Anomaly Detection and Intrusion Prevention Using ML in Python

[A] Nishant Kumar Singh

M.Tech Student, Department of Computer Science and Engineering All Saints College of Technology, Bhopal, India
Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya (RGPV) nishantkumarsingh1912@gmail.com

[B] Prof. Sarwesh Site

Associate Professor, Department of Computer Science and Engineering All Saints College of Technology, Bhopal, India
Affiliated to Rajiv Gandhi Proudyogiki Vishwavidyalaya (RGPV) er.sarwesh@gmail.com

ABSTRACT

This project focuses on developing an intrusion detection system (IDS) using ML techniques. An IDS is a software tool designed to continuously monitor network traffic to identify malicious activities or violations of security policies. In this study, Python has been utilized to implement the ML models and configure the detection system based on specific design requirements. There are various types of IDS, such as network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS), each serving distinct purposes in enhancing cybersecurity. The system is also capable of identifying anomalies, thereby helping in understanding and mitigating hacker activities.

Multiple sensors have been integrated within the IDS to generate and track security events effectively. Additionally, a control console has been used to manage the detection process, which is essential for implementing robust security protocols. A thorough literature review has been conducted, including summaries and research questions, to support this work. The methodology section of the project covers the philosophical assumptions, research questions, and considerations for ensuring the validity and reliability of the study.

The methods for data collection and selection have been explained in detail, providing a comprehensive approach to gathering the necessary information. Limitations of the project have also been discussed to present a realistic perspective. The dataset used for this work was sourced from a reliable website and applied to evaluate the accuracy of intrusion detection models. The results section presents the outcomes along with screenshots that highlight the performance of the system.

This research was implemented using Python in the Jupyter Notebook environment. The artefact's development process has been clearly described, outlining its purpose and how it contributes to the overall goal of improving intrusion detection. The methodology further explores how the accuracy score was computed, specifically using a RF classifier, which proved effective in predicting and identifying intrusions.

Finally, recommendations have been provided to offer insights and suggestions for enhancing the system's efficiency.

This project not only demonstrates the use of ML for intrusion detection but also serves as a guide for future *improvements in network security.*

Keywords — Intrusion Detection System, ML, Python, Network Security, Anomaly Detection, RF Classifier, Accuracy Score, Jupyter Notebook, Data Collection, Data Selection, Host-based IDS, Network-based IDS, Cybersecurity, Security Events, Threat Detection, Sensors, Control Console, Validity, Reliability, Literature Review, Security Protocols.

1 Introduction

1.1 Background

The main goal of this dissertation is to explore the development of a network intrusion detector using ML techniques implemented in Python. Intrusion detection systems (IDS) are specialized software solutions designed to identify unauthorized or malicious activity within a network by leveraging various ML methods. Sensors play a critical role in collecting and processing network data while ensuring optimal performance for monitoring purposes. IDS are capable of identifying different types of attacks, such as denial of service, unauthorized remote access, probing attempts, and local privilege escalations. These systems enhance network security by providing better visibility and aiding compliance with security policies through detailed logging. Moreover, IDS need to operate discreetly to prevent detection by attackers while securing sensitive network assets.

1.2 Rationale

The primary function of an intrusion detection system is to recognize malicious network traffic that cannot be detected by traditional firewall solutions. This is crucial for ensuring the confidentiality, integrity, and availability of organizational data and systems. IDS are broadly classified into two categories: Anomaly-based IDS (AIDS) and Signature-based IDS (SIDS). Anomaly-based systems identify unusual patterns of behavior, while signature-based systems rely on known attack patterns for detection. Intrusions pose significant threats to organizations, leading to financial losses and compromised data security. As a result, IDS are increasingly being deployed by organizations, with the assistance of developers, to detect malicious activities, perform data mining, and mitigate cyber threats. These systems play a vital role in preserving system integrity and safeguarding sensitive information from unauthorized access or exploitation.

1.3 Case Study

This dissertation focuses on identifying network intrusions by applying various ML techniques, analyzing threat patterns, and employing classification algorithms. The approach involves creating a database containing known intrusion signatures, which are then compared against incoming activities to detect suspicious behavior. When a match is found, an alert is generated to inform security personnel of potential threats. These alerts enable further investigation to identify the source of the attack and prevent additional damage. The IDS framework offers significant benefits by assisting IT teams in responding promptly to threats, thereby protecting organizational assets and data privacy. The system is capable of monitoring both inbound and outbound traffic, ensuring comprehensive protection against potential network attacks.

1.4 Objectives and Research Questions

Objectives

- To examine the key factors influencing the detection of intrusions within an organization’s network.
- To apply ML techniques for building an effective intrusion detection system.
- To utilize Python programming for identifying malicious activities in datasets.
- To evaluate the effectiveness of IDS in detecting intrusions and its impact on organizational security.

Research Questions

1. What factors contribute to the successful detection and identification of network intrusions?
2. What methodologies are used to develop IDS that can be employed in organizational settings?
3. What Python-based techniques are applied for identifying threats from datasets?
4. What are the outcomes achieved by using IDS in organizations?

1.5 Dissertation Structure

This dissertation begins with an introduction to the importance, purpose, and background of intrusion detection systems in organizational environments. It presents research objectives and questions designed to guide the study. The literature review section discusses both primary and secondary data sources used to understand IDS, including their functions, applications, and advancements in ML-based detection techniques. The methodology section outlines the research methods, data collection strategies, and philosophical assumptions that support the study. The dissertation further explores how IDS contribute to enhanced data protection and organizational security, supported by findings and relevant industry perspectives.

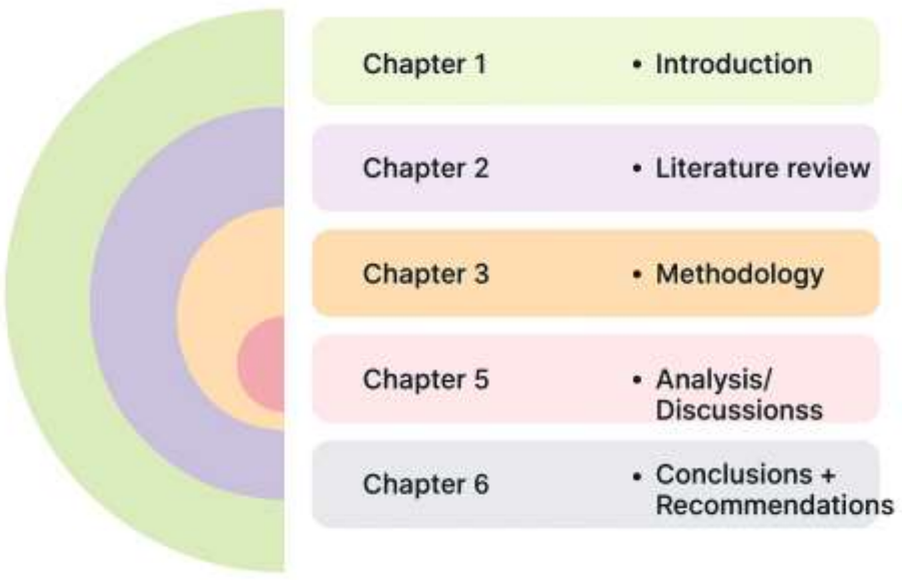


Figure 1.5.1: Dissertation structure

The selected dataset will be used for detecting intrusions and identifying potential limitations. ML techniques will be applied to analyze the dataset, with results presented through screenshots and artefact descriptions. The findings will be discussed to highlight the effectiveness of the intrusion detection process.

2. LITERATURE REVIEW

2.1 Overview

No network can be considered completely secure, and no firewall can guarantee absolute protection. Therefore, organizations must adopt robust measures to safeguard their systems and prevent unauthorized access or attacks. Cyber attackers continuously evolve their methods, leveraging social engineering tactics and malware to illegally access sensitive information and exploit vulnerabilities in network defenses. This highlights the importance of implementing effective intrusion detection systems (IDS) that help monitor, identify, and respond to malicious network traffic, ensuring the integrity and security of organizational data.

With the ever-increasing volume of network traffic, many existing IDS solutions struggle to efficiently analyze and process all incoming data (Gupta and Singh, 2017). Early detection of potential threats allows IT teams to manage attacks in a timely and controlled manner, minimizing potential damage. IDS management plays a critical role in maintaining security by continuously monitoring network activity and generating alerts for suspicious behavior.

However, monitoring network traffic can lead to both false positives and false negatives, where benign activities are mistakenly flagged as threats or actual intrusions go undetected. It is essential for IT professionals to possess

the right expertise and technologies to distinguish between legitimate and malicious actions effectively. Signature-based IDS solutions offer accurate detection of known threats but may occasionally misclassify normal activities as harmful. To address this, IDS systems must be built upon well-defined baselines of normal network behavior and be carefully tuned to reduce false alerts while allowing legitimate traffic to pass through.

2.2 Topics

2.2.1 Working and Types of IDS

Intrusion detection systems function by identifying anomalies in network behavior to prevent attackers from causing significant harm. IDS solutions can be deployed either at the host level or across the network, depending on the approach used. Host-based IDS (HIDS) are installed on individual computers, while network-based IDS (NIDS) monitor traffic flowing through the network infrastructure.

These systems examine network traffic and look for known patterns or signatures associated with attacks. Suspicious activities are flagged and further analyzed at the application or protocol layer for deeper inspection. ML algorithms are often integrated into IDS to improve detection accuracy by comparing current network behavior with known attack signatures (Nerlikar et al., 2020). This helps in identifying events such as DNS poisoning or specific scanning techniques like the Christmas tree scan.

IDS can be implemented as software applications or integrated into specialized hardware solutions. Based on their detection mechanisms and deployment methods, IDS are classified into several types, including Network-based IDS (NIDS), Host-based IDS (HIDS), Signature-based IDS (SIDS), and Anomaly-based IDS (AIDS). Furthermore, IDS can be categorized as either active or passive systems depending on whether they automatically respond to threats or simply alert administrators for further action.

