

A Project Report

On

MALWARE ANALYSIS

BY

TANISHA RANA – SE20UCSE202

IMAZ SURVE- SE20UCSE098

JAHNAVI SIRIPURAPU – SE20UCSE178

ROHAN REDDY- SE20UARI098

AGASTYA GANDRA – SE20UARI011

DISHA S - SE20UCAM013

Under the supervision of

RAGHU KISHORE NEELISETTI

**SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF
PR301: PROJECT TYPE COURSE**



**MAHINDRA UNIVERSITY
HYDERABAD**

(JUNE 2023)

ACKNOWLEDGMENT

This project has been an invaluable learning experience, allowing us to expand our knowledge in various unfamiliar domains and gain insights into solving real-world challenges in this specific field. However, the realization of this project would not have been possible without the unwavering support and guidance of numerous individuals. We would like to express our heartfelt gratitude to all those who have contributed.

First and foremost, we extend our deepest appreciation to Professor N. Raghu Kishore for his constant motivation and guidance throughout the project. His valuable insights and provision of necessary information and resources have been instrumental in our progress. We are immensely grateful for his unparalleled and timely support, which has been pivotal in the successful completion of this project. His extensive knowledge, profound experience, and professional expertise have been invaluable assets.

We would also like to extend our thanks to all those who have generously offered their assistance and expertise. Their willingness to lend a helping hand has been crucial in enhancing the quality and depth of our work.

In conclusion, we acknowledge the indispensable contributions of everyone involved, as their support has played a pivotal role in the success of this project.



Mahindra University

Global Thinkers. Engaged Leaders.

Mahindra University

Hyderabad

Certificate

This is to certify that the project report entitled “Malware Analysis” submitted by Ms. Tanisha Rana(SE20UCSE202), Mr. Imaz Surve(SE20UCSE098), Ms. Jahnvi Siripurapu(SE20UCSE178), Rohan Reddy(SE20UARI098), Mr. Agastya Gandra(SE20UARI011) and Ms. Disha S (SE20UCAM013) in partial fulfillment of the requirements of the course PR301, Project Course, embodies the work done by him/her under my supervision and guidance.

(Raghu Kishore Neelisetti & Signature)

Mahindra University, Hyderabad.

Date:

(External Examiner& Signature)

Mahindra University, Hyderabad.

Date:

ABSTRACT

This research project focuses on the analysis of malware, addressing the growing threat it poses to computer systems and networks. By employing advanced techniques such as deep learning, machine learning, and behavioural analysis, the project aims to gain insights into the nature, behaviour, and impact of malware.

The project begins with an exploratory analysis phase, where features are extracted and selected to identify patterns and characteristics of malware samples. Statistical methods and data visualization techniques are utilized to uncover hidden relationships within the dataset.

Dynamic analysis is then performed to simulate malware execution and observe its runtime behaviour and interactions with the system. System calls, network traffic, and API invocations are closely monitored to understand the intentions, capabilities, and potential system impact of the malware.

Behavioural analysis is employed to study the long-term behaviour of malware samples, with a focus on identifying persistence mechanisms, communication channels, and propagation strategies. Machine learning algorithms are leveraged, enabling the classification and clustering of malware based on behavioural characteristics, facilitating the identification of malware families and variants.

The project also explores the application of multi-layer perceptron (MLP), a powerful deep learning model, to enhance the accuracy and effectiveness of malware analysis. The MLP is trained on extracted features and behaviour patterns, enabling it to learn complex relationships and detect subtle patterns in malware data, ultimately improving malware detection, classification, and identification of new threats.

By employing advanced techniques and methodologies, this research project aims to advance the field of malware analysis and contribute to improved cybersecurity. The insights gained will help in developing robust tools and frameworks for effective detection, classification, and mitigation of malware, bolstering the security of computer systems and networks against evolving threats.

