الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

Université Badji Mokhtar - Annaba
Badji Mokhtar – Annaba University

جامعة باجي مختار – عنابـــة

Faculté : Technologie
Département : Informatique
Domaine : Mathématique-Informatique
Filière : Informatique
Spécialité : systèmes informatiques

Mémoire

Présenté en vue de l'obtention du Diplôme de Licence

Thème

Détection d'intrusion par classifieurs classiques

Présenté par : Hadiby Chirine
Hamdoud Fatma Zohra

Encadrant : Mme.Yahiouche Salima Grade : MAA Université Badji Mokhtar-Annaba

Année Universitaire : 2022/2023

Acknowledgement

Before all, it is a genuine pleasure to express our deep sense of thanks and gratitude to our supervisor Mme. Yahiouche Salima for her guidance and assistance throughout the different stages of our project development.

Also, we want to show our appreciation for our colleagues who helped us with the constant improvement of our project.

Dedication

This thesis is dedicated to our families and our friends and all the people who helped and supported us through our journey.

To my mother Linda and my father Mohamed without whom I wouldn’t get this far.

To my siblings: Saif eddine, Riham, Ritadj, Hanine who were there for me throughout my journey.

To my partner Fatma Zohra from Chirine, I wanted to take a moment to express my deepest gratitude for the incredible effort and hard work you've put into our studies. Your dedication has truly made a difference, and I genuinely appreciate the partnership we've formed.

To my partner Chirine from Fatma Zohra, without your support, your encouragement, always pushing us to do better none of this would’ve happened. Your commitment and hard work have truly had a significant impact.

Thank you all for this beautiful supporting.

To Rym our best friend whitout whome the whole project whould’ve gone to ashes, thank you for your support, your help and everything you have given to us through the hole year especially these last days. You are truly an angel.

Table of contents

Table of figure

Tables

Acronyms

AI : Artificial Intelligence

IOT : Internet Of Things

ML : Machine learning

IDS : Intrusion Detection System

NIDS : Network-based IDS

HIDS : Host-based IDS

DOS : Denial-Of-Service attacks

U2R : User-To-Root

R2L : Remote-To-Local

SVM : Support Vector Machine

GRU : Gated Recurrent Unit

LSTM : Long Short-Term Memory

RNN : Recurent Neural Network

Introduction

1. Project ‘s context

The constant development of technology has led to both new opportunities and new challenges in the field of network security. As technology evolves, new vulnerabilities and threats emerge, requiring security professionals to stay up-to-date with the latest security trends and technologies. Additionally, the rise of new technologies such as the Internet of Things (IoT), cloud computing, and artificial intelligence (AI) has created new security concerns and complexities that need to be addressed. At the same time, advancements in security technologies such as intrusion detection and prevention systems, biometric authentication, and machine learning are helping to strengthen network security and improve overall cyber defense. It is important for organizations to stay vigilant and adapt to the ever-changing technology landscape to ensure the security of their networks and protect against potential threats.

2. Problem

No firewall is foolproof, and no network is impenetrable. Attackers continuously develop new exploits and attack techniques designed to overcome our defenses. With the rise of ransomware attacks, data breaches, and other cyber threats, it is imperative that organizations take network security seriously and invest in robust security measures to protect their valuable assets and data.

3. Motivations

Cyberattacks can cause significant harm to individuals, organizations, and even countries by taking so many forms including malware. The faster a cyberattack is detected, the sooner remedial measures can be taken to limit the damage. The motivation for detecting cyberattacks and intrusions in general is to protect against the increasing threat of cybercrime and ensure the security and safety of individuals, and society as a whole.

4. Objectives

We aim to build an Intrusion Detection System model to identify anomalous behaviors by analyzing network traffic using some Machine Learning classifiers.

5. Project's content

- **Chapter 1:** The first chapter provides a comprehensive review of the existing literature on intrusion detection systems (IDS), machine learning and classical classifiers.
- **Chapter 2:** This chapter will be completely dedicated to the system's conception and its theoretical aspects.
- **Chapter 3:** In this last chapter, we will discuss different tools that helped developing our project along with its implementation. Finally, we will end up with a general conclusion and discuss some perspectives.

Chapter 1: Background and basic concepts

The first chapter is designed to provide a solid foundation of knowledge in Intrusion Detection System. We will cover the fundamental concepts and principles of IDS, ML and classical classifiers, including their definitions and basic applications that are important for the understanding of the ongoing project.

1. Intrusion Detection Systems (IDS)

1.1. Definition

An intrusion detection system (IDS) is a security technology designed to monitor network traffic or system activity in order to detect and alert on potential threats and attacks. IDS solutions typically use a combination of signature-based detection, behavioral analysis, and machine learning algorithms to identify suspicious activity and patterns that may indicate an intrusion. The primary goal of an IDS is to help organizations quickly detect and respond to security incidents in order to minimize the impact of a potential attack on their systems or networks.

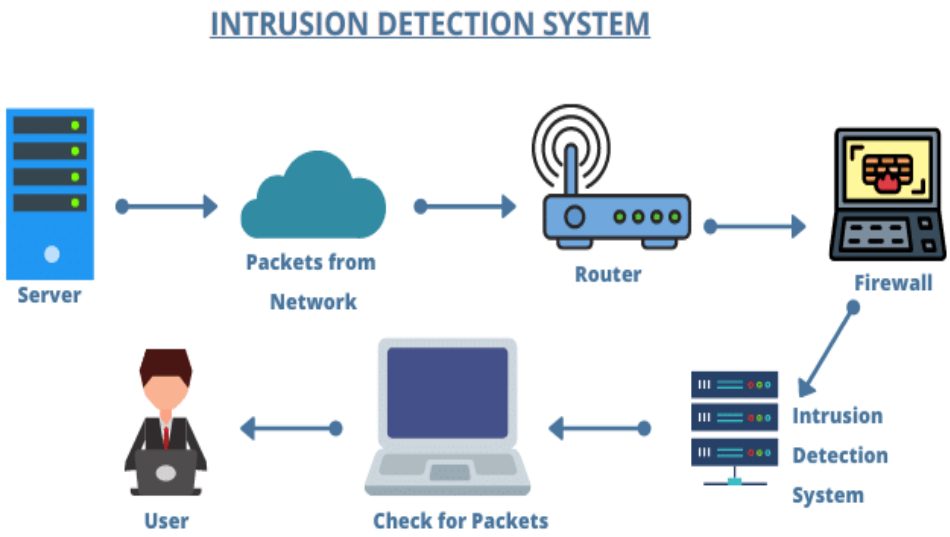

Figure 1 : Intrusion Detection System [1]

1.2. Types of Intrusion Detection Systems (IDS)

There are two common types of IDS, each with its own unique approach to detecting and preventing security threats: [2]

1.2.1. Network-based IDS (NIDS)

A NIDS is a solution that monitors the entire network through one or more touch points, and alerts security teams when it detects abnormal behavior, such as suspicious traffic patterns, port scanning, or other potential security threats.

1.2.2. Host-based IDS (HIDS)

A HIDS monitors individual devices or hosts, such as servers or workstations, for signs of unauthorized access or malicious activity. It analyzes system logs, file system changes, and other indicators to detect potential threats.

1.3. Intrusion Detection System methods

Three methods of IDS exist: signature-based detection, anomaly-based detection, and hybrid detection, which combines elements of both.

1.3.1. Signature-based detection

This method solution uses fingerprints of known threats to identify them. Once malware or other malicious content has been identified, a signature is generated and added to the list used by the IDS solution to test incoming content. This enables an IDS to achieve a high threat detection rate with no false positives because all alerts are generated based upon detection of known-malicious content. However, a signature-based IDS is limited to detecting known threats and is blind to zero-day vulnerabilities.

1.3.2. Anomaly-based detection

This method builds a model of the “normal” behavior of the protected system. All future behavior is compared to this model, and any anomalies are labeled as potential threats and generate alerts. While this approach can detect novel or zero-day threats, the difficulty of building an accurate model of “normal” behavior means that these systems must balance false positives (incorrect alerts) with false negatives (missed detections).

1.3.3. Hybrid detection

This method uses both signature-based and anomaly-based detection. This enables it to detect more potential attacks with a lower error rate than using either system in isolation.

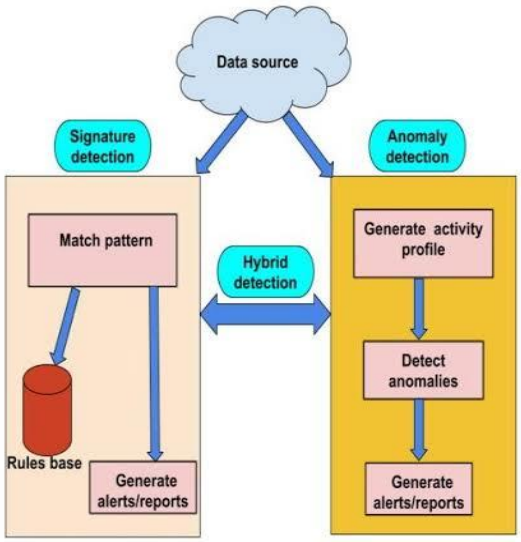

Figure 2: Detection methods used by IDS [3]

2. Machine Learning (ML)

2.1. Definition

Machine learning is a subset of artificial intelligence that enables computer systems to learn and improve from experience without being explicitly programmed. Figure 3 displays different machine learning classes and their applications.