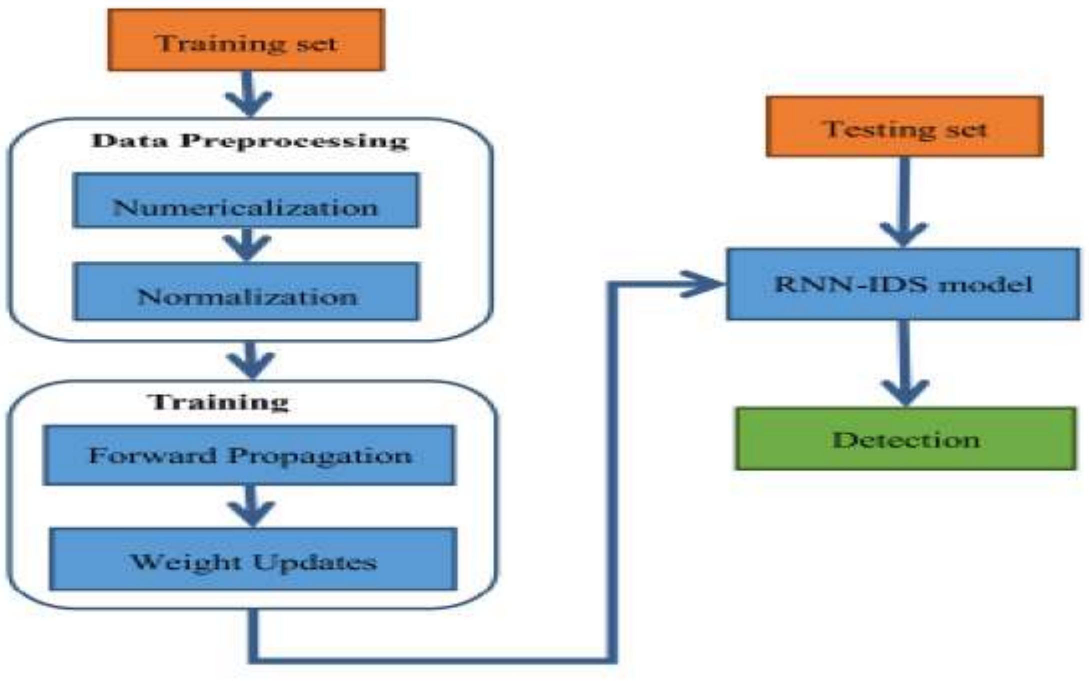


Figure 2.2.1.1: IDS Functional Approaches

Intrusion Detection Systems (IDS) use machine learning methods like support vector machines for supervised threat detection (Kavitha et al., 2021). Network-based IDS (NIDS) monitor traffic at key points in a network, while Host-based IDS (HIDS) work on individual devices. Signature-based IDS (SIDS) compare packets with known attack signatures, whereas Anomaly-based IDS (AIDS) flag deviations from normal network behavior such as bandwidth or port usage.

2.2.2 Operational Value of Intrusion Detection Systems

Intrusion detection systems (IDS) offer advanced monitoring of firewalls, routers, servers, and critical files to help detect, prevent, and recover from cyber threats. They assist administrators in organizing audit trails and logs, making it easier to track suspicious activities. IDS solutions often feature user-friendly interfaces, allowing even non-technical staff to access relevant information for security checks. By identifying changes or anomalies in data files, IDS can quickly alert security teams of potential breaches. Once an intrusion is detected, the system can block access from compromised servers to prevent further attacks. Overall, IDS enhance an organization’s ability to identify and respond to security incidents effectively.

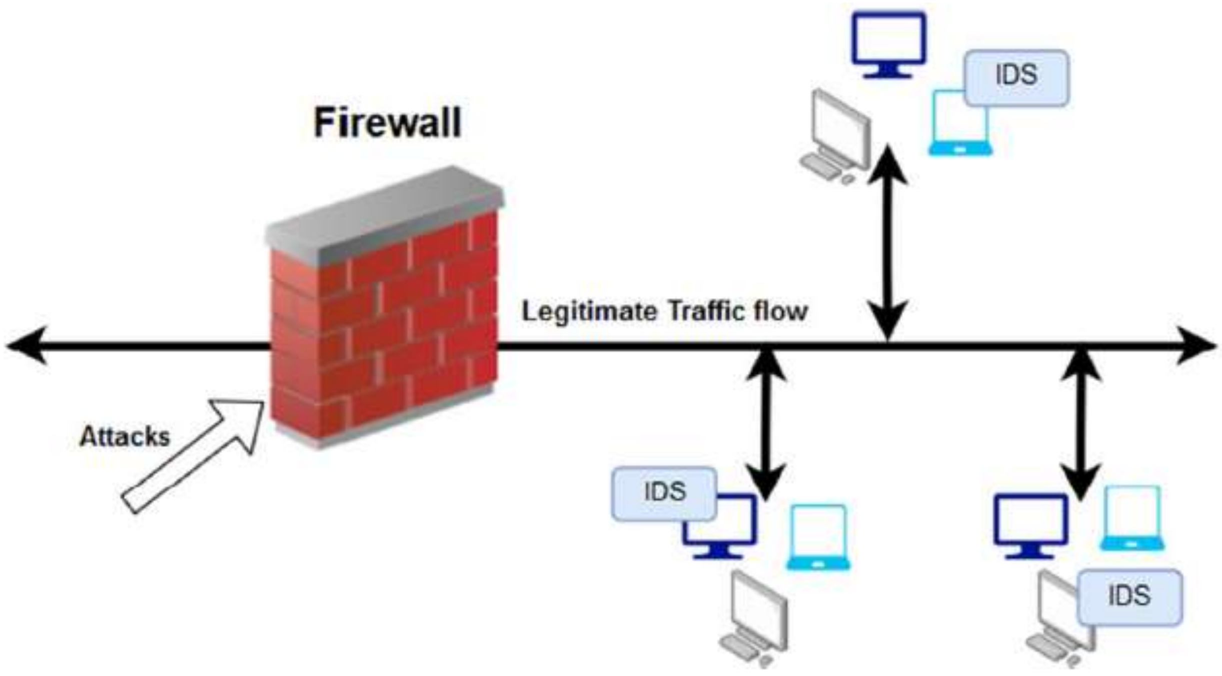


Figure 2.2.2.1: Design Philosophy for Host IDS

(Source: Adapted from Kasongo and Sun, 2020)

Intrusion detection systems (IDS) analyze the type and frequency of attacks to provide valuable insights into ongoing threats. These systems play a vital role in monitoring network activities and protecting organizations from external risks. IDS metrics help in evaluating future risks and ensuring compliance with security standards. By offering improved visibility into network traffic, IDS assist organizations in maintaining security protocols and enhancing response strategies. Additionally, they aid in identifying compromised devices and monitoring data packets, making it easier to detect threats. The automated collection of information through IDS also supports manual inspection efforts, making threat detection more efficient.

2.2.3 Challenges of IDS

Intrusion detection systems can sometimes generate false alarms, requiring organizations to regularly adjust and fine-tune them after initial deployment. Accurate configuration is necessary to distinguish between normal traffic and suspicious activities. Although false positives can be inconvenient, they are generally less harmful than false negatives, where real threats go undetected. With the rapid growth of internet applications, IDS need constant updates and improvements using advanced statistical and data mining techniques. One major challenge is that false negatives may allow attackers to go unnoticed, leaving networks vulnerable until significant damage occurs. Therefore, continuous tuning and vigilance are essential for maintaining IDS effectiveness (Imran and Jamil, 2021).

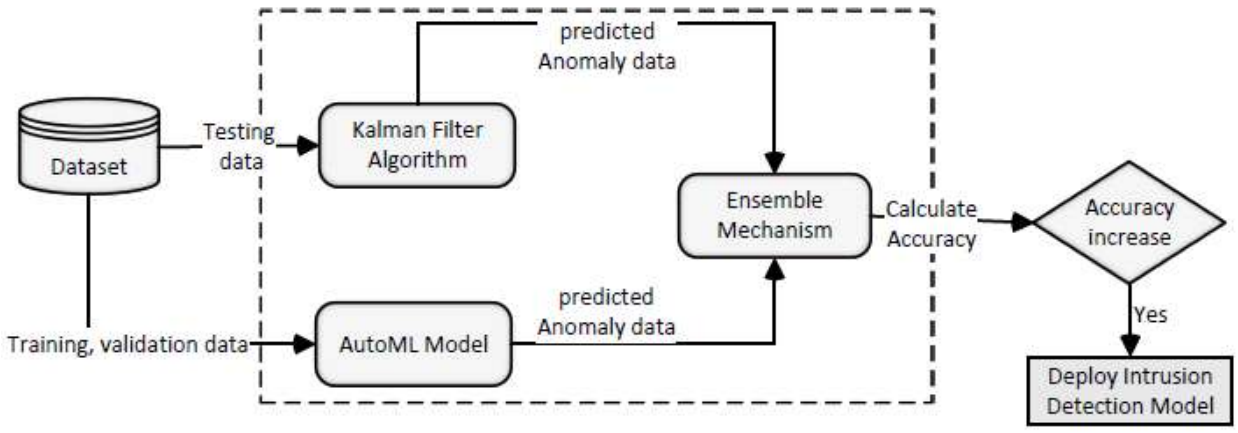


Figure 2.2.3.1: Model for Anomaly Detection

(Source: Adapted from Imran and Jamil, 2021)
False negatives remain a major challenge for IDS as malware becomes more advanced and harder to detect. New threats often bypass existing detection methods, reducing the effectiveness of IDS. Despite this, IDS are essential tools for identifying malicious activities by analyzing network traffic and host logs (Maia, 2021). The growing complexity of cyber threats has increased the demand for IDS that can quickly detect novel attack patterns using advanced techniques. Even with these challenges, IDS remain a reliable approach to protecting organizations from evolving threats.

2.3 Summary

IDS use hardware and software monitoring to analyze network data and detect attacks. Traditional systems are complex and slow, making them vulnerable during prolonged analysis periods. Modern IDS leverage big data technologies to process large datasets more efficiently, though delays can still leave systems exposed. The rise in cyberattacks has driven the need for advanced IDS solutions to prevent data breaches and reduce organizational losses (Thi-Thu-Huong Le, Kim, and Kim, 2019). IDS are vital for identifying anomalies and mitigating threats.

Three primary IDS methods are hybrid, anomaly-based, and signature-based detection. Signature-based IDS match known threats using a preloaded database but struggle with new attacks if not regularly updated. Anomaly-based systems compare user activity against predefined profiles, offering better protection against unknown threats but prone to false alarms. Combining methods in hybrid systems helps balance strengths and weaknesses, with ML algorithms enhancing detection and classification.