CONTENTS

Title page.....	1
Acknowledgements.....	2
Certificate.....	3
Abstract.....	4
1.Introduction.....	6
2.Problem Definition.....	9
3. Implementation.....	11
4.1) Dataset.....	11
4.2) Exploratory Analysis.....	12
4.3) Dynamic Analysis.....	14
4.4) Behavioural Analysis.....	16
4.5) Multi-Layer Perceptron.....	18
Result.....	19
Conclusion.....	19
References.....	20

INTRODUCTION

2.1 What is malware

Malware is a kind of intrusive software that damages and destroys computer systems, servers, host systems, or networks. It is a catch-all term for all types of malicious software that is specifically intended to cause damage or exploit any programmable device, network, or service. Viruses, worms, adware, spyware, trojan viruses, and ransomware are various types of malware threats.

2.2 What Is Malware Analysis?

Malware analysis is the process of examining and understanding malicious software, to gain insights into its behaviour, functionality, and potential impact on computer systems, networks, and users. The primary goal of malware analysis is to identify and mitigate the threats posed by malware, as well as to develop effective strategies for detecting, analysing, and responding to new and evolving malware threats.

Malware analysis involves various techniques and methodologies to uncover the inner workings of malware, including its code, propagation methods, communication channels, and malicious payloads. The analysis process helps security professionals and researchers understand the intentions of malware authors, the techniques employed to evade detection, and the potential vulnerabilities it exploits.

2.3 What did we do in this project?

We conducted a malware analysis on API call sequence dataset. The analysis involves advanced deep learning and machine learning techniques such as exploratory analysis, dynamic analysis, behavioural analysis and multi-layer perceptron to gain insights into the nature, behaviour, and impact of malware.

2.4 Why did we chose this project?

The topic of malware analysis is crucial in the field of cybersecurity. Malware poses significant threats to individuals, organizations, and even nations, leading to financial losses, data breaches, and disruptions to critical systems. By choosing this project, the group aims to contribute to the understanding and mitigation of these threats. Malware analysis allows us to delve into the intricate details of malicious software, deciphering its intentions, identifying vulnerabilities, and developing effective countermeasures.

malware analysis also enables us to stay at the forefront of emerging threats and techniques employed by cybercriminals. By analysing the latest malware samples, we can uncover new attack vectors, zero-day vulnerabilities, and sophisticated evasion mechanisms. This knowledge is vital for developing proactive defence strategies, improving security solutions, and enhancing overall cybersecurity posture. Furthermore, malware analysis provides an opportunity to explore cutting-edge technologies and methodologies in the field of cybersecurity.

2.5 Importance of Malware Analysis

Malware analysis plays a crucial role in the field of cybersecurity by providing valuable insights and contributing to various aspects of defence against cyber threats. It helps in identifying and classifying different types of malwares, which enables security professionals to develop effective countermeasures and protection strategies. Additionally, malware analysis supports incident response efforts by providing information about the infection vector, propagation methods, and malicious activities associated with the malware. This knowledge aids in containing incidents, mitigating their impact, and recovering compromised systems.

Furthermore, malware analysis helps in discovering vulnerabilities and weaknesses in software and infrastructure that are exploited by malware. This information assists in patching and improving system security, reducing the risk of future attacks. Malware analysis is also used to detect and block known malware, enhancing the overall defence against infections, enhancement of security solutions, Understanding Attack Techniques and Tendencies. Etc...

2.6 There are different types of malware analysis.

1. **Static Analysis:** Static analysis involves examining the malware without executing it. This includes analysing the malware's code, file headers, metadata, and other characteristics to identify patterns, signatures, and potential indicators of malicious behaviour.
2. **Dynamic Analysis:** Dynamic analysis involves executing the malware in a controlled environment, such as a virtual machine or sandbox, to observe its behaviour in real-time. This includes monitoring system activities, network communications, file system modifications, and interactions with other processes.
3. **Behavioural Analysis:** Behavioural analysis focuses on understanding the actions and impact of malware on a system. It involves monitoring the behaviour of malware samples, such as processes spawned, files created or modified, registry modifications, network traffic generated, and system changes.
4. **Code Analysis:** Code analysis involves examining the malware's code to understand its internal workings, logic, and potential vulnerabilities. This can include disassembling or decompiling the malware to gain insights into its functionality and algorithms.
5. **Network Traffic Analysis:** Network traffic analysis involves capturing and analysing network communications generated by malware. This helps in identifying command-and-control servers, communication protocols, data exfiltration attempts, and potential network-based indicators of compromise.