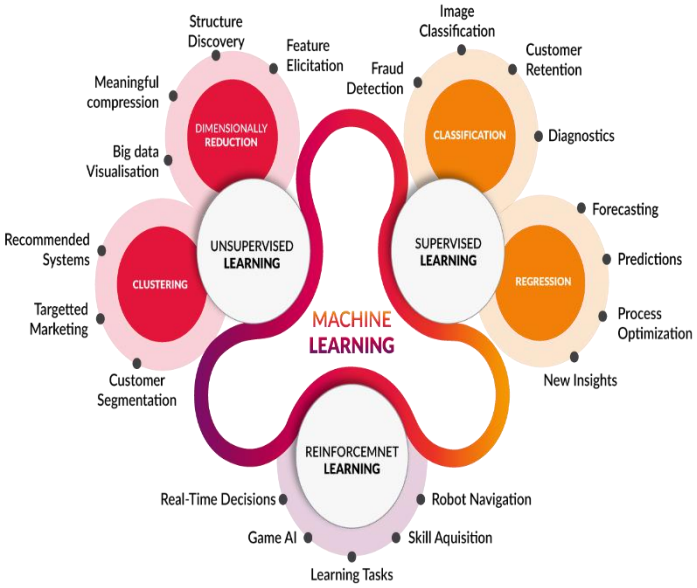

Figure 3: Machine Learning and its applications [4]

2.3. Developing a Machine Learning model

Developing a machine learning model typically involves the following steps: [5]

1. Collecting and preparing the data.
2. Choosing an appropriate machine learning algorithm.
3. Training the model on the data.
4. Evaluating the model's performance.
5. Fine-tuning the model by adjusting hyperparameters and/or modifying the features
6. Deploying the model to a production environment. It's important to note that the exact process and techniques used may vary depending on the specific problem and data set being used.

3. Classical Classifiers

3.1. Definition

Machine learning classical classifiers are algorithms that use supervised learning to predict the class or category of a given input based on training data. These classifiers are called "classical" because they have been used in the field of machine learning for many years and are well-established. Some common examples of machine learning classical classifiers are as follows:

3.1.1. Decision Trees

A model that recursively partitions the feature space based on the values of the input features, and makes predictions based on the most common class of the training examples in each partition.

3.1.2. AdaBoost

Adaboost classifier is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. It iteratively adjusts the weights of training instances, focusing on misclassified examples to improve overall classification performance.

3.1.3. Naive Bayes

A probabilistic model that predicts the class of a new input based on the conditional probability of the input features given each possible class, using Bayes' theorem.

3.1.4. Support Vector Machines (SVMs)

A model that finds the best hyperplane to separate the different classes based on the input features.

3.1.5. Random Forests

An ensemble of decision trees that reduces overfitting by combining multiple trees to create a stronger classifier.

5. Previous studies on IDS and their findings

Several studies have explored machine learning-based intrusion detection system (IDS) models and found promising results. One study compared different algorithms, revealing that the random forest algorithm excelled in accuracy and false positive rates for detecting network intrusions. Another study focused on deep learning models, showing high accuracy in identifying intrusions and detecting previously unknown attacks. Overall, these studies highlight the potential of machine learning-based IDS models to enhance intrusion detection systems, with ongoing research aimed at further advancements.

6. Conclusion

Intrusion Detection System (IDS) plays a vital role in safeguarding computer networks against malicious activities. Machine learning techniques are widely used in IDS to identify abnormal network traffic that may signify an attack. Traditional classifiers like decision trees, random forest, and support vector machines are commonly used to classify network traffic as either normal or malicious. Overall, IDS incorporating machine learning and classical classifiers provide an efficient approach to detect and mitigate cyberattacks. However, ongoing research and enhancements are imperative to adapt to the constantly evolving cybersecurity environment.

Chapter 2: System conception

This chapter provides a comprehensive overview of the key stages of the machine learning pipeline which includes data collection, pre-processing, feature selection, model selection, and model evaluation and the importance of each stage in building an accurate and effective model.

1. Insufficiency of existing solutions dealing with the subject

The KDD Cup 1999 dataset was the first public benchmark for evaluating intrusion detection systems (IDS) and provided a valuable resource for researchers to develop and test their models. However, it suffered from several insufficiencies, such as an insufficient representation of modern attacks and a lack of diversity in the data. As a result, the NSL-KDD dataset was developed as an evolution of the KDD Cup 1999 dataset, which addressed some of these issues. The NSL-KDD dataset included more modern attacks and a more diverse set of data to represent the current threat landscape. However, even with these improvements, existing IDS solutions still face significant challenges in effectively detecting and responding to cyber threats in real-time. As such, there is a continued need for more advanced and adaptive approaches to address the evolving nature of cyber threats.

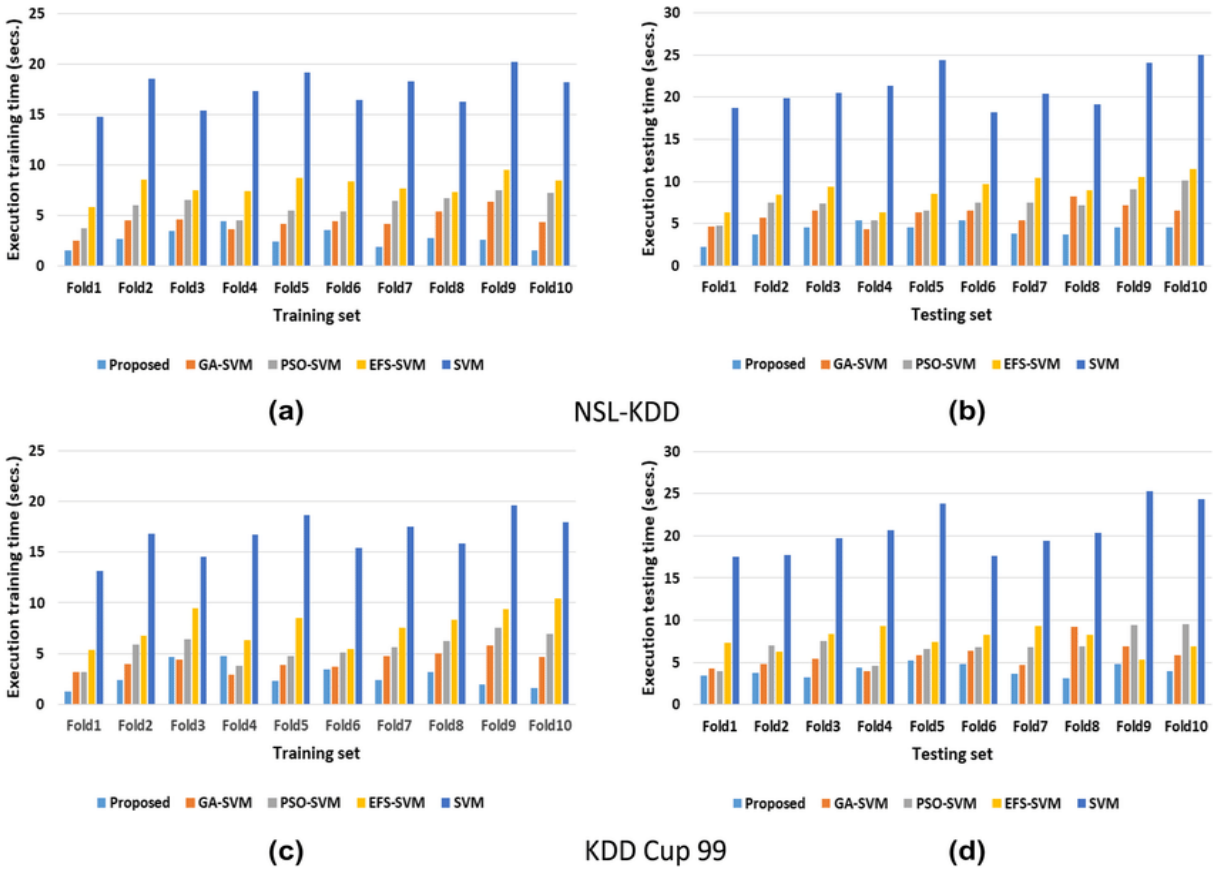

Figure 4 : Comparison of training time and testing time (a-d) in NSL-KDD and KDD Cup 99 [10]

2. Work hypothesis

This study will be dedicated to the functionalities of the system and its characteristics. By examining the features and capabilities of the system, as well as its technical specifications and operational requirements, we can identify potential areas for optimization and enhance overall system functionality.