2.4 Questions

1. How effective are IDS in preventing malicious activities within networks?
2. Why are organizations increasingly adopting IDS for detecting fraud?
3. What vulnerabilities in IDS can be exploited by attackers?
4. What is the future of IDS in safeguarding networks and servers from threats?

3. Methodology

3.1 Introduction

This dissertation focuses on developing an intrusion detection system (IDS) using ML techniques to analyze datasets containing network intrusion information. Intrusions can include activities such as infiltration, denial of service, or brute force attacks. Since attack patterns change over time, selecting a comprehensive dataset is essential for accurate analysis. The dataset is used to build a deep learning model that identifies anomalies and helps detect suspicious behavior. ML helps by finding specific patterns in data and comparing them to known threats. After processing and training the dataset, the model's accuracy is evaluated to ensure reliable intrusion detection.

3.2 Philosophical Assumptions

Intrusion detection relies on host-based, network-based, or application-based monitoring. ML models compare newly collected data with previously recorded data to identify potential intrusions. Common models in Python include logistic regression, gradient boosting, and RF. This study uses the RF classifier due to its ability to handle both categorical and continuous data, as well as its ease of use and accurate predictions. Before modeling, the dataset is cleaned to remove null values and irrelevant columns, ensuring better performance. The processed data helps detect network threats and strengthen overall security.

3.3 Research Questions

1. What methods are used to check and clean the dataset?

Datasets are checked for missing or incorrect values, which are either replaced or removed to ensure accurate results. Firewalls alone are not enough; IDS adds another layer of protection.

2. What models are used for IDS creation?

The RF classifier is widely used for both categorical and continuous data. Decision tree classifiers are also applied to calculate accuracy.

3. What limitations exist in ML techniques for IDS?

Large datasets and updated data are necessary for accurate predictions. Even after cleaning, errors may persist, requiring careful monitoring.

4. How can IDS results help prevent attacks?

Predictions from IDS alert IT teams to possible threats, allowing timely action to prevent major damage. The findings can also guide other organizations.

3.4 Validity and Reliability

ML advancements have increased the use of IDS but concerns over privacy and security remain. Accurate decision-making models, like decision trees, require reliable networks and good-quality data. Pre-processing steps, such as entropy-based feature selection, enhance training and improve detection rates. IDS aim to protect data's confidentiality, integrity, and availability by implementing security policies and using hybrid technologies to adapt to new threats.

3.5 Data Selection and Collection

Choosing a high-quality dataset is critical for IDS performance. The dataset used in this study is sourced from Kaggle and includes both training and test datasets with various features such as duration, protocol type, service, and failed login attempts. It contains real intrusion data in formats like string, integer, and float. These datasets allow the model to predict threats effectively without requiring complex splitting functions. The data helps evaluate detection accuracy and improve algorithms, complementing firewall protections.

3.6 Ethics and Bias

ML algorithms can produce biased results if used with improper or incomplete datasets. This study uses publicly available datasets from reliable sources like Kaggle, ensuring that ethical guidelines are followed. The dataset is used only for research purposes, and results are based on the selected data and applied methods. Developers and users are informed about the dataset's usage and limitations.

3.7 Limitations

IDS often detect false alarms, which can obscure real threats. Real attacks may go unnoticed due to noise from false positives. Regular system updates and bug fixes are necessary to maintain detection accuracy. IDS are also vulnerable to protocol-based attacks since they monitor the same network they protect. While IDS help identify anomalies, they may not cover all attack vectors, and reliance on audit logs can expose systems to manipulation.

4. Results and Discussion

4.1 Introduction

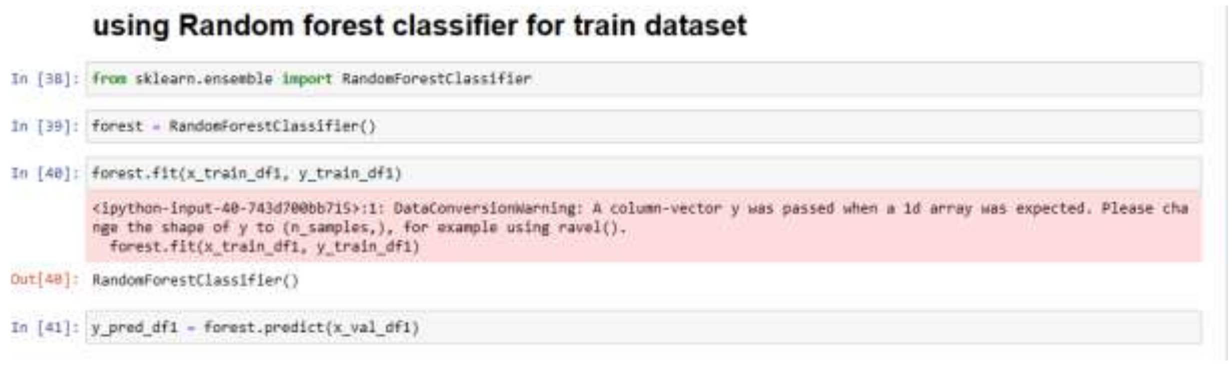
The (IDS) developed in this study plays a crucial role in network security and administration. When attackers attempt to steal or manipulate data, organizations face significant losses. The IDS uses ML algorithms to analyze a selected dataset and detect malicious activities. It provides real-time alerts and safeguards systems by identifying attacks before they cause damage. The IDS compares new traffic patterns with previously recorded data to detect threats like denial of service and brute force attacks. Accurate analysis enhances detection by improving understanding and prediction of attack behaviors. ML offers innovative solutions to tackle attacks such as memory overflow and DDoS, helping prevent data loss and reputational damage.

4.2 Response Rates

IDS works alongside firewalls to enhance cybersecurity by monitoring threats and providing alerts. While the IDS itself does not block attacks, firewalls prevent the detected intrusions. The system analyzes network metadata to control traffic based on predefined rules, filtering suspicious activities like Neptune and smurf attacks. IDS identifies known attack signatures and unusual traffic patterns, allowing IT teams to assess and respond to threats. Risk evaluations based on these alerts help organizations, especially those handling sensitive data like e-banking and e-commerce, respond quickly and accurately. However, IDS must be frequently updated to detect new attack methods and reduce false positives.

4.3 Results

The analysis of large datasets helped distinguish between normal and malicious network traffic. Active network ports functioned as sensors, continuously monitoring incoming traffic and establishing baseline behaviors. ML models focused on anomaly detection, filtering out irregular user behavior while highlighting threats. These models use decision-making systems to predict potential attacks, providing IT teams with the necessary tools to secure their networks. The results confirmed the system's ability to accurately identify and mitigate threats, improving overall network security.

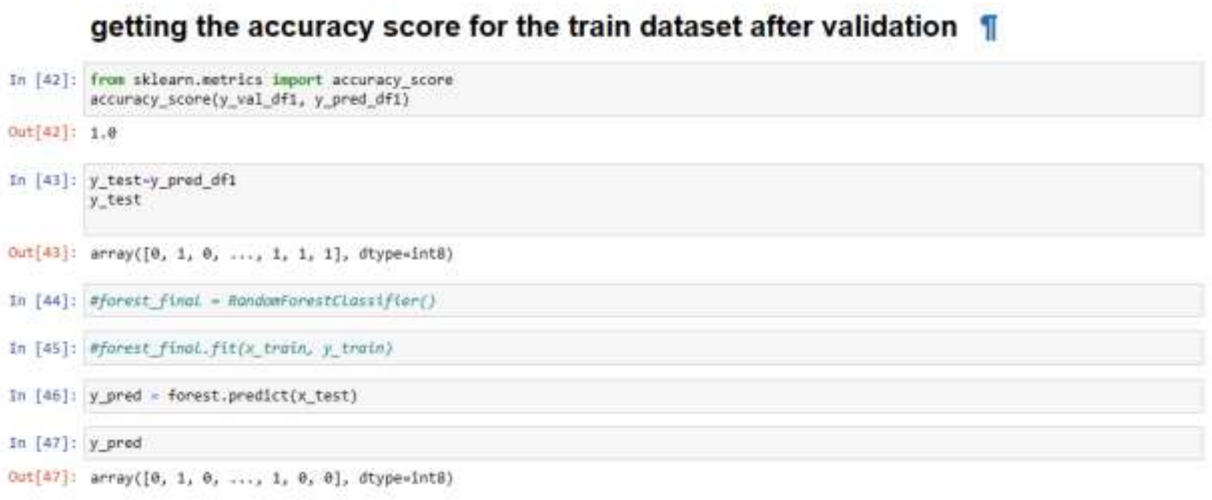


using Random forest classifier for train dataset

```
In [38]: from sklearn.ensemble import RandomForestClassifier

In [39]: forest = RandomForestClassifier()

In [40]: forest.fit(x_train_df1, y_train_df1)

         <ipython-input-40-743d700bb715>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
         nge the shape of y to (n_samples,), for example using ravel().
           forest.fit(x_train_df1, y_train_df1)

Out[40]: RandomForestClassifier()

In [41]: y_pred_df1 = forest.predict(x_val_df1)
```

Figure 4.3.1: RF Classifier

Monitoring network traffic is a key function of the intrusion detection system (IDS), which analyzes data from training and test datasets containing captured packets. Historical data is used to support real-time decision-making and identify unauthorized access attempts. ML models, such as RF and decision tree classifiers, are applied to extract critical details like source IP addresses, packet sizes, and attack vectors. Ports under attack, along with packet types such as TCP, UDP, or ICMP, are closely examined to identify malicious behavior. The results assist in recognizing user anomalies by leveraging network performance analysis (Khatri, Shrestha, and Nam, 2021). Additionally, deep learning techniques are employed to enhance anomaly detection by generating adaptive networks capable of identifying threats in real-time environments.