2.7 Machine Learning techniques for Malware Analysis:

2.7.1 K-nearest neighbours: -

K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm used for classification and regression tasks. It works based on the principle of finding the "k" nearest data

points in the training set to a given test data point and making predictions based on the majority vote or averaging of their labels or values. KNN is a non-parametric algorithm, meaning it does not make any assumptions about the underlying data distribution. It is easy to understand and implement, but can be computationally expensive for large datasets it is also versatile and can handle both numerical and categorical data.

2.7.2 naive-bayes algorithm: -

Naive Bayes is a probabilistic machine learning algorithm commonly used for classification tasks. It is based on Bayes' theorem, which calculates the probability of a hypothesis given the observed evidence. The "naive" assumption in Naive Bayes is that all features are independent of each other, even though this assumption may not hold. Despite this simplifying assumption, Naive Bayes performs well in many real-world scenarios and is computationally efficient. It is particularly effective when working with large feature spaces and is often used in text classification, spam filtering, and sentiment analysis tasks. Naive Bayes is known for its simplicity, interpretability, and ability to handle both numerical and categorical data.

2.7.3 Decision Trees: -

Decision trees are a machine learning algorithm that uses a flowchart-like structure to make predictions. Each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. The algorithm learns by selecting the best features to split the data and create homogeneous subsets. Decision trees are easy to understand, can handle various data types, and capture complex relationships. However, they can overfit and may not generalize well. Ensemble methods are often used to improve performance. Decision trees are used in many domains for tasks like segmentation and risk assessment.

2.8 Deep Learning techniques for Malware Analysis:

In this project we discuss about Deep learning techniques to identify and protect against all types of malware threats, with a multiclass classification model for maximum detection accuracy and accurate predictions of future infections.

Multilayer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected nodes. It is widely used for classification, regression, and pattern recognition tasks. MLP learns complex relationships in the data by adjusting weights and biases during training. It uses hidden layers to extract abstract representations from the input data. MLP has been successful in various domains but may suffer from overfitting and to overcome this issue Regularization techniques are used.

PROBLEM DEFINITION

Malware threats continue to pose a significant risk to computer systems and networks, necessitating effective identification and protection mechanisms. This project aims to leverage deep learning and machine learning techniques to address this challenge by developing a multiclass classification model capable of accurately identifying and defending against diverse types of malware threats. The focus will be on utilizing the API call sequence malware dataset, which captures the sequential API calls made by malware samples during execution. The primary objective is to achieve maximum detection accuracy and make precise predictions regarding future malware infections.

Key Questions:

1. Can deep learning and machine learning techniques be effectively employed to classify various types of malwares based on their API call sequences?
2. How can a multiclass classification model be developed to accurately identify and protect against a wide range of malware threats?
3. What is the performance comparison between different machine learning models, including K-Nearest Neighbors (KNN), Naive Bayes, and Decision Tree, in classifying malware based on API call sequences?
4. How does the performance of a deep learning model, specifically the Multi-Layer Perceptron (MLP), compare to traditional machine learning models in terms of detection accuracy and the prediction of future malware infections?

Mathematical Representation:

Let's denote the API call sequence malware dataset as X , where each sample x_i represents a sequence of API calls made by a malware sample during its execution. The goal of the project is to learn a multiclass classification function f that maps the input dataset X to their corresponding malware types y_i . Here, y_i belongs to a predefined set of malware classes.