2.1. System functionalities

The analysis and understanding of system functionalities are essential for developing effective and efficient system that allows the detection of any intrusion, attack, malware within the dataset samples:

- **Attack detection :** The system detects different types of network attacks. It uses machine learning algorithms to analyze network traffic and detect patterns that are indicative of an attack.

- **Real-time monitoring :** It continuously monitors network traffic in real-time to identify any suspicious activity that may indicate an attack.
- **Alert generation :** When the system identifies a potential attack, it generates an alert to notify the user. The alert contains information about the type of attack and the classifier used to detect it.

2.2. System characteristics

Our machine learning model has the next characteristics:

- **Restricted coverage:** The system is designed to detect only a limited number of attack types or network traffic patterns. It may not have the ability to detect more complex attacks.
- **Simple architecture:** The architecture of our system may be simple, consisting of a few machine learning and deep learning algorithms.
- **Limited resources:** It is limited in terms of computational resources, memory, and storage capacity. This may affect the performance and accuracy of the IDS.
- **Basic user interface:** The user interface is simple and straightforward, with limited reporting and visualization capabilities.

3. Functional Architecture of the Application

3.1. Model's Architecture

As depicted in the diagram below, our machine learning models (SVM, Decision Tree, Random Forest, AdaBoost) are typically constructed by combining various layers, each serving a specific purpose in the learning process. These layers include the input layer, hidden layers, and the output layer. The hidden layers, often composed of multiple layers, perform complex computations and extract meaningful representations from the input data. The output layer produces the final prediction or classification based on the learned representations.

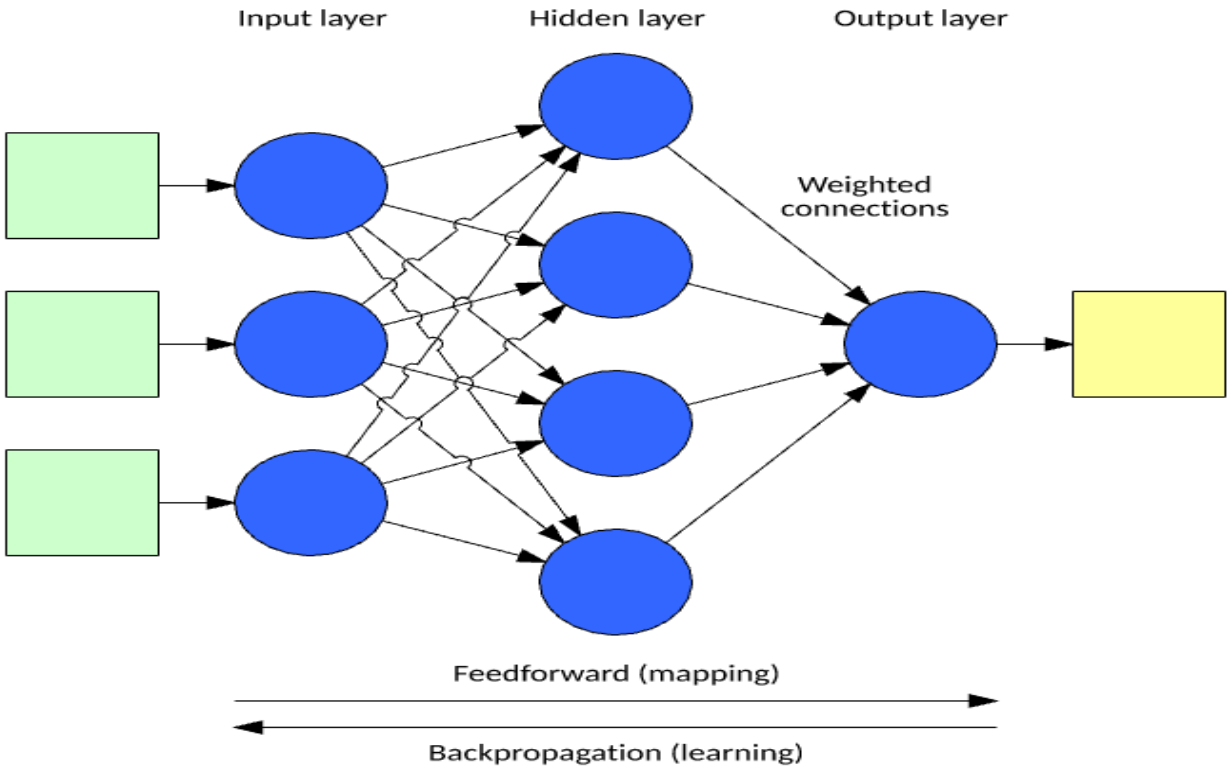

Figure 5 : Machine Learning model's general architecture [11]

As shown in the chart below, our deep learning models (GRU, LSTM, RNN, Sequential) are constructed using artificial neural networks composed of multiple layers. These layers include an input layer, one or more hidden layers, and an output layer. The hidden layers, often consisting of numerous neurons, perform complex computations and progressively learn hierarchical representations of the input data. Deep learning models leverage these layers to automatically extract intricate patterns and features, enabling them to handle complex tasks such as image recognition, natural language processing, and speech synthesis.

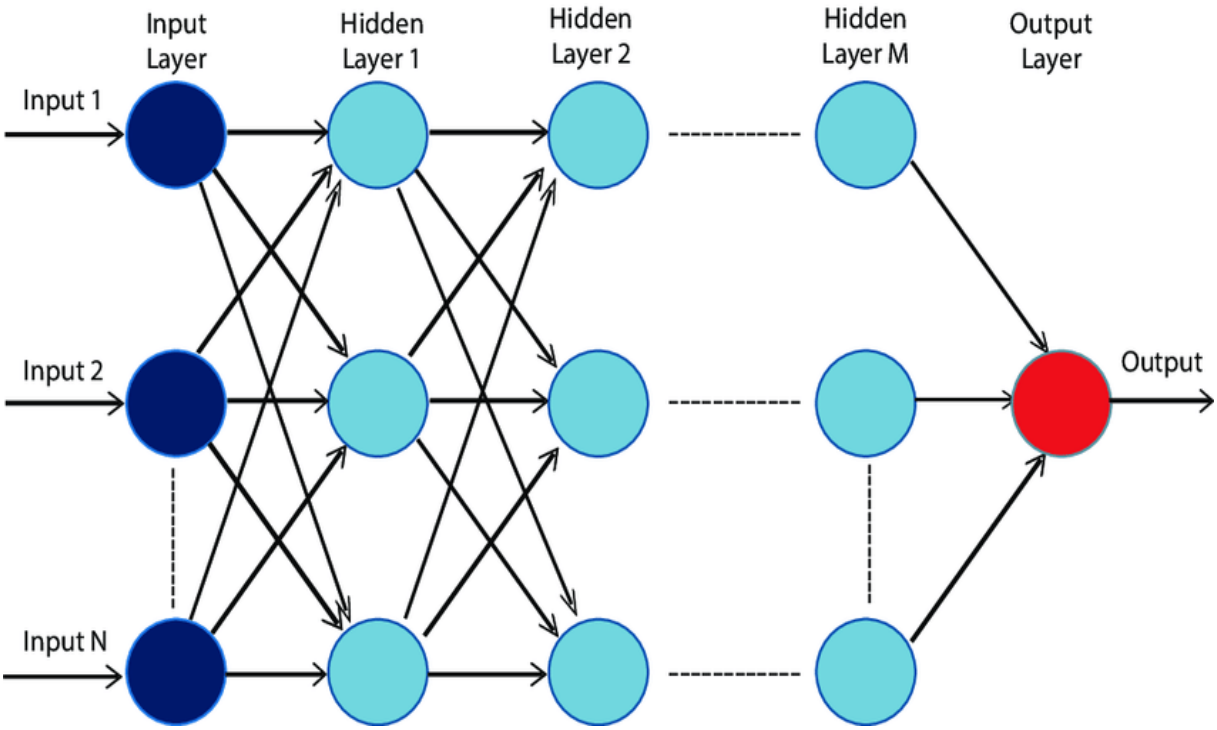

Figure 6: Deep Learning model's general architecture [12]

«Check annex A for more information about the used deep learning models. »

3.2. Data collection

The first and essential step in any classification task is data collection. For this purpose, we decided to utilize the NSL-KDD dataset downloaded from Kaggle, which is widely used in the field of network intrusion detection.

The dataset is constructed as follows:

- A total of 20,000 training sample
- 4047 samples in the validation dataset
- 1000 samples each of NORMAL, DoS, and Probe types.
- 995 samples of R2L.
- 52 samples of U2R.

This dataset contains various features related to network connections, including protocol type, service, source and destination addresses, flags, and more. With a substantial amount of data available, the NSL-KDD dataset is well-suited for our classification task.