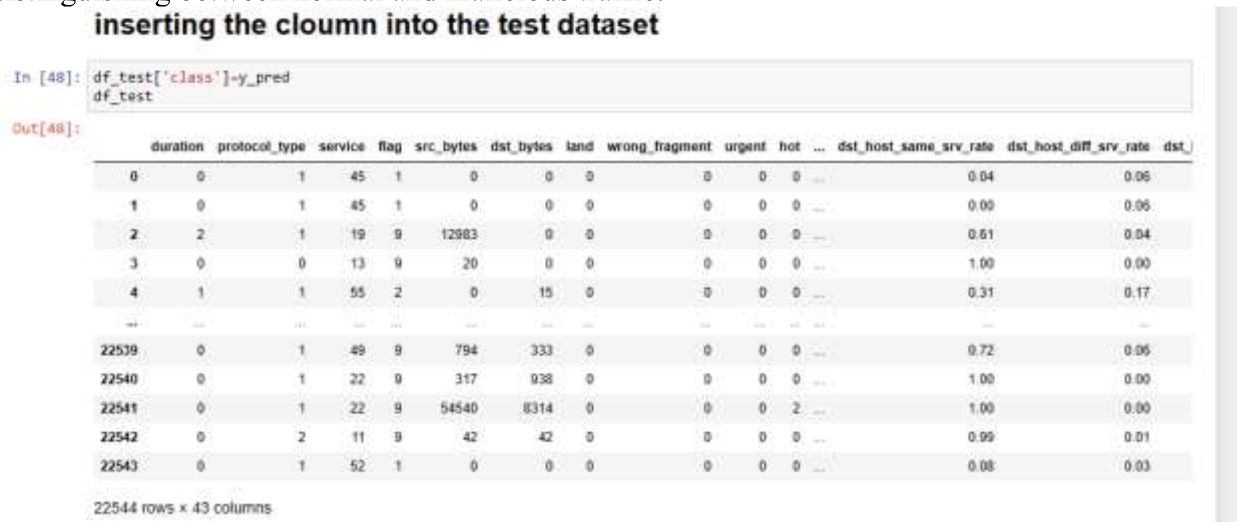


getting the accuracy score for the train dataset after validation ¶

```
In [42]: from sklearn.metrics import accuracy_score
         accuracy_score(y_val_df1, y_pred_df1)

Out[42]: 1.0

In [43]: y_test=y_pred_df1
         y_test

Out[43]: array([0, 1, 0, ..., 1, 1, 1], dtype=int8)

In [44]: #forest_final = RandomForestClassifier()

In [45]: #forest_final.fit(x_train, y_train)

In [46]: y_pred = forest.predict(x_test)

In [47]: y_pred

Out[47]: array([0, 1, 0, ..., 1, 0, 0], dtype=int8)
```

Figure 4.3.2: Performance Evaluation of Classification Models.

The activity patterns of anomalous users differ significantly from those of legitimate users, whose behavior does not typically endanger the network. However, malicious user activity can cause considerable harm. Implementing an appropriate framework allows the extraction of frequent patterns from network traffic, offering a clear representation of user behavior. In the dataset, class labels are included only during training and are absent in the test data. For evaluation purposes, the test dataset is assigned a class label column, generated through predictions obtained from the *y_pred* function. Within intrusion detection systems, suspicious behavior is identified by continuously monitoring alerts and traffic flow (Nerlikar et al., 2020). Accurate detection methods are therefore vital to maintaining network security, with machine learning models playing a key role in distinguishing between normal and malicious traffic.



inserting the cloumn into the test dataset

```
In [48]: df_test['class']=y_pred
         df_test
```

```
Out[48]:
       duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_same_srv_rate  dst_host_diff_srv_rate  dst_
     0        0              1       45     1          0          0     0               0       0    0  ...                    0.04                    0.06
     1        0              1       45     1          0          0     0               0       0    0  ...                    0.00                    0.06
     2        2              1       19     9      12983          0     0               0       0    0  ...                    0.61                    0.04
     3        0              0       13     9         20          0     0               0       0    0  ...                    1.00                    0.00
     4        1              1       55     2          0         15     0               0       0    0  ...                    0.31                    0.17
   ...      ...            ...      ...   ...        ...        ...   ...             ...     ...  ...  ...                     ...                     ...
 22539        0              1       49     9        794        333     0               0       0    0  ...                    0.72                    0.06
 22540        0              1       22     9        317        938     0               0       0    0  ...                    1.00                    0.00
 22541        0              1       22     9      54540       8314     0               0       0    2  ...                    1.00                    0.00
 22542        0              2       11     9         42         42     0               0       0    0  ...                    0.99                    0.01
 22543        0              1       52     1          0          0     0               0       0    0  ...                    0.08                    0.03

22544 rows × 43 columns
```

Figure 4.3.3: Insertion of column

4.4 Summary

Normal behaviors in the dataset are defined based on historical patterns of users, hosts, and network connections over specific time periods. Initially, intrusion detection systems (IDS) use signature-based methods to recognize known threats, while anomaly-based IDS focus on identifying unusual behaviors. Signature-based detection typically results in fewer false alarms compared to anomaly-based methods. Specific rules, known as signatures, help identify intrusions by analyzing unknown patterns and suspicious activities (Imran and Jamil, 2021). Benchmark datasets are used to test and improve the accuracy of these detection systems through systematic evaluation.

The raw data is processed to create baseline patterns by averaging packet flows over defined time scales. Variances are measured to understand normal network behavior, helping identify anomalies. ML algorithms such as RF and DT classifiers use this data to train models for accurate intrusion detection (Stern et al., 2021). These algorithms rely on labeled data to distinguish between legitimate and malicious traffic.

Additionally, statistical analysis of network traffic is performed by continuously monitoring active ports and sensors. Baseline behaviors are established by averaging packet activity over time, allowing for the detection of deviations that may indicate threats. These baselines and thresholds support efficient decision-making, enabling the IDS to respond quickly to potential intrusions by analyzing uncertainties and variations in network behavior.

Chapter 5: Final Deliverable

5.1 Overview

For this project, Jupyter Notebook was chosen as the primary platform to work with the dataset and analyze results for the intrusion detection system (IDS). Jupyter Notebook offers an interactive environment that allows computations to be executed within the browser, making it easier to run experiments, test prediction models, and measure accuracy scores efficiently.

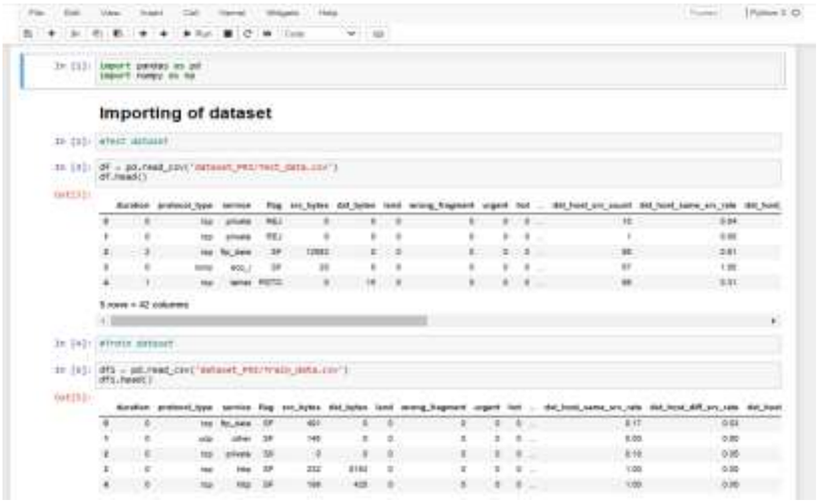
The Python library pandas played a crucial role in data analysis and model development. It offers built-in functions, especially for data cleaning, which is essential for removing errors and ensuring the dataset is accurate. The dataset initially contained several null and blank fields, which were either removed or filled with zeroes using pandas' drop and fillna functions. Handling such missing data is important because it can otherwise lead to incorrect results.

The dataset was further processed by applying ML techniques, with RF classifiers being used to predict intrusions.

The accuracy score was computed to measure the correctness of the model's predictions, while precision was calculated to determine the proportion of true positives. Data splitting functions were employed to randomly divide the dataset for training and testing, allowing for classification reports to be generated.

Visualizations, such as approval graphs or plots, were also created using plotting functions to better understand the results. The estimator values generated by the models helped assess the system's performance by selecting the optimal predictions. These models provided reliable outputs for detecting intrusions and can be adjusted as needed to refine the analysis process.

The artefact developed in this project demonstrates how data processing, cleaning, and ML techniques can be integrated to build an effective intrusion detection system.



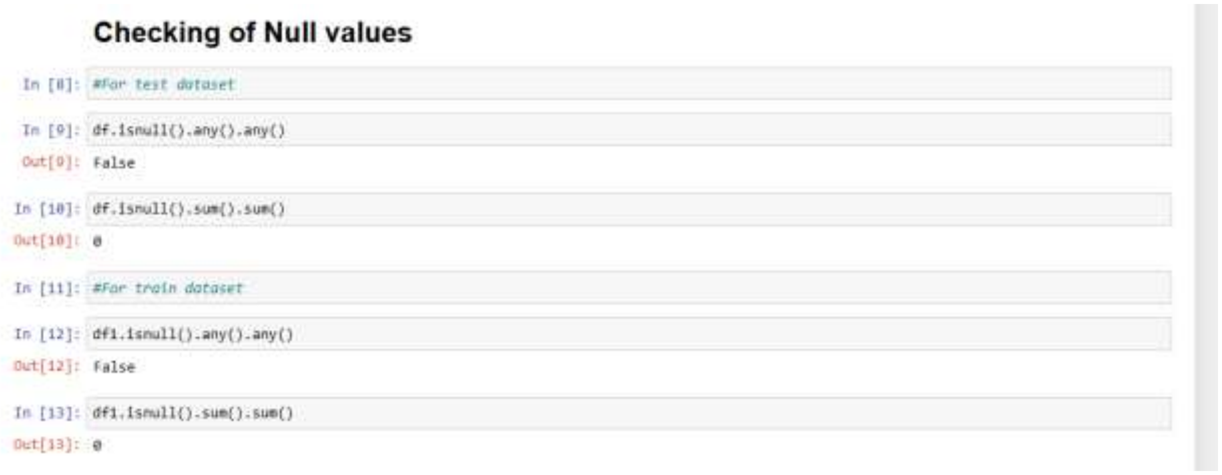
```
In [1]: import pandas as pd
        import numpy as np
```

