346, 2020.

[4] M. Kim, I. Doh, and K. Chae, "Denial-of-service (dos) detection through practical entropy estimation on hierarchical sensor networks," in Advanced Communication Technology (ICACT), 2020.

[5] Halimaa, A., & Sundarakantham, K. (2019, April). Machine learning based intrusion detection system. In 2019 3rd International conference on trends in electronics and informatics (ICOEI) (pp. 916-920). IEEE.

[6] Dong, Y., Wang, R., & He, J. (2019, October). Real-time network intrusion detection system based on

deep learning. In 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS) (pp. 1-4). IEEE.

[7] Hijazi, A., El Safadi, A., & Flaus, J. M. (2018, December). A Deep Learning Approach for Intrusion Detection System in Industry Network. In BDCSIntell (pp. 55-62)

[8] Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017, September). Evaluation of machine learning algorithms for intrusion detection system. In 2017 IEEE 15th international symposium on intelligent systems and informatics (SISY) (pp. 000277-000282). IEEE.

[9] R. Hofstede et al. provided an explanation of flow monitoring, from packet capture to data analysis using NetFlow and IPFIX, in their paper published in IEEE Communications Surveys & Tutorials in 2014.

[10] Hijazi, A., El Safadi, A., & Flaus, J. M. (2018, December). A Deep Learning Approach for Intrusion Detection System in Industry Network. In BDCSIntell (pp. 55-62)

[11] Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017, September). Evaluation of machine learning algorithms for intrusion detection system. In 2017 IEEE 15th international symposium on intelligent systems and informatics (SISY) (pp. 000277-000282). IEEE.

[12] R. Hofstede et al. provided an explanation of flow monitoring, from packet capture to data analysis using NetFlow and IPFIX, in their paper published in IEEE Communications Surveys & Tutorials in 2014.

[13] Hijazi, A., El Safadi, A., & Flaus, J. M. (2018, December). A Deep Learning Approach for Intrusion Detection System in Industry Network. In BDCSIntell (pp. 55-62)

[14] Almseidin, M., Alzubi, M., Kovacs, S., & Alkasassbeh, M. (2017, September). Evaluation of machine learning algorithms for intrusion detection system. In 2017 IEEE 15th international symposium on intelligent systems and informatics (SISY) (pp. 000277-000282). IEEE.

[15] R. Hofstede et al. provided an explanation of flow monitoring, from packet capture to data analysis using NetFlow and IPFIX, in their paper published in IEEE Communications Surveys & Tutorials in 2014.