3.3. Feature selection

In the feature selection section, we went through 4 steps:

3.3.1. Feature Selection Technique:

In the feature selection process of our IDS, we used Recursive Feature Elimination (RFE) from the scikit-learn library. RFE is a wrapper method that iteratively removes less important features based on the importance ranking determined by the SVM classifier. By recursively eliminating features, RFE aims to identify the most relevant ones that have the most significant impact on the performance of the SVM model. RFE helps us focus on the crucial features for our IDS, improving its ability to detect and classify intrusion activities accurately.

3.3.2. Model and Parameter Selection:

We employed multiple machine learning algorithms, including Support Vector Machines (SVM), Decision Tree, Random Forest, and AdaBoost. For exemple for SVM, we utilized a linear kernel with hyperparameters set to C=11.9, random state=42, and gamma=0.01.

3.3.3. Feature selection process:

The RFE process begins by fitting the model on the training data (X_train_df, y_train). This step involves training the classifiers on the provided training dataset, which allows the models to learn and capture the patterns in the data. During this training phase, the algorithm assigns importance rankings to each feature based on their impact on the model's performance.

After obtaining the feature rankings from the model, the RFE algorithm proceeds to select the top `n_features_to_select` features. In our code, `n_features_to_select` is set to 10, indicating that we aim to identify the 10 most relevant features for the IDS. RFE achieves this by recursively eliminating the features with the lowest rankings. By iteratively removing less important features, RFE focuses on retaining the features that have the greatest influence on the model's performance.

3.3.4. Mapping and Extraction of Selected Features:

In the mapping and extraction step, we create a feature map using the `zip_longest` function from the itertools module. This function pairs each feature from our dataset with its corresponding RFE support value, which can be either True or False. The resulting feature map provides a convenient way to associate each feature with its relevance based on the RFE process.

To extract the selected features, we utilize a list comprehension `[v for i, v in feature_map if i]`. This comprehension filters the feature map based on the RFE support value, considering only those features where the support value is True. In other words, we extract the features that have been identified as important by the RFE algorithm.

The selected features, obtained through this extraction process, are considered the most important for our IDS model. They are deemed crucial based on their rankings and the support values assigned by RFE. By focusing on these selected features, we aim to enhance the performance and accuracy of our IDS by leveraging the most relevant information for intrusion detection.

3.4. Model selection

In the model selection section for our intrusion detection application, the following elaboration can be provided for the chosen models:

1. **Support Vector Machines (SVM):**

SVM is a popular choice for intrusion detection in cybersecurity due to its ability to handle high-dimensional data and capture intricate decision boundaries. It offers strong generalization capabilities and can handle both linear and non-linear tasks using kernel functions. Additionally, SVM provides interpretable results by identifying support vectors, enabling it to explain its classification reasoning for distinguishing between normal and intrusive instances.

2. **Decision Tree:**

Decision trees are highly suitable for intrusion detection as they possess the advantage of being intuitive and easily interpretable models. They excel at capturing intricate decision rules and can effectively handle a combination of numerical and categorical features. Decision trees' interpretability lies in their ability to create a hierarchical structure resembling a tree. Internal nodes within the tree represent decision rules based on specific features, while the leaf nodes correspond to classification outcomes. This transparent decision-making process enables security analysts to comprehend and explain the underlying logic behind the model's predictions.

3. **Random Forest:**

Random Forest is a highly effective ensemble model widely used in intrusion detection. By combining multiple decision trees, it can handle high-dimensional data, prevent overfitting, and capture intricate feature interactions through bagging. This ensemble capability makes Random Forest ideal for robust predictions in intrusion detection, especially when dealing with noisy or unbalanced datasets, as it can mitigate individual tree biases and improve overall performance.

4. **AdaBoost:**

AdaBoost, chosen for intrusion detection, is an ensemble learning technique that combines weak classifiers into a powerful one. It is particularly suited for addressing challenging instances that are often misclassified by weak classifiers. By iteratively training

weak classifiers on misclassified instances with increasing weights, AdaBoost creates a final strong classifier by combining their predictions. With its ensemble capabilities, AdaBoost excels at handling complex datasets, delivering high predictive performance, and providing resilience against overfitting. This makes AdaBoost a valuable and effective choice for intrusion detection tasks.

Due to inadequate accuracy, F1 score, and recall score achieved by the traditional machine learning classifiers (SVM, decision tree, random forest, and AdaBoost), we transitioned to deep learning models. This shift was motivated by the need for improved performance, and the adoption of deep learning techniques led to enhanced accuracy and F1 score, ensuring robust and accurate intrusion detection on the NSL-KDD dataset.

Our deep learning models: LSTM, GRU, RNN, Sequential are very high on accuracy and recall, f1, precision score. The results showed a 29% difference with the classical ML classifiers which implies that this change was needed.

3.5. Attack types

We used the classification report from sklearn library to view what kind of attack that each model can detect, the figure below is the classification report of our SVM model:

	precision	recall	f1-score	support
apache2	0.00	0.00	0.00	737
back	0.00	0.00	0.00	359
buffer_overflow	0.00	0.00	0.00	20
ftp_write	0.00	0.00	0.00	3
guess_passwd	0.00	0.00	0.00	1231
httptunnel	0.00	0.00	0.00	133
imap	0.00	0.00	0.00	1
ipsweep	0.23	0.26	0.25	141
land	1.00	1.00	1.00	7
loadmodule	0.00	0.00	0.00	2
mailbomb	0.00	0.00	0.00	293
mscan	0.00	0.00	0.00	996
multihop	0.00	0.00	0.00	18
named	0.00	0.00	0.00	17
neptune	0.90	0.99	0.94	4657
nmap	0.40	0.99	0.57	73
normal	0.66	0.97	0.79	9711
perl	0.09	0.50	0.15	2
phf	0.50	0.50	0.50	2
pod	1.00	0.59	0.74	41
portsweep	0.15	0.85	0.26	157
processtable	0.00	0.00	0.00	685
ps	0.00	0.00	0.00	15
rootkit	0.01	0.08	0.01	13
saint	0.00	0.00	0.00	319

Figure 7: classification report of SVM model part 1

ps	0.00	0.00	0.00	15
rootkit	0.01	0.08	0.01	13
saint	0.00	0.00	0.00	319
satan	0.59	0.96	0.73	735
sendmail	0.00	0.00	0.00	14
smurf	1.00	0.56	0.72	665
snmpgetattack	0.00	0.00	0.00	178
snmpguess	0.00	0.00	0.00	331
sqlattack	0.00	0.00	0.00	2
teardrop	0.24	1.00	0.39	12
udpstorm	0.00	0.00	0.00	2
warezclient	0.00	0.00	0.00	0
warezmaster	0.85	0.06	0.11	944
worm	0.00	0.00	0.00	2
xlock	0.00	0.00	0.00	9
xsnoop	0.00	0.00	0.00	4
xterm	0.00	0.00	0.00	13
accuracy			0.69	22544
macro avg	0.20	0.24	0.18	22544
weighted avg	0.56	0.69	0.59	22544

Figure 7: classification report of SVM model part 2

4. The conception of the application

4.1. Use case diagram

A use case diagram is a visual representation of user-system interactions, used for requirements gathering, system understanding, and stakeholder communication in software development.

The main function of use case diagram is to provide a high-level overview of the system's functionalities and how different actors interact with it.

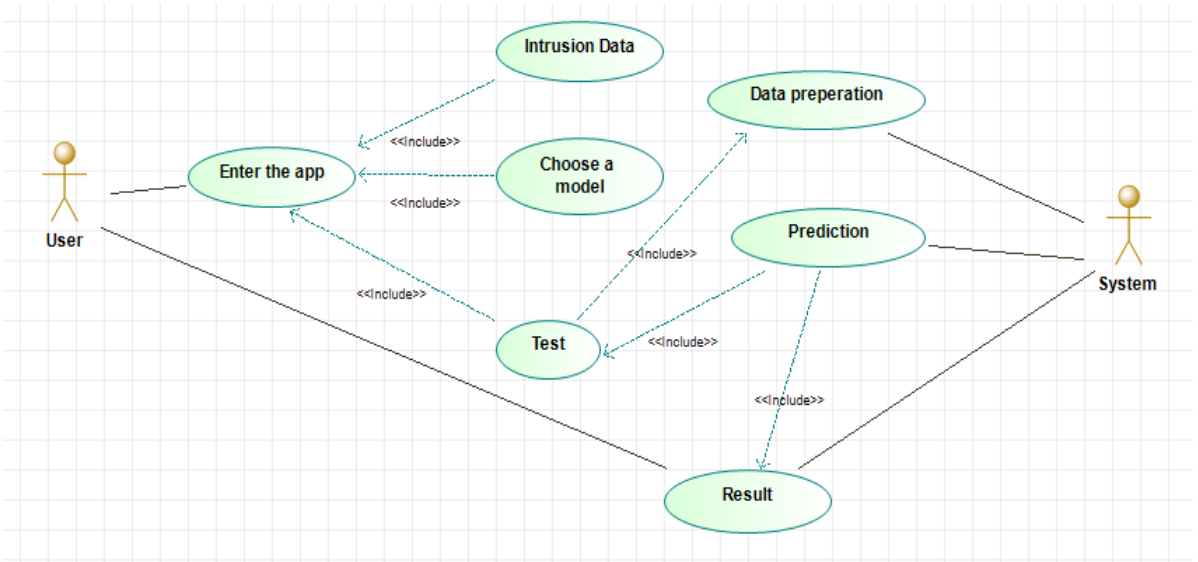

Figure 8: Use case diagram

This figure of use case diagram abstracts our application functionalities: the user can enter to the app, The user puts Intrusion data and chooses a model to detect it; after a Data preparation from the system the user can test it and the app shows the result after the prediction.