Importing of dataset

```
In [2]: #test dataset

In [3]: df = pd.read_csv('dataset_PS1/test_data.csv')
        df.head()
Out[3]:
   duration protocol_type  service  flag src_bytes dst_bytes land wrong_fragment urgent hot ... dst_host_srv_count dst_host_same_srv_rate dst_host
0         0           tcp  private   REJ         0         0    0              0      0   0 ...                 10                   0.04
1         0           tcp  private   REJ         0         0    0              0      0   0 ...                  1                   0.00
2         2           tcp ftp_data    SF     12983         0    0              0      0   0 ...                 86                   0.61
3         0          icmp    eco_i    SF        20         0    0              0      0   0 ...                 57                   1.00
4         1           tcp   telnet  RSTO         0        15    0              0      0   0 ...                 86                   0.31

5 rows × 42 columns

In [4]: #train dataset

In [5]: df1 = pd.read_csv('dataset_PS1/train_data.csv')
        df1.head()
Out[5]:
   duration protocol_type  service flag src_bytes dst_bytes land wrong_fragment urgent hot ... dst_host_same_srv_rate dst_host_diff_srv_rate dst_host
0         0           tcp ftp_data   SF       491         0    0              0      0   0 ...                   0.17                   0.03
1         0           udp    other   SF       146         0    0              0      0   0 ...                   0.00                   0.60
2         0           tcp  private   S0         0         0    0              0      0   0 ...                   0.10                   0.05
3         0           tcp     http   SF       232      8153    0              0      0   0 ...                   1.00                   0.00
4         0           tcp     http   SF       199       420    0              0      0   0 ...                   1.00                   0.00
```

Figure 5.2.1: Acquisition of Training and Testing Data

The dataset analysis begins by importing two files, "train" and "test", into the system where the processing will be performed. The file paths are specified to load the datasets correctly. The "train" dataset is stored in a variable named df, while the "test" dataset is assigned to df1. Once imported, the datasets can be examined to review the columns and details required for further analysis.

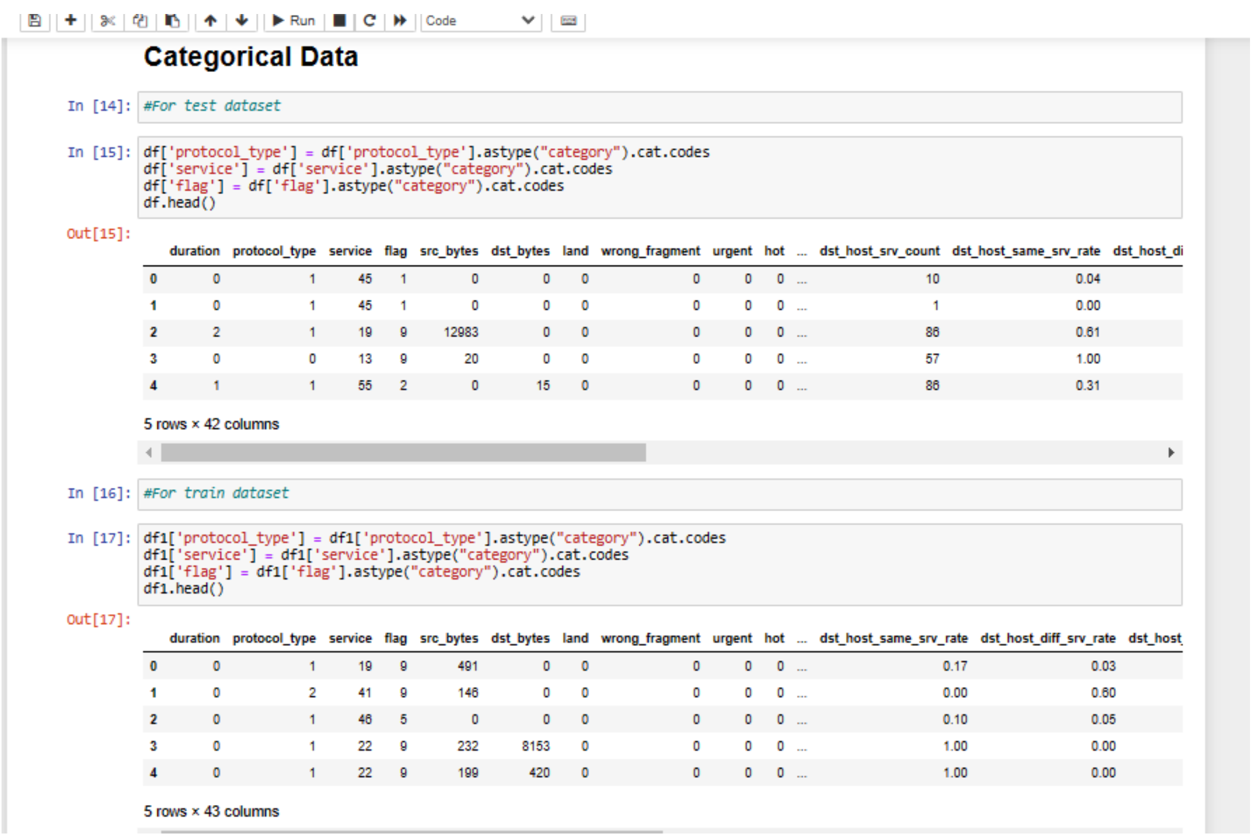


Checking of Null values

```
In [8]: #For test dataset

In [9]: df.isnull().any().any()
Out[9]: False

In [10]: df.isnull().sum().sum()
Out[10]: 0

In [11]: #For train dataset

In [12]: df1.isnull().any().any()
Out[12]: False

In [13]: df1.isnull().sum().sum()
Out[13]: 0
```

Figure 5.2.3: Null Value Inspection for Train and Test Datasets

Both the train and test datasets are checked for null values to prevent errors during processing. This is done using the "isnull" function along with "any" and "sum" to identify the presence and total count of missing values. Handling these null entries ensures improved predictions when using models like the RF classifier and helps in achieving better accuracy scores.



Categorical Data

```
In [14]: #For test dataset

In [15]: df['protocol_type'] = df['protocol_type'].astype("category").cat.codes
         df['service'] = df['service'].astype("category").cat.codes
         df['flag'] = df['flag'].astype("category").cat.codes
         df.head()
Out[15]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	1	45	1	0	0	0	0	0	0	...	10	0.04	
1	0	1	45	1	0	0	0	0	0	0	...	1	0.00	
2	2	1	19	9	12983	0	0	0	0	0	...	86	0.61	
3	0	0	13	9	20	0	0	0	0	0	...	57	1.00	
4	1	1	55	2	0	15	0	0	0	0	...	86	0.31	

5 rows × 42 columns

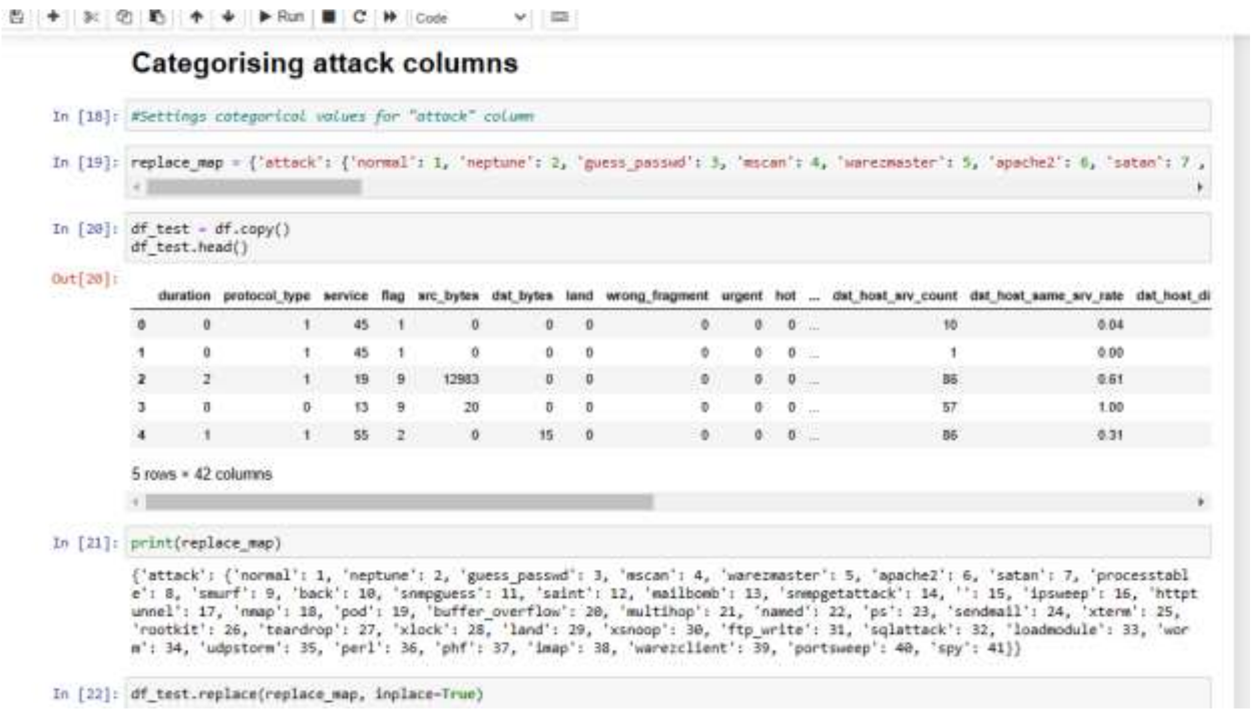
```
In [16]: #For train dataset

In [17]: df1['protocol_type'] = df1['protocol_type'].astype("category").cat.codes
         df1['service'] = df1['service'].astype("category").cat.codes
         df1['flag'] = df1['flag'].astype("category").cat.codes
         df1.head()
Out[17]:
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host
0	0	1	19	9	491	0	0	0	0	0	...	0.17	0.03	
1	0	2	41	9	146	0	0	0	0	0	...	0.00	0.60	
2	0	1	46	5	0	0	0	0	0	0	...	0.10	0.05	
3	0	1	22	9	232	8153	0	0	0	0	...	1.00	0.00	
4	0	1	22	9	199	420	0	0	0	0	...	1.00	0.00	

5 rows × 43 columns

Figure 5.2.4: Splitting and Categorizing Data for Train-Test Phases

String variables in the dataset are converted into categorical forms for improved representation and processing. Since Python's ML algorithms cannot directly handle string or object types, these are transformed into numerical values for effective model training. This conversion is applied to columns like "protocol type," "service," and "flag." The output displays how object values are mapped to integers for both datasets.