To achieve this, the project will utilize deep learning and machine learning techniques, including the Multi-Layer Perceptron (MLP) as a deep learning model and traditional machine learning models such as K-Nearest Neighbors (KNN), Naive Bayes, and Decision Tree.

The training process involves optimizing the parameters of the classification function f to minimize the classification error between the predicted outputs and the ground truth labels. This can be formulated as an optimization problem, where the objective is to find the optimal values for the parameters that minimize a specific loss function.

For example, in the case of the MLP model, the classification function f can be represented as:

$$f(W, b) = y_i$$

where x_i is the input sequence of API calls for a given malware sample, W represents the weights of the MLP's hidden layers, b represents the biases, and y_i is the predicted output, indicating the probability distribution over the possible malware classes.

The training process involves iteratively adjusting the weights and biases to minimize the difference between the predicted output and the true label using techniques such as backpropagation and gradient descent. This process aims to find the optimal values for the model's parameters that result in accurate predictions for malware classification.

Once the model is trained, it can be used to predict the malware class for new, unseen API call sequences. The model's prediction is obtained by applying the learned weights and biases to the input sequence, resulting in the estimated probabilities for each malware class. The class with the highest probability is selected as the predicted malware type.

To evaluate the performance of the models, various metrics can be utilized, including accuracy, precision, recall, and F1 score. These metrics assess the model's ability to correctly classify malware samples and make accurate predictions for future infections.

By employing these mathematical representations and utilizing deep learning and machine learning techniques, this project aims to develop an effective multiclass classification model that can accurately identify and protect against various types of malware threats based on API call sequences.

IMPLEMENTATION

3.1) DATASET

The dataset we used is “**Malware Analysis Datasets: API Call Sequences**”, this dataset contains 42,797 malware API call sequences and 1,079 goodware (non-malicious software) API call sequences. Each API call sequence in the dataset is derived from the 'calls' elements of Cuckoo Sandbox reports. These sequences capture the first 100 non-repeated consecutive API calls associated with the parent process. This type of dataset is valuable for analysing and detecting malware based on the patterns and sequences of API calls. By examining the API calls made by different software, researchers and analysts can gain insights into the behaviour and characteristics of malware, which can aid in the development of more effective detection and prevention mechanisms.

Here is a snapshot of how the dataset looks,

hash	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_10	t_11	t_12	t_13	t_14	t_15	t_16	t_17
071e8c3f8	112	274	158	215	274	158	215	298	76	208	76	172	117	172	117	172	76	117
33f8e6d08	82	208	187	208	172	117	172	117	172	117	172	117	172	117	172	117	172	117
b68abd06	16	110	240	117	240	117	240	117	240	117	240	117	240	117	172	117	99	260
72049be7	82	208	187	208	172	117	172	117	172	117	172	117	172	117	172	117	172	117
c9b3700a	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274
cc6217be8	117	208	117	208	117	240	117	240	117	208	228	215	274	158	215	274	158	215
f7a1a3c38	215	274	158	215	274	158	215	172	117	172	117	172	117	198	208	260	257	25
164b5652	82	240	117	240	117	240	117	240	117	240	117	172	117	172	117	16	31	86
56ae1459f	82	240	117	240	117	240	117	240	117	240	117	16	208	187	208	240	117	39
c4148ca9f	82	208	187	208	172	117	172	208	16	208	240	117	240	117	82	112	123	65
fb7569d1c	172	117	208	76	274	158	215	274	158	215	76	215	76	172	117	172	117	286
e7ac6a2d4	82	240	117	240	117	93	117	172	117	16	117	215	228	208	240	117	82	198
12828373	82	172	117	16	294	94	215	274	158	215	274	158	215	94	208	274	158	215
2688d034f	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274
2109cd663	82	240	117	240	117	240	117	240	117	172	117	172	117	16	240	117	11	274
96bb462a	240	117	240	117	240	117	228	208	187	208	172	117	172	117	93	208	172	117
2a1e576d	286	110	172	240	117	240	117	240	117	106	171	260	141	65	260	141	65	260