4.2. Activity diagram

An activity diagram is a graphical representation used to model the flow of activities and actions within a system or process. It focuses on the sequence of activities, their dependencies, decision points, and concurrent flows, providing a visual overview of the system's behaviour. The primary function of an activity diagram is to capture and illustrate the dynamic aspects of a system or process.

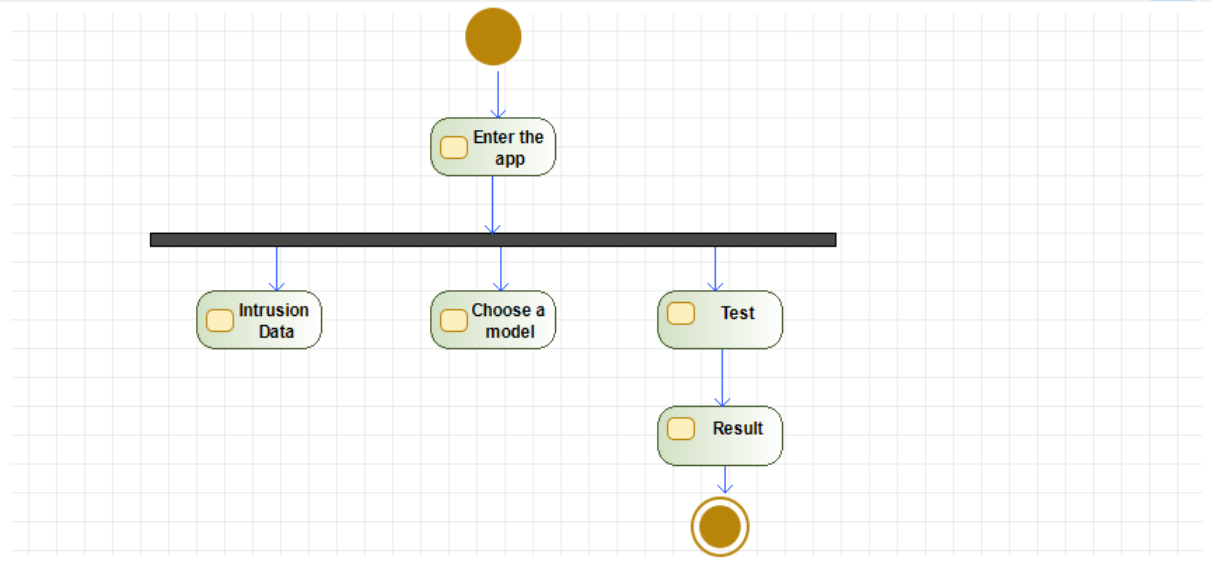

Figure 9: Activity diagram.

The activity diagram presented here provides an overview of the workflow and functionality of our application. It depicts the sequence of activities involved in the application's operation, including decision points and parallel flows.

4.3. Sequence diagram

A sequence diagram visually represents the chronological interactions and message exchanges between objects, aiding in understanding system behaviour and identifying collaborations and dependencies. It serves as a communication tool and helps validate system designs

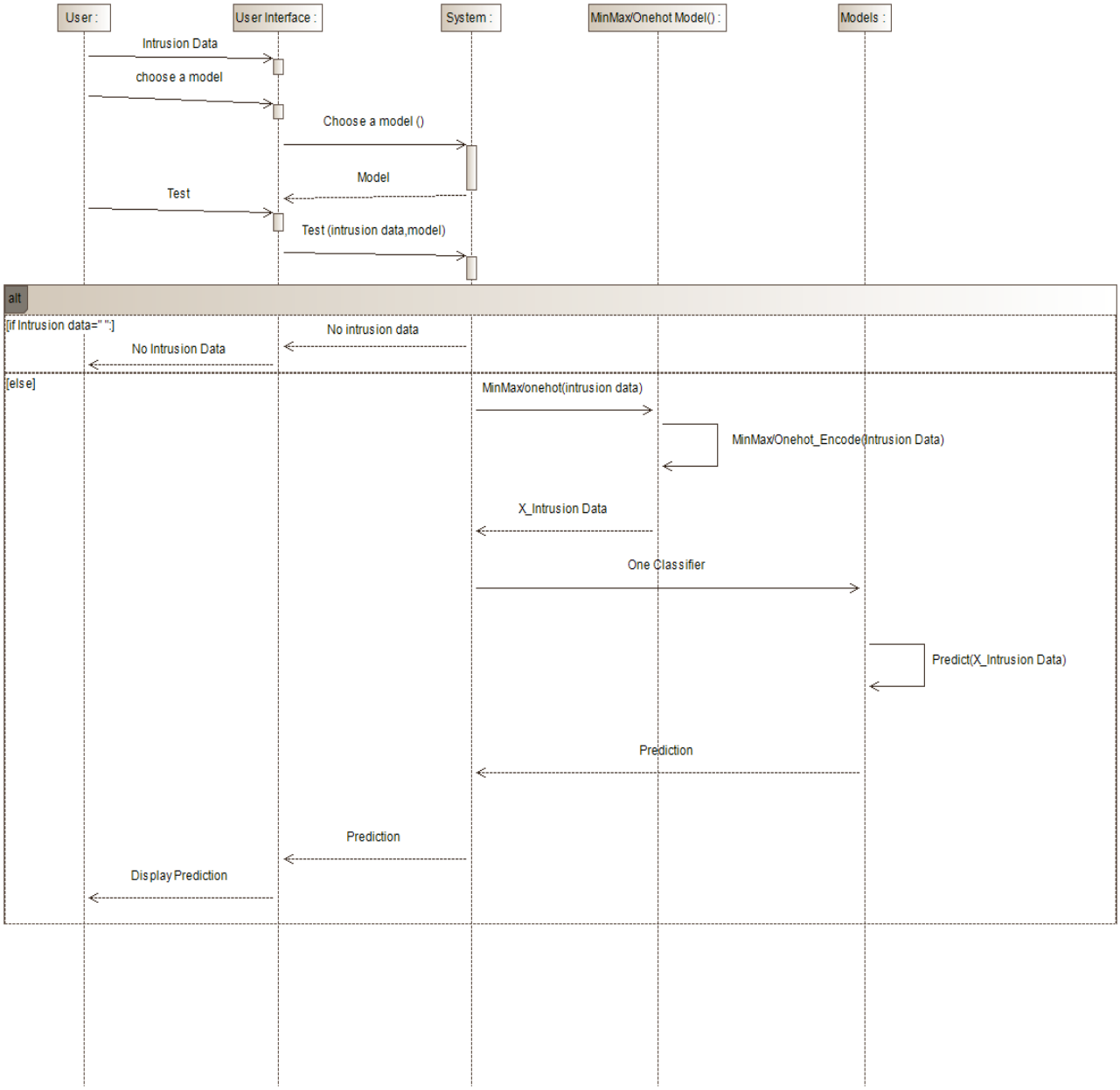

Figure 10: UML Sequence diagram of the publishing use case

From the diagram shown in Figure, the system first checks if the input area is not empty. If the area does not contain any intrusion data, it returns a message to the user. Otherwise, the system retrieves the intrusion data and the model, and it sends the intrusion data to the MinMax and Onehot model. The MinMax and Onehot model constructs the intrusion data's feature vector. The vector created by MinMax and Onehot is then sent to the selected model for analysis. Once the analysis is completed, the model returns a prediction indicating the state of the intrusion data (attack or normal). These states are displayed to the user using an appropriate user interface.

5. Conclusion

In this chapter, we have presented the conceptual steps for creating ML and DL models, as well as the essential use case, activity and sequence diagrams to understand the flow of our implemented web application. In the next chapter, we will discuss the implementation steps of our project.

Chapitre 3 : System implementation

In this chapter, we cover the technologies used in our project and why we selected them. We examine code samples to understand how our system is implemented. Lastly, we conclude with a discussion comparing our work to other relevant works in the field.

1. The used technologies

1.1. Technologies used in the model

1.1.1. The environment

We leveraged Google Colab as our development environment in conjunction with other tools. Google Colab, known for its data cleaning, statistical modeling, training machine learning models, and data visualization capabilities, played a central role in our project. We used Google Colab to preprocess and prepare our data for machine learning, deep learning, train our models, and visualize performance metrics such as accuracy and precision score, and analyze the confusion matrix, enabling us to evaluate the effectiveness of our models.