Categorising attack columns

```
In [18]: #Settings categorical values for "attack" column

In [19]: replace_map = {'attack': {'normal': 1, 'neptune': 2, 'guess_passwd': 3, 'mscan': 4, 'warezmaster': 5, 'apache2': 6, 'satan': 7 ,

In [20]: df_test = df.copy()
         df_test.head()
Out[20]:
```

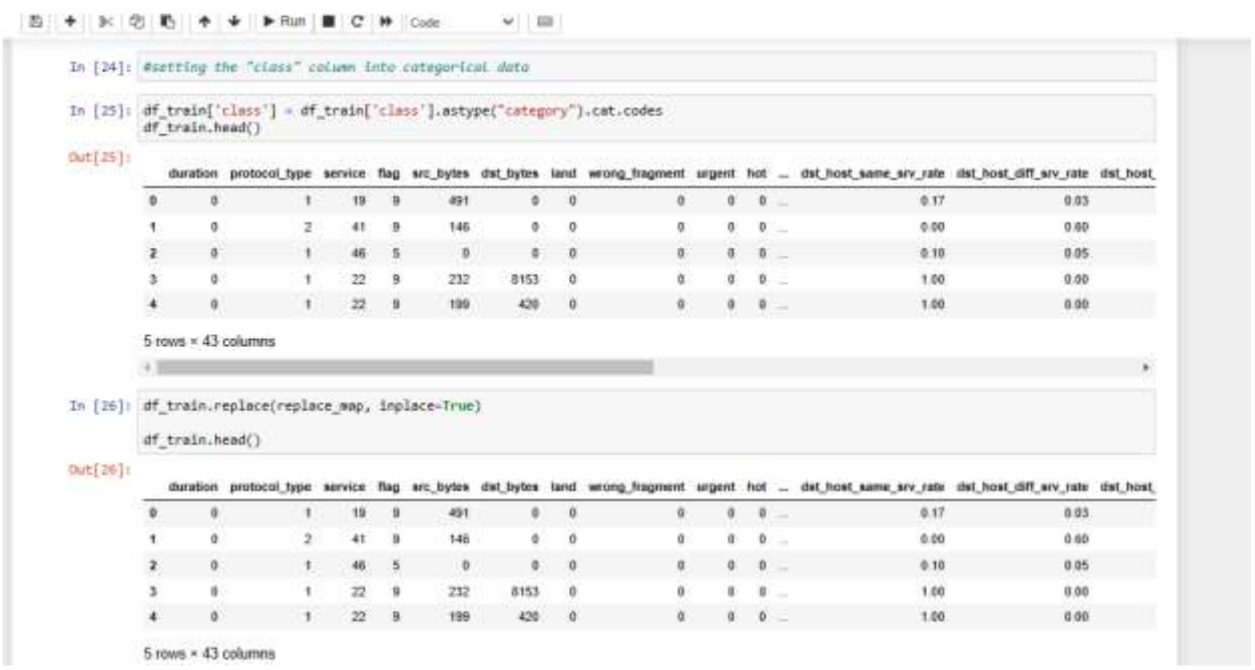
	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	1	45	1	0	0	0	0	0	0	...	10	0.04	
1	0	1	45	1	0	0	0	0	0	0	...	1	0.00	
2	2	1	19	9	12983	0	0	0	0	0	...	86	0.61	
3	0	0	13	9	20	0	0	0	0	0	...	57	1.00	
4	1	1	55	2	0	15	0	0	0	0	...	86	0.31	

5 rows × 42 columns

```
In [21]: print(replace_map)

{'attack': {'normal': 1, 'neptune': 2, 'guess_passwd': 3, 'mscan': 4, 'warezmaster': 5, 'apache2': 6, 'satan': 7, 'processtabl
e': 8, 'smurf': 9, 'back': 10, 'snmpguess': 11, 'saint': 12, 'mailbomb': 13, 'snmpgetattack': 14, '': 15, 'ipsweep': 16, 'httpt
unnel': 17, 'nmap': 18, 'pod': 19, 'buffer_overflow': 20, 'multihop': 21, 'named': 22, 'ps': 23, 'sendmail': 24, 'xterm': 25,
'rootkit': 26, 'teardrop': 27, 'xlock': 28, 'land': 29, 'xsnoop': 30, 'ftp_write': 31, 'sqlattack': 32, 'loadmodule': 33, 'wor
m': 34, 'udpstorm': 35, 'perl': 36, 'phf': 37, 'imap': 38, 'warezclient': 39, 'portsweep': 40, 'spy': 41}}

In [22]: df_test.replace(replace_map, inplace=True)
```

Figure 5.2.5: Assigning Labels for Attack Classification

The *attack* column in both the training and testing datasets is transformed into categorical values to enable effective processing. Each entry in this column is mapped to a unique category, corresponding to the type of attack represented. This step is necessary because the initial rows in the datasets may not share consistent variable values, and a direct categorization could lead to errors. To resolve this, attacks are encoded with numerical labels—for example, “normal” is assigned as 1, “Neptune” as 2, “guess password” as 3, and so forth. These encoded values are then systematically applied to both the training and testing datasets, ensuring consistency in representation for subsequent analysis and model training.

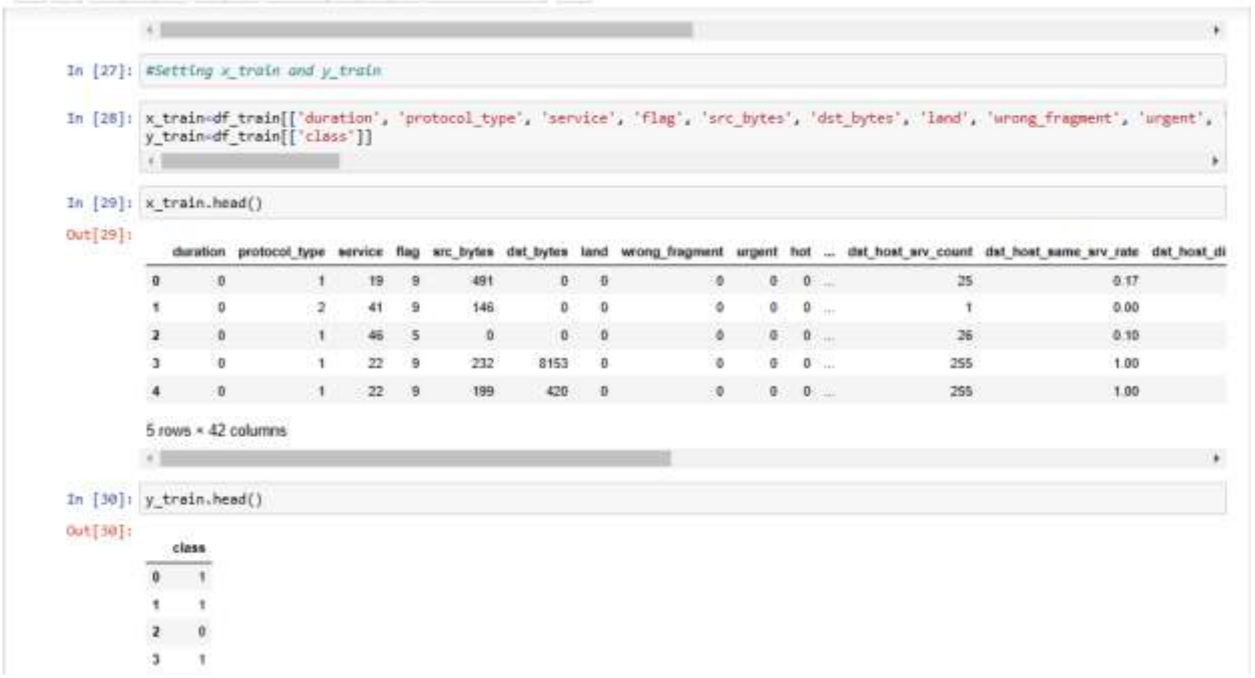


```
In [24]: #setting the "class" column into categorical data

In [25]: df_train['class'] = df_train['class'].astype("category").cat.codes
         df_train.head()

Out[25]:
   duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_same_srv_rate  dst_host_diff_srv_rate  dst_host_
0         0              1       19     9        491          0     0               0       0    0  ...                    0.17                    0.03
1         0              2       41     9        146          0     0               0       0    0  ...                    0.00                    0.60
2         0              1       46     5          0          0     0               0       0    0  ...                    0.10                    0.05
3         0              1       22     9        232       8153     0               0       0    0  ...                    1.00                    0.00
4         0              1       22     9        199        420     0               0       0    0  ...                    1.00                    0.00

5 rows × 43 columns

In [26]: df_train.replace(replace_map, inplace=True)

         df_train.head()

Out[26]:
   duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_same_srv_rate  dst_host_diff_srv_rate  dst_host_
0         0              1       19     9        491          0     0               0       0    0  ...                    0.17                    0.03
1         0              2       41     9        146          0     0               0       0    0  ...                    0.00                    0.60
2         0              1       46     5          0          0     0               0       0    0  ...                    0.10                    0.05
3         0              1       22     9        232       8153     0               0       0    0  ...                    1.00                    0.00
4         0              1       22     9        199        420     0               0       0    0  ...                    1.00                    0.00

5 rows × 43 columns
```

Figure 5.2.6: Reformatting Class Labels into Categorical Variables

The “class” column is categorized separately because it exists only in the training dataset. It is first converted into numerical categorical values, which are then applied to the test dataset as needed. This conversion is essential since the “class” column serves as the target variable for detecting intrusions in the system.