FIGURE 1: API CALL SEQUENCE DATASET

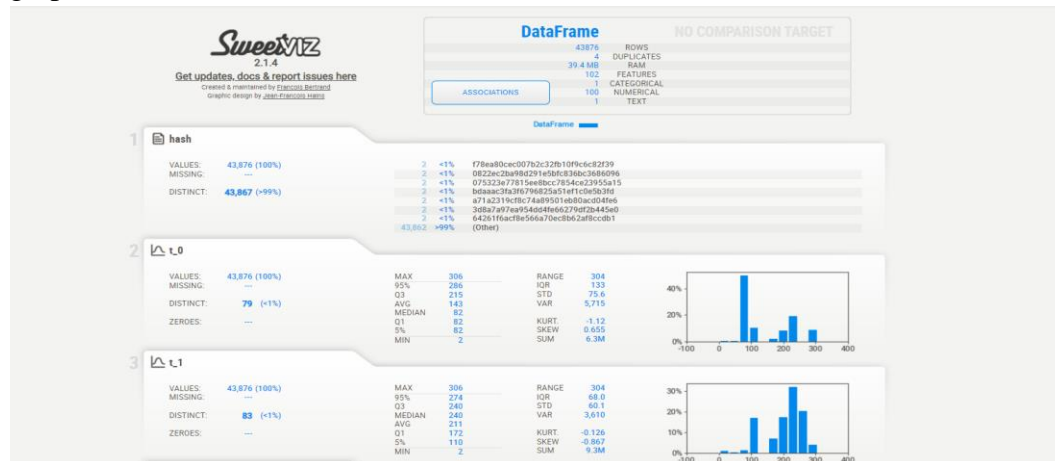
4.2) EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA) is an approach used to analyse and summarize data sets to understand their main characteristics, patterns, and relationships. The primary goal of EDA is to gain insights and generate hypotheses, which can then be further tested and refined using more advanced statistical techniques.

Libraries used for EDA:

1. Pandas: Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrames, which are useful for handling and exploring tabular data.
2. NumPy: NumPy is a fundamental library for numerical computing in Python. It provides support for handling arrays and mathematical operations, which are essential for data analysis.
3. Matplotlib: Matplotlib is a widely used plotting library in Python. It offers a variety of plotting options to visualize data, including histograms, scatter plots, line plots, and more.
4. Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides high-level interfaces for creating visually appealing and informative statistical

graphics.



SWEETVIZ FOR EDA

Now, let's look at the different plots we have used for the project:

1) Distribution Graph function:

We defined a function that takes a DataFrame, the number of graphs to show, and the number of graphs to display per row as inputs. The function aims to plot the distribution of each column in the DataFrame. The purpose of this function is to provide a quick way to visualize the distribution of each column in a DataFrame, helping to gain insights into the data and identify any patterns or anomalies.