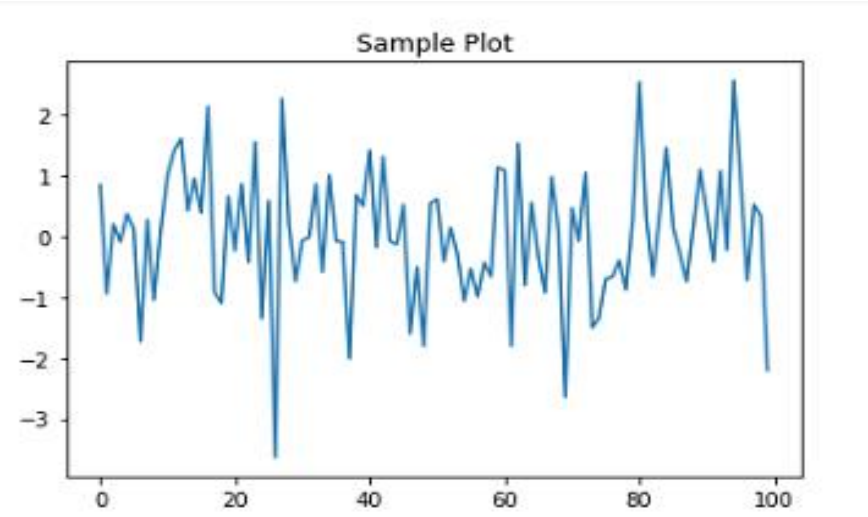

Figure 11: Example of graphical output in Google Colab[13]

1.1.2. The Programing Language and libraries

For the Classification Models, we used Python programming language employing the Pandas, NumPy, Scikit-learn, Joblib libraries also we added the Keras library for our deep learning models .

Python's simplicity and clean syntax contribute to readable code, facilitating the identification of syntactic errors. This characteristic allows developers to focus on solving algorithmic problems rather than grappling with code intricacies. Furthermore, Python's immense popularity as one of the most widely used programming languages ensures the existence of extensive online communities and abundant resources for support and knowledge sharing.

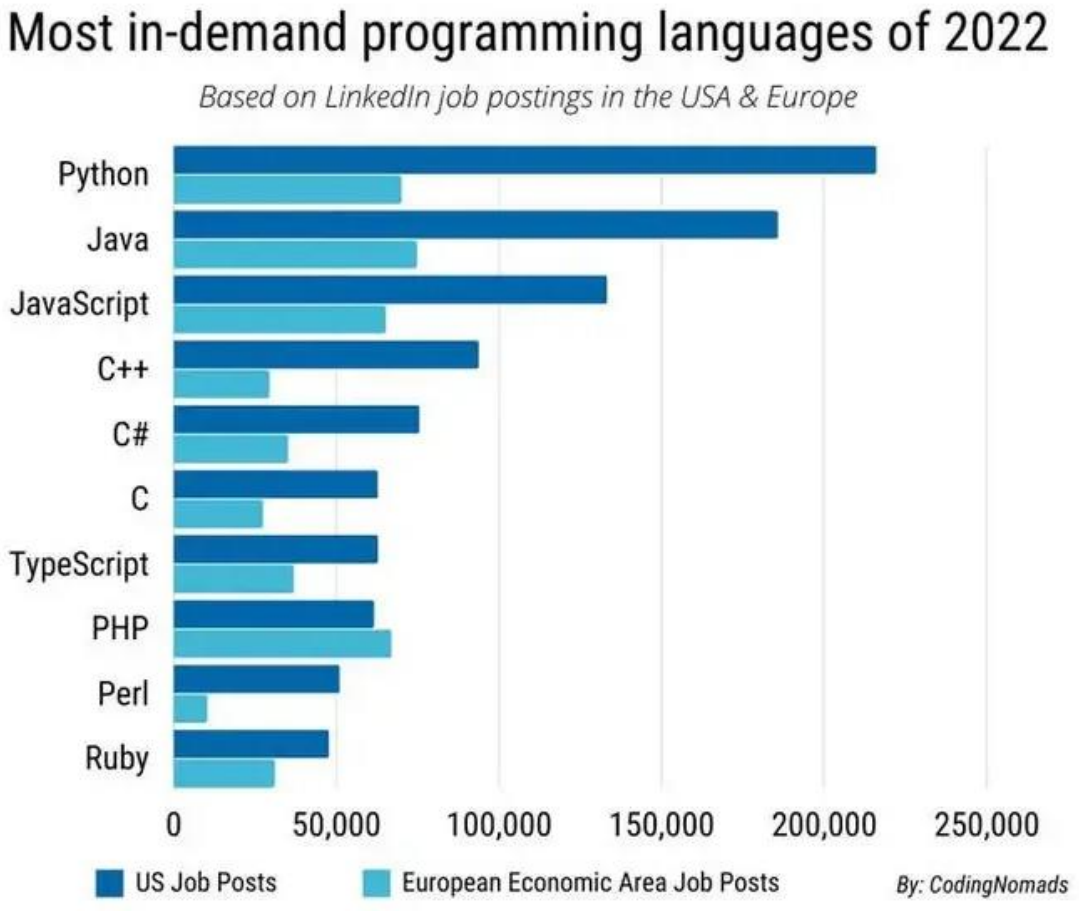

Figure 12: Most in-demand programming languages of 2022 [14]

Furthermore, python is rich of libraries especially those of data analysis and machine learning.

Pandas, NumPy, scikit-learn, and joblib are essential libraries widely used in machine learning models. Pandas provides powerful data manipulation and analysis tools, allowing efficient handling of structured data. NumPy offers efficient numerical computations and supports multidimensional arrays. Scikit-learn is a comprehensive machine learning library,

providing various algorithms and tools for tasks such as classification, regression, and clustering. Joblib enables easy serialization and deserialization of Python objects, allowing efficient model persistence and deployment. Together, these libraries form a powerful ecosystem for developing and deploying machine learning models.

1.2. The web-application

Flask served as an excellent framework for implementing our web application. Leveraging Flask's routing capabilities, we could define the necessary endpoints to handle incoming requests and interact with the application. Through the use of HTML templates and Flask's templating engine, we could dynamically generate and render the web page for users to interact with the intrusion detection system.

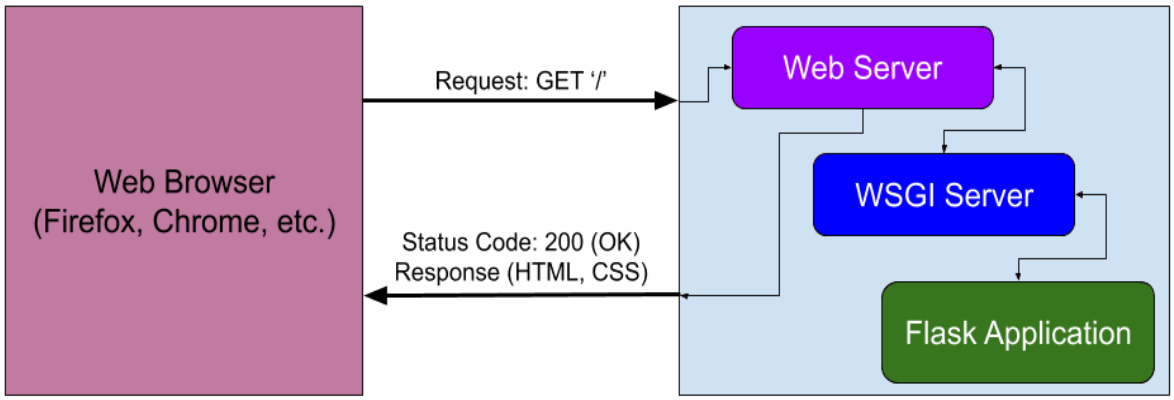

Figure 13: Web server, WSGI server, Flask app diagram [15]

For the frontend of our app, we used HTML. The use of HTML allowed us to structure and organize the content, ensuring an aesthetically pleasing and interactive user-friendly interface.