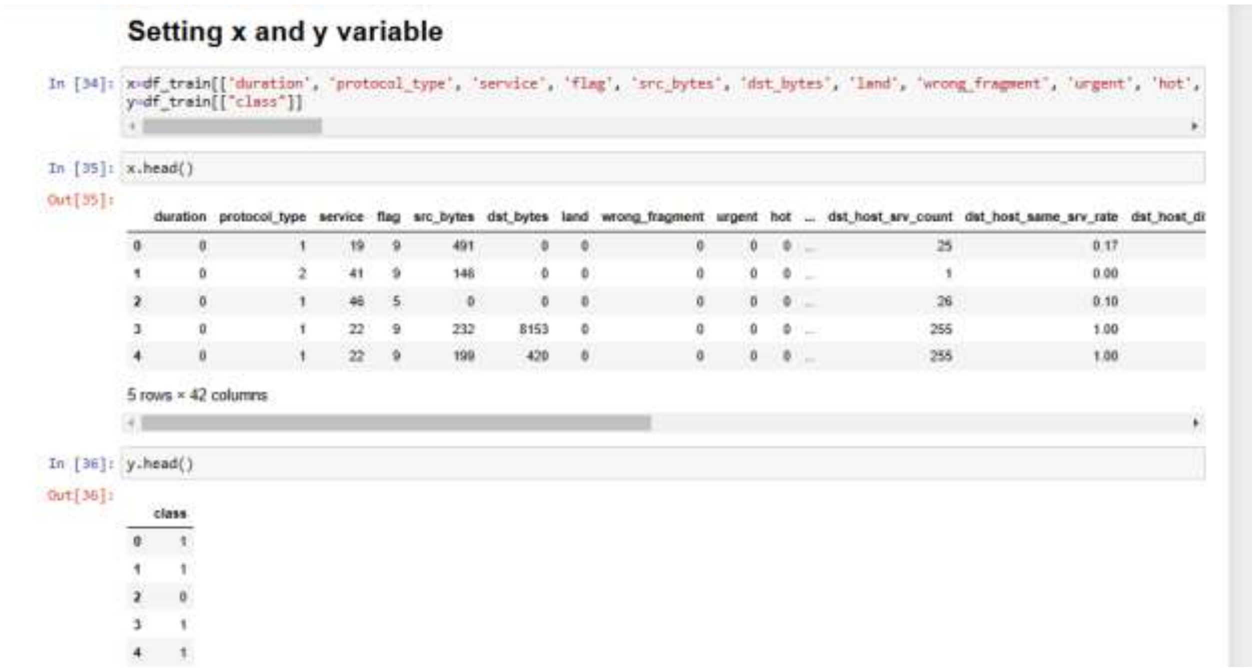
The updated dataset with the categorized “class” column is shown in the figure above.



```
In [27]: #Setting x_train and y_train

In [28]: x_train=df_train[['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent',
         y_train=df_train[['class']]

In [29]: x_train.head()

Out[29]:
   duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_host_di
0         0              1       19     9        491          0     0               0       0    0  ...                  25                    0.17
1         0              2       41     9        146          0     0               0       0    0  ...                   1                    0.00
2         0              1       46     5          0          0     0               0       0    0  ...                  26                    0.10
3         0              1       22     9        232       8153     0               0       0    0  ...                 255                    1.00
4         0              1       22     9        199        420     0               0       0    0  ...                 255                    1.00

5 rows × 42 columns

In [30]: y_train.head()

Out[30]:
   class
0      1
1      1
2      0
3      1
```

Figure 5.2.7: Preparation of x_train and y_train Datasets

The train and test datasets are split into X_train, y_train, X_test, and y_test sets. The target column, “class,” is assigned to y_train, while the remaining features are placed in X_train and X_test. This splitting, along with random shuffling, allows effective training and testing of the prediction models. It also enables model validation, fine-tuning, and comparison for selecting the best-performing model.

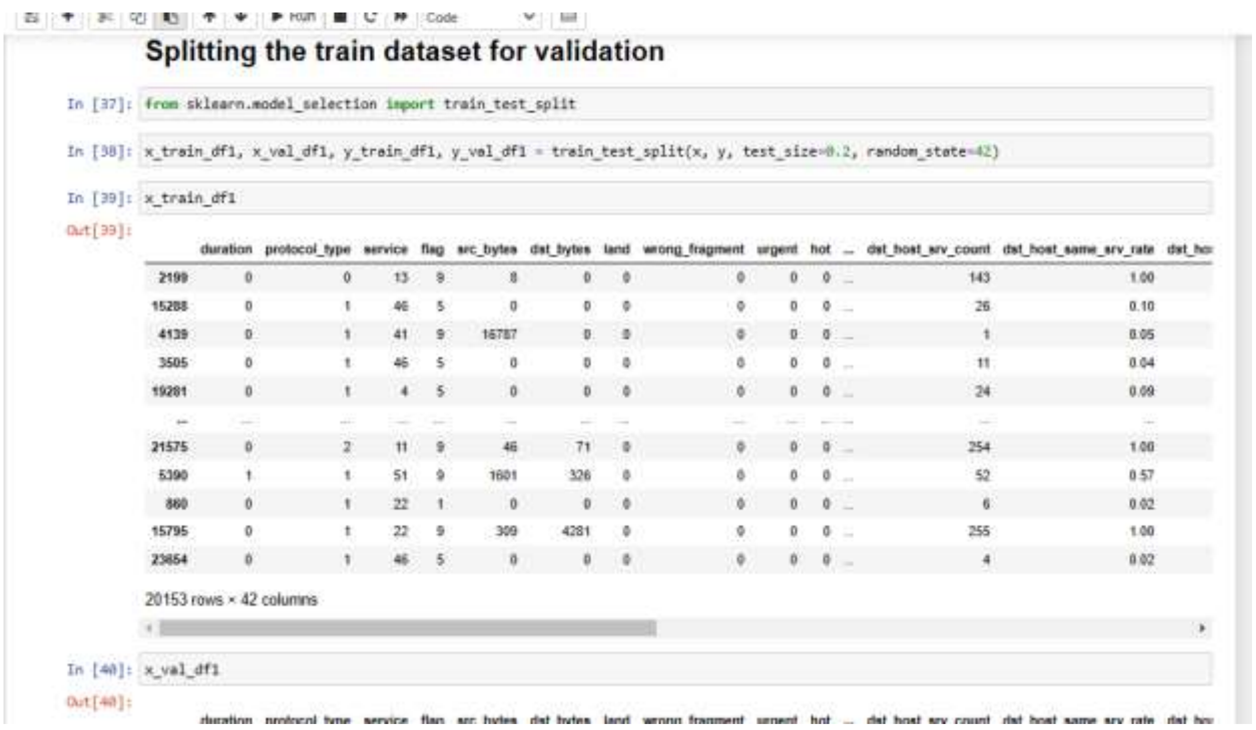


Setting x and y variable

```
In [34]: x=df_train[['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
         y=df_train[["class"]]

In [35]: x.head()
Out[35]:
   duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_host_di
0         0              1       19     9        491          0     0               0       0    0  ...                  25                    0.17
1         0              2       41     9        146          0     0               0       0    0  ...                   1                    0.00
2         0              1       46     5          0          0     0               0       0    0  ...                  26                    0.10
3         0              1       22     9        232       8153     0               0       0    0  ...                 255                    1.00
4         0              1       22     9        199        420     0               0       0    0  ...                 255                    1.00

5 rows × 42 columns

In [36]: y.head()
Out[36]:
   class
0      1
1      1
2      0
3      1
4      1
```

Figure 5.2.8: Dataset Mapping for Predictors and Targets

The training dataset is divided into X and y variables for use in the prediction process with the RF classifier. The “class” column is assigned to y, while the remaining columns are used as X. The successful splitting and conversion of these variables are confirmed by printing them, as shown in the figure above.



Splitting the train dataset for validation

```
In [37]: from sklearn.model_selection import train_test_split

In [38]: x_train_df1, x_val_df1, y_train_df1, y_val_df1 = train_test_split(x, y, test_size=0.2, random_state=42)

In [39]: x_train_df1
Out[39]:
       duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_ho
 2199         0              0       13     9          8          0     0               0       0    0  ...                 143                    1.00
15288         0              1       46     5          0          0     0               0       0    0  ...                  26                    0.10
 4139         0              1       41     9      16787          0     0               0       0    0  ...                   1                    0.05
 3505         0              1       46     5          0          0     0               0       0    0  ...                  11                    0.04
19281         0              1        4     5          0          0     0               0       0    0  ...                  24                    0.09
  ...       ...            ...      ...   ...        ...        ...   ...             ...     ...  ...  ...                 ...                     ...
21575         0              2       11     9         46         71     0               0       0    0  ...                 254                    1.00
 5390         1              1       51     9       1601        326     0               0       0    0  ...                  52                    0.57
  860         0              1       22     1          0          0     0               0       0    0  ...                   6                    0.02
15795         0              1       22     9        309       4281     0               0       0    0  ...                 255                    1.00
23854         0              1       46     5          0          0     0               0       0    0  ...                   4                    0.02

20153 rows × 42 columns

In [40]: x_val_df1
Out[40]:
       duration  protocol_type  service  flag  src_bytes  dst_bytes  land  wrong_fragment  urgent  hot  ...  dst_host_srv_count  dst_host_same_srv_rate  dst_ho
```

Figure 5.2.9: Partitioning Training Data for Validation

5.3 Test-Train Split and Artefact Development

The dataset was split using scikit-learn’s train_test_split into X_train_df1, X_val_df1, y_train_df1, and

y_val_df1. This division allows proper validation and assessment of the model's performance. Generating predictions on the training set helps measure accuracy and evaluate how well the model interprets and classifies the data.
Jupyter Notebook was chosen as the platform for developing the IDS artefact. Datasets were collected from reliable sources such as Kaggle, which ensures data accuracy and relevance for intrusion detection. The datasets contained details like duration, protocol type, service, flags, wrong fragmentation, and attack types. Initial processing involved checking for null values and removing unnecessary columns to improve model performance. Object-type columns were converted to numerical variables for ML compatibility. Attack columns were manually encoded into categorical values to standardize both train and test datasets, ensuring accurate model input.

5.4 Development Methodology Discussion

After data cleaning, the "class" column was added to the training dataset to align with the test dataset. The data was then split into X and y variables, and the RF classifier was applied. This model was chosen because it efficiently handles both categorical and continuous data, and can process large datasets effectively.
The model generated accuracy scores by comparing predictions with actual values, providing insights into the performance of the intrusion detection system. After prediction, the class column was inserted into the test dataset. The developed IDS can detect potential intrusions and, when combined with prevention mechanisms, can help protect organizational data from malicious attacks. ML-based IDS provides early detection of threats, minimizing damage and losses. The methodology can be adapted for other datasets or alternative approaches to enhance intrusion detection efficiency.