Distribution graph

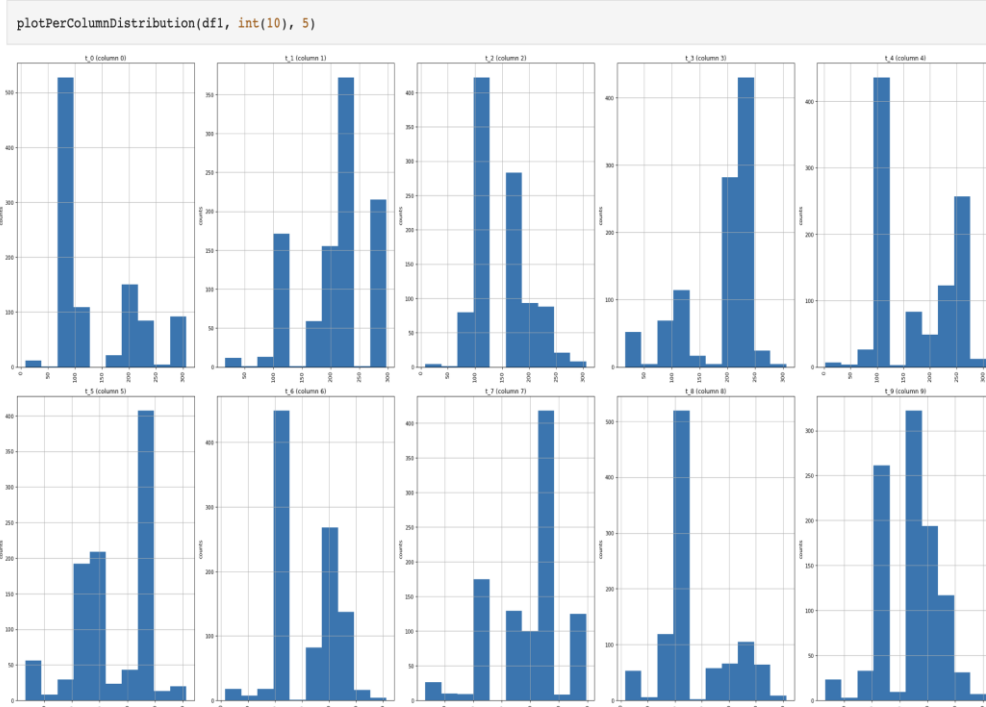


FIGURE 2: DISTRIBUTION GRAPH

2) Correlation Matrix:

We defined a function that takes a DataFrame and the width of the graph as inputs. The function aims to plot a correlation matrix for the columns in the DataFrame. The purpose of this function is to visualize the correlation between columns in a DataFrame, helping to identify relationships and dependencies between variables. The correlation matrix provides a visual representation of the pairwise correlation coefficients, where higher values indicate stronger positive or negative correlations.

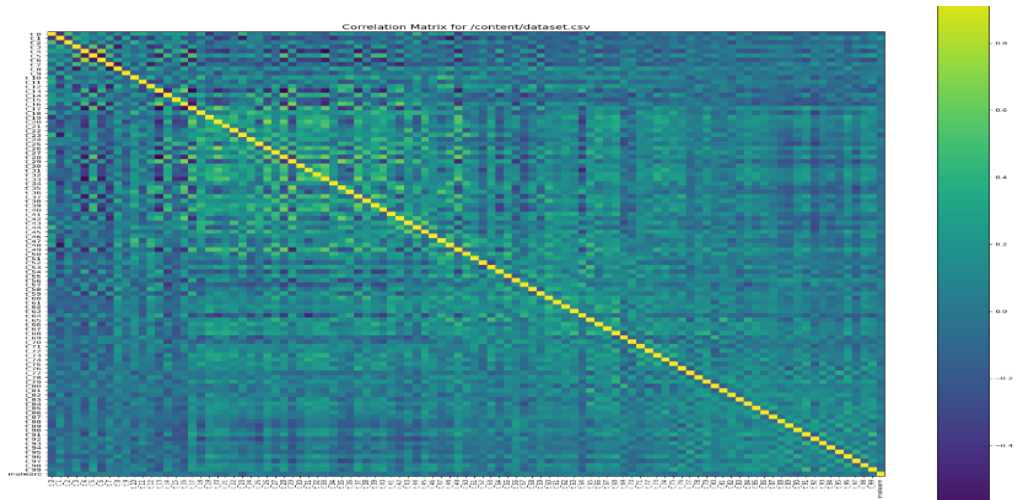


FIGURE 3: CORRELATION MATRIX

3) Scatter and Density Plot:

We defined a function that takes a DataFrame , the plot size, and the text size as inputs. The function aims to create a scatter matrix plot with kernel density estimates. The purpose of this function is to create a scatter matrix plot that shows the relationships between pairs of numerical variables in a DataFrame. The diagonal elements are kernel density estimates, providing insights into the univariate distribution of each variable. The off-diagonal elements represent scatter plots, allowing for visual examination of the bivariate relationships and correlations between variables.