2. Application's implementation

2.1. Machine Learning models

2.1.1. The preprocessing

This code snippet demonstrates a typical preprocessing pipeline in Python, which includes encoding categorical variables using one-hot encoding and scaling numerical features using

the MinMaxScaler. These preprocessing steps are crucial for preparing the data before feeding it into a machine learning model for training or testing.

```
X_train = train_data.drop('label', axis=1)
y_train = train_data['label']
X_test = test_data.drop('label', axis=1)
y_test = test_data['label']

X_train = pd.get_dummies(X_train, columns=['protocol_type', 'service', 'flag'])
X_test = pd.get_dummies(X_test, columns=['protocol_type', 'service', 'flag'])

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

from sklearn.preprocessing import LabelEncoder
def le(df):
    df = pd.DataFrame(df)
    for col in df.columns:
        if df[col].dtype == 'object':
            label_encoder = LabelEncoder()
            df[col] = label_encoder.fit_transform(df[col])
    return df.values

le(X_train)
le(X_test)
```

Figure 14: Code snippet showing preprocessing in ML classifiers

2.2. Deep Learning models

2.2.1. The preprocessing

This code snippet involves about creating dictionaries to encode categorical features ('protocol_type', 'service', 'flag') into numerical representations. The 'map()' function is used to replace the original categorical values in the DataFrame with their corresponding numerical indices from the dictionaries. Additionally, the code utilizes the MinMaxScaler from scikit-learn to scale the numerical features within a specific range.

```
def make_dict(l):
    d = {}
    s = set(l)
    for e,i in enumerate(s):
        d[i] = e
    return d
```

```
dict_protocol = make_dict(df['protocol_type'])
dict_service = make_dict(df['service'])
dict_flag = make_dict(df['flag'])
```

```
df['flag'] = df['flag'].map(dict_flag)
df['protocol_type'] = df['protocol_type'].map(dict_protocol)
df['service'] = df['service'].map(dict_service)
```

```
from sklearn.preprocessing import MinMaxScaler

l = ['protocol_type', 'service', 'flag','level']

scaler = MinMaxScaler()

for col in df.columns:
    if col not in l:
        df[col] = scaler.fit_transform(np.array(df[col]).reshape(-1, 1))
```

Figure 15: Code snippet showing preprocessing in DL models

2.3. Feature selection

We used the Recursive Feature Elimination (RFE) method for feature selection, implemented through the scikit-learn library. The RFE algorithm selects a subset of features that are most relevant to the target variable. The following code snippet demonstrates the feature selection process:

```
from sklearn.feature_selection import RFE
import itertools

svm_clf = SVC(kernel='linear', C=11.9, random_state=42, gamma=0.01)
svm_clf.fit(X_train_df, y_train)

rfe = RFE(svm_clf, n_features_to_select=10)
rfe = rfe.fit(X_train_df, y_train)

feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.support_, X_train_df.columns)]
selected_features = [v for i, v in feature_map if i]
```

Figure 16: Code snippet showing feature selection in ML models

2.4. Model evaluation

To evaluate our models, we used evaluation metrics: accuracy, recall score, f1 score, precision score, adding loss function for the DL models.

For the accuracy we used different codes for each one of our machine learning and deep learning models.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))
```

Figure 17: Code snippet showing the accuracy and confusion matrix in ML models

```
met = model.evaluate(X_test, y_test)
print("Accuracy : ",met[1])
print("Loss : ",met[0])
```

Figure 18: Code snippet showing the accuracy and loss in DL models

The following code snippet demonstrates the usage of scikit-learn's evaluation metrics functions to assess the performance of a classification model by calculating the F1 score, precision score, and recall score as the same code was used for both ML and DL models. These metrics provide insights into the models' ability to correctly classify instances and capture both precision (the proportion of correctly predicted positive instances) and recall (the proportion of actual positive instances correctly classified).

```
from sklearn.metrics import f1_score, precision_score, recall_score
f= f1_score(y_test, y_pred,average="micro")
p= precision_score(y_test, y_pred,average="weighted",zero_division=1.0)
r= recall_score(y_test, y_pred,average="weighted",zero_division=1.0)
print("f1_score = ",f)
print("Precision Score = ",p)
print("recall score = ",r)
```

Figure 19: Code snippet showing the recall, precision and F1 score.

NB. Please check Annex A for more information about the used evaluation metrics.

The following table demonstrates the results of each evaluation metric for the machine learning models

Model	Accuracy	Recall score	Precision score	F1 score
SVM	68%	68%	80%	68%
Random Forest	70%	70%	80%	70%
Decision Tree	65%	65%	67%	65%
AdaBoost	62%	62%	76%	62%

Table 1: Evaluation metrics results for ML models

The following table shows the results of each evaluation metric and the loss function for the deep learning models.

Model	Accuracy	Lost	Recall score	Precision score	F1 score
Sequential	0,98	0,02	98%	98%	98%
LSTM	0,99	0,03	99%	99%	99%
RNN	0,95	0,11	95%	95%	95%
GRU	0,98	0,04	98%	98%	98%

Table 2: Evaluation metrics + loss results for DL models

These graphs show the accuracy difference between our models, this comparison was made to evaluate the performance.

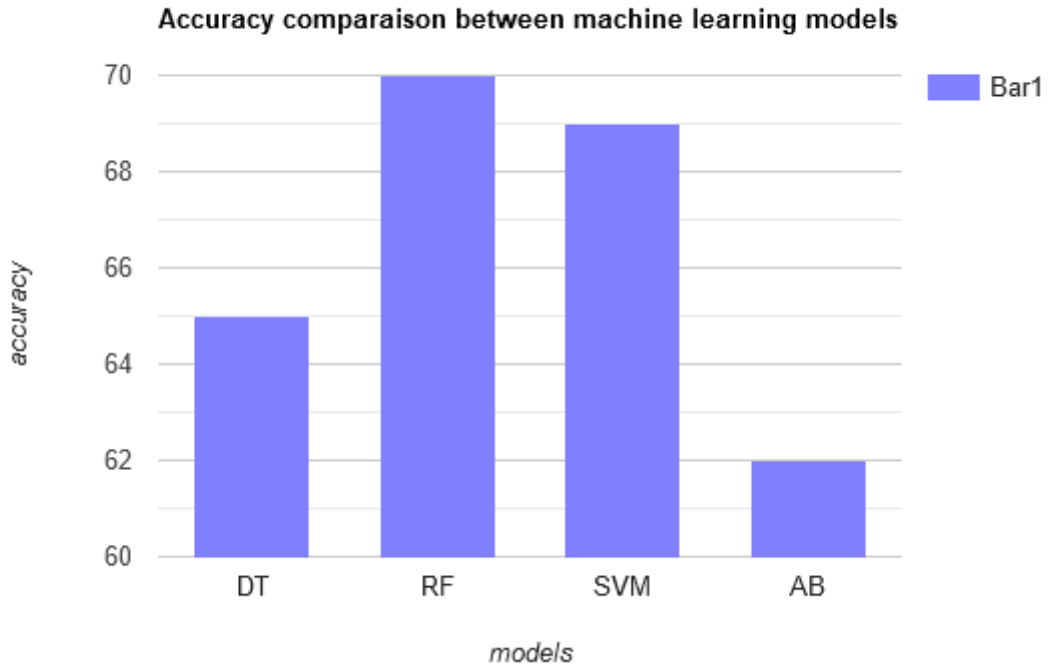

Figure 20: Accuracy comparison between machine learning models

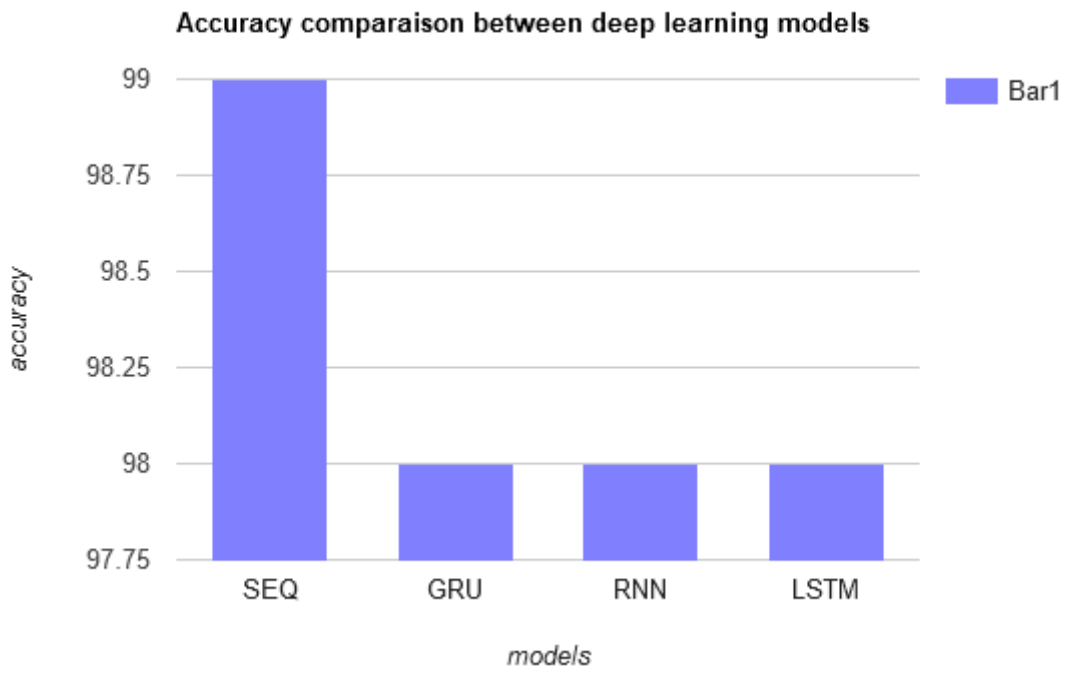

Figure 21: Accuracy comparison between deep learning models

3. Application's presentation

Our user interface (UI) is designed to provide a seamless and intuitive experience as one navigates through its features and functionalities. With a modern and clean design, we've created an interface that is both visually appealing and user-friendly.

Figure 22: Presentation of the app

- Enter script to start scanning here:

Figure 23: Enter Intrusion Data to start scanning

- Select model

Figure 24 : Select Model

-Test button:

Figure 25: Test button

- Result is shown here:

Figure 26 : Result

4. Discussion and conclusion

The aim of this study was to develop an effective intrusion detection system using the NSL-KDD dataset. The medium accuracy, F1 score, precision, and recall scores achieved by the trained ML models was found to be satisfactory.

One notable aspect of our study was the need to implement certain deep learning models. These models, known for their ability to extract complex features from raw data, showed significant improvements in the evaluation metrics compared to traditional machine learning algorithms. The deep learning models leveraged their hierarchical representations and learned patterns to enhance the detection of intrusions, leading to improved accuracy, F1 score, precision, and recall scores.

This study demonstrates the potential of machine learning models, particularly deep learning approaches, in building effective intrusion detection systems. The continuous advancement in machine learning techniques holds promise for enhancing the security of network infrastructures and protecting against emerging threats.

General Conclusion and perspectives

The goal behind this project was to build a smart Intrusion Detection System model using Machine Learning classifiers that can detect any unauthorized access or activity in a computer network or system. Especially that the number and complexity of cyber threats has increased with the daily use of technology which threatens the security and integrity of network systems.

Developing a Machine Learning based Intrusion Detection System (IDS) can be a challenging task due to several factors. First and foremost, IDS models require a significant amount of labeled data to train accurately, which can be time-consuming and resource-intensive to obtain. Additionally, selecting the appropriate features and algorithms for the model can be a complex process, as different datasets and use cases may require different approaches. Developing an effective ML-based IDS model cannot be done in this short period of time and what made it harder is the state of the current hardware which led us to face a lot of errors and problems with increasing the accuracy, that's why adding a few deep learning models to the application was needed.

Our model can detect and classify various types of network attacks by analyzing network traffic data, including R2L, U2R, DOS, Probe attacks. Based on Machine/Deep Learning algorithms (SVM, Random Forest, AdaBoost, Decision tree, Sequential, GRU, LSTM, RNN), and trained on the « NSL-KDD » dataset from Kaggle. This model was fully developed in python.

We could say that the current model is in constant need of improvement because it has limitations and cannot detect all types of attacks accurately and, for a subject like network security it is important to continuously evaluate and improve IDS models to ensure that they are providing effective security for the network infrastructure.

For our application, it is simple to use although we think there are some functionalities that can be added like selecting the file type for file analysis etc,.

References

[1] Figure 1 : https://networksimulationtools.com/intrusion-detection-system-projects/

[2] https://www.helixstorm.com/blog/types-of-intrusion-detection-systems/

[3] Figure 2:

https://www.researchgate.net/figure/Detection-techniques-used-by-IDS_fig1_332625093

[4] Figure 3 :

https://medium.com/ml-research-lab/machine-learning-algorithm-overview-5816a2e6303

[5] https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps

[6]https://www.techtarget.com/searchenterpriseai/definition/recurrent-neural-networks#:~:text=Recurrent%20neural%20networks%20recognize%20data's,activity%20in%20the%20human%20brain.

[7]https://intellipaat.com/blog/what-is-lstm/?US#:~:text=LSTM%20stands%20for%20long%20short,especially%20in%20sequence%20prediction%20problems.

[8]https://blog.marketmuse.com/glossary/gated-recurrent-unit-gru-definition/#:~:text=The%20Gated%20Recurrent%20Unit%20(GRU,using%20datasets%20with%20longer%20sequences.

[9] https://towardsdatascience.com/a-comprehensive-introduction-to-tensorflows-sequential-api-and-model-for-deep-learning-c5e31aee49fa

[10] Figure 4 : https://www.researchgate.net/figure/Comparison-of-training-time-and-testing-time-a-d-in-NSL-KDD-and-KDD-Cup-99-dataset_fig5_348445888

[11] Figure 5 : https://developer.ibm.com/articles/cc-models-machine-learning/

[12]Figure6 :https://www.researchgate.net/figure/General-architecture-of-deep-learning_fig1_348357543

[13]Figure10 : https://www.tutorialspoint.com/google_colab/google_colab_graphical_outputs.htm

[14] Figure 11 : https://invozone.com/blog/top-10-programming-languages/

[15] Figure 12 : https://testdriven.io/blog/flask-contexts-advanced/

Annex A

1. Deep learning

1.1. Definition

Deep learning refers to a subset of machine learning that involves training artificial neural networks with multiple layers to automatically learn and extract intricate patterns and representations from complex data.

1.2. Famous Deep Learning models

- **Recurrent neural networks (RNNs) :** Recurrent neural networks RNNs are used in deep learning and in the development of models, it recognize data's sequential characteristics and use patterns to predict the next likely scenario.[6]
- **Long Short-Term Memory networks (LSTM):** used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems.[7]
- **The Gated Recurrent Unit (GRU):** is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short-term memory (LSTM). GRU uses less memory and is faster than LSTM.[8]
- **Sequential:** The sequential model allows us to specify a neural network, precisely, sequential: from input to output, passing through a series of neural layers, one after the

other. TensorFlow also allows us to use the functional API for building deep learning models.[9]

2. Evaluation metrics

- **Precision and recall score:** Precision and Recall are machine learning performance indicators that evaluate the accuracy of positive and negative predictions made by a model. Precision focuses on the proportion of correct positive predictions out of all positive predictions, while recall emphasizes the proportion of correct positive predictions out of the combined number of true positives and false negatives. Both metrics are represented by values between 0 and 1 and are crucial for assessing the predictive quality of a model.
- **F1 score:** The F1-score is a metric that combines precision and recall into a single measure by calculating their harmonic mean. It is frequently used to compare the performance of two classifiers. When one classifier has higher recall and another has higher precision, comparing their F1-scores helps determine which classifier yields better overall results. By considering both precision and recall, the F1-score provides a balanced evaluation of a classifier's performance.
- **Accuracy:** Accuracy is a widely known and frequently used evaluation metric. It simply measures the proportion of correct predictions made by a model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

TP: True Positives

TN: True Negatives

FP: False Positives

FN: False Negatives

Summary

As our reliance on technology grows, cyberattacks have become a common threat. These attacks involve exploiting weaknesses in computer systems. To combat these malicious activities, Intrusion Detection Systems (IDS) have been developed as security measures to identify and warn for possible cyberattacks.

Intrusion detection Systems are necessary because of the increasing frequency and sophistication of cyber-attacks, which can result in serious consequences such as data theft, data loss, financial losses etc. It can also help detect and mitigate all of these types of attacks by monitoring network traffic and identifying patterns or behaviors associated with known attack signatures or unusual activity.

The goal of this project is to build a machine learning model that can detect various types of attacks including Denial-Of-Service (DOS), User-To-Root(U2R), Remote-To-Local(R2L), Probe, malicious infections, etc, Although the chosen ML classical trained models: SVM, Random Forest, AdaBoost, Decision tree, showed acceptable results, the DL models: Sequential, GRU, LSTM, RNN, obtained far best scores. The models were trained on the « NSL-KDD » Kaggle dataset .

Key words: Malwares, Intrusion Detection System(IDS), Denial-Of-Service attacks(DOS), User-To-Root(U2R), Remote-To-Local(R2L), Probe, Machine Learning, Classifiers..

الملخص

مع نمو اعتمادنا على التكنولوجيا ، أصبحت الهجمات الإلكترونية تهديدًا مشتركًا. تتضمن هذه الهجمات استغلال نقاط الضعف في أنظمة الكمبيوتر. لمكافحة هذه الأنشطة الخبيثة ، تم تطوير أنظمة كشف التسلل (IDS) كإجراءات أمنية لتحديد والتحذير من الهجمات الإلكترونية المحتملة.

الهدف من هذا المشروع هو بناء نموذج للتعلم الآلي يمكنه اكتشاف أنواع مختلفة من الهجمات بما في ذلك (DOS) و User-To-Root (U2R) و Remote-To-Local (R2L) و Probe و Malicious.

تعد أنظمة اكتشاف التسلل ضرورية بسبب زيادة وتيرة الهجمات الإلكترونية وتعقيدها .

والتي يمكن أن تؤدي إلى عواقب وخيمة مثل سرقة البيانات وفقدان البيانات والخسائر المالية وما إلى ذلك ، ويمكن أن تساعد أيضًا في اكتشاف كل هذه الأنواع من الهجمات وتخفيفها من خلال المراقبة حركة مرور الشبكة وتحديد الأنماط أو السلوكيات المرتبطة بتوقيعات هجوم معروفة أو نشاط غير عادي.