6: Conclusions

6.1 Introduction

This study focused on developing an intrusion detection system (IDS) to identify malicious activities by monitoring network traffic within an organization. A dataset containing both normal and attack scenarios was analyzed using Python in Jupyter Notebook. Various ML models, particularly the RF Classifier, were employed to predict intrusions and calculate accuracy scores. With the increasing number of internet users and online applications, IDS deployment is becoming essential for maintaining network security. The developed system sorts and organizes alerts for incoming connections, detecting potentially malicious activities. While IDS primarily identifies threats, detected information can be shared with IT teams to prevent future attacks, and the system can be enhanced with preventive measures.

6.2 General Conclusions

Key findings indicate that ML techniques effectively detect intrusion activities. Using RF Classifier, high accuracy scores were achieved, providing detailed insights into the frequency and types of attacks. The IDS monitors network traffic, inspects packets, and compares them with historical data to identify anomalies and potential threats. This automation reduces human labor, improves detection accuracy, and supports preventive actions to safeguard organizational systems.

6.3 Research Question Conclusions

The identification of intrusions relies on sensors and monitoring consoles to record incoming network events. These records are analyzed with ML techniques to compute accuracy scores. Open-source tools like Snort can provide rules for preventing malicious activity. In this study, the RF Classifier compared historical data with new observations to detect attack patterns. The IDS successfully identified attack types and quantities, supporting detailed analysis and threat management within an organization.

6.4 Recommendations

For effective deployment, IDS should be accompanied by manuals, tools, and procedures for IT teams to respond to detected attacks. IDS can function as an alert system, while preventive actions must be taken by professionals. Both network-based and host-based IDS approaches are recommended, with anomaly-based and signature-based detection as the most common methods. Anomaly-based systems are particularly useful for detecting unknown attacks. By integrating policies and legal measures, organizations can take action against malicious activities and improve system security.

6.5 Errors and Limitations

- Data imbalance and scarcity: Some datasets contain uneven representation of attack types, causing challenges in accurately modeling low-frequency attacks. Large datasets may exceed Python's processing capabilities for certain features.
- Low-frequency attacks: Attacks such as U2R and R2L are difficult to detect due to insufficient training data. ML models may not capture these infrequent patterns effectively.
- Real-time re-training limitations: ML-based IDS cannot always adapt in real-time to new attack patterns. Labeled datasets may not cover all possible attack scenarios, limiting predictive performance.

6.6 Recommendations for Further Study

Future work should explore semi-supervised learning and clustering techniques to improve detection of low-frequency attacks. Hyperparameter tuning and ensemble learning methods can enhance RF performance. Buffer overflow and teardrop attack detection should be further investigated. Suspicious activity monitoring and re-training strategies should focus on improving algorithm efficiency, security, and resilience against parallel attacks. Additionally, database management and encoding formats should be optimized for handling IDS data effectively.

7: Refrences

Books

1. K. Kim, M. E. Aminanto, and H. C. Tanuwidjaja, *Network Intrusion Detection Using Deep Learning: A Feature Learning Approach*, Springer, 2018.

2. R. Rehim, *Python Penetration Testing Cookbook: Practical Recipes on Implementing Information Gathering, Network Security, Intrusion Detection, and Post-Exploitation*, Packt Publishing Ltd, 2017.

Journals

1. H. Alkahtani, H. H. A. Theyazn, and M. Al-Yaari, "Adaptive Anomaly Detection Framework Model Objects in Cyberspace," *Applied Bionics and Biomechanics*, vol. 2020, 2020.

2. M. Al-Omari, M. Rawashdeh, F. Qutaishat, A. H. Mohammad, and N. Ababneh, "An Intelligent Tree-Based Intrusion Detection Model for Cyber Security," *Journal of Network and Systems Management*, vol. 29, no. 2, pp. 1-18, 2021.

3. R. A. Calix, S. B. Singh, T. Chen, D. Zhang, and M. Tu, "Cyber Security Tool Kit (CyberSecTK): A Python Library for ML and Cyber Security," *Information*, vol. 11, no. 2, p. 100, 2020.

4. M. Chattopadhyay, R. Sen, and S. Gupta, "A Comprehensive Review and Meta-Analysis on Applications of ML Techniques in Intrusion Detection," *Australasian Journal of Information Systems*, vol. 22, 2018.

5. A. Chiche and M. Meshesha, "Towards a Scalable and Adaptive Learning Approach for Network Intrusion Detection," *Journal of Computer Networks and Communications*, vol. 2021, 2021.

6. J. Gupta and J. Singh, "Detecting Anomaly Based Network Intrusion Using Feature Extraction and Classification Techniques," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, pp. 1453-1456, 2017.

7. F. Imran and F. Jamil, "An Ensemble of Prediction and Learning Mechanism for Improving Accuracy of Anomaly Detection in Network Intrusion Environments," *Sustainability*, vol. 13, no. 18, p. 10057, 2021.

8. I. Jung and J. Lim, "PF-TL: Payload Feature-Based Transfer Learning for Dealing with the Lack of Training Data," *Electronics*, vol. 10, no. 10, p. 1148, 2021.

9. S. M. Kasongo and Y. Sun, "Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset," *Journal of Big Data*, vol. 7, no. 1, 2020.

10. M. Kavitha, K. Elamukhil, R. Ajeeth, R. Ashwin, and V. Balasubramaniam, "Distributed Ensemble Based Deep Learning Architecture for Intrusion Detection Against Cyber Attacks," *Journal of Physics: Conference Series*, vol. 1916, no. 1, 2021.

11. N. Khatri, R. Shrestha, and S. Y. Nam, "Security Issues with In-Vehicle Networks, and Enhanced Countermeasures Based on Blockchain," *Electronics*, vol. 10, no. 8, p. 893, 2021.

12. E. Maia, "Intelligent Cyber Attack Detection and Classification for Network-Based Intrusion Detection Systems," *Applied Sciences*, vol. 11, no. 4, p. 1674, 2021.

13. P. Nerlikar, S. Pandey, S. Sharma, and S. Bagade, "Analysis of Intrusion Detection Using ML Techniques," *International Journal of Computer Networks and Communications Security*, vol. 8, no. 10, pp. 84-93, 2020.

14. V. Priya, I. S. Thaseen, R. G. Thippa, M. K. Aboudaif, and A. N. Emad, "Robust Attack Detection Approach for IIoT Using Ensemble Classifier," *Computers, Materials & Continua*, vol. 66, no. 3, pp. 2457-2470, 2021.

15. S. Rajagopal, P. P. Kundapur, and K. S. Hareesha, "A Stacking Ensemble for Network Intrusion Detection Using Heterogeneous Datasets," *Security and Communication Networks*, vol. 2020, 2020.

16. K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, S. Chen, D. Liu, and J. Li, "Performance Comparison and Current Challenges of Using ML Techniques in Cybersecurity," *Energies*, vol. 13, no. 10, p. 2509, 2020.

17. Y. N. Soe, Y. Feng, I. S. Paulus, R. Hartanto, and K. Sakurai, "Towards a Lightweight Detection System for Cyber Attacks in the IoT Environment Using Corresponding Features," *Electronics*, vol. 9, no. 1, p. 144, 2020.

18. M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, "Supervised Learning in Physical Networks: From ML to Learning Machines," *Physical Review X*, vol. 11, no. 2, 2021.

19. T. T. H. Le, Y. Kim, and H. Kim, "Network Intrusion Detection Based on Novel Feature Selection Model and Various Recurrent Neural Networks," *Applied Sciences*, vol. 9, no. 7, 2019.

20. R. N. Thomas and R. Gupta, "Design and Development of an Efficient Network Intrusion Detection System Using ML Techniques," *Wireless Communications & Mobile Computing*, vol. 2021, 2021.

21. S. F. Yilmaz and S. S. Kozat, "PySAD: A Streaming Anomaly Detection Framework in Python," arXiv preprint arXiv:2009.02572, 2020.

22. Y. Zhao, J. Chen, D. Wu, J. Teng, N. Sharma, A. Sajjanhar, and M. Blumenstein, "Network Anomaly Detection by Using a Time-Decay Closed Frequent Pattern," *Information*, vol. 10, no. 8, p. 262, 2019.

23. Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A Python Toolbox for Scalable Outlier Detection," arXiv preprint arXiv:1901.01588, 2019.

Online Articles

1. "American Banking and Market News - American Consumer News: Comparing Intrusion (NASDAQ:INTZ) and Lantronix (NASDAQ:LTRX)," *Chatham: Newstex*, 2020.

2. A. Chon, R. R. Sims, and T. Gregorio, "Develop Your Cyber Resilience Plan," *Cambridge: Massachusetts Institute of Technology*, 2020.

3. "Corp, N., 2021. Newsfile Corp.: GoldSpot Discoveries & Critical Elements Identify Lithium-Tantalum Targets in the Nemiscau Belt Using Artificial Intelligence," *Chatham: Newstex*, 2021.

4. "Ticker Report - American Consumer News: Lantronix (NASDAQ:LTRX) & Intrusion (NASDAQ:INTZ) Financial Contrast," *Chatham: Newstex*, 2020.

5. "ValueWalk: How do you design ML models for malicious network detection?" *Chatham: Newstex*, 2019.

6. "Watchlist News - American Consumer News: Head-To-Head Review: Intrusion (NASDAQ:INTZ) and Lantronix (NASDAQ:LTRX)," *Chatham: Newstex*, 2020.

Websites

1. activereach.net, "Network Security," [Online]. Available: https://activereach.net/solutions/network-security/protect/network-perimeter-security/network-perimeter-ids-ips/. [Accessed: 04-Oct-2021].

2. geeksforgeeks.org, "Intrusion Detection System Using ML Algorithms," [Online]. Available: https://www.geeksforgeeks.org/intrusion-detection-system-using-machine-learning-algorithms/. [Accessed: 30-Sep-2021].

3. iup.edu, "WorkArea," [Online]. Available: https://www.iup.edu/WorkArea/linkit.aspx?LinkIdentifier=id&ItemID=80703. [Accessed: 30-Sep-2021].

4. section.io, "Comparing Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS)," [Online]. Available: https://www.section.io/engineering-education/comparing-intrusion-detection-systems-ids-and-intrusion-prevention-systems-ips/. [Accessed: 04-Oct-2021].

5. snort.org, "Snort Application," [Online]. Available: https://www.snort.org/. [Accessed: 04-Oct-2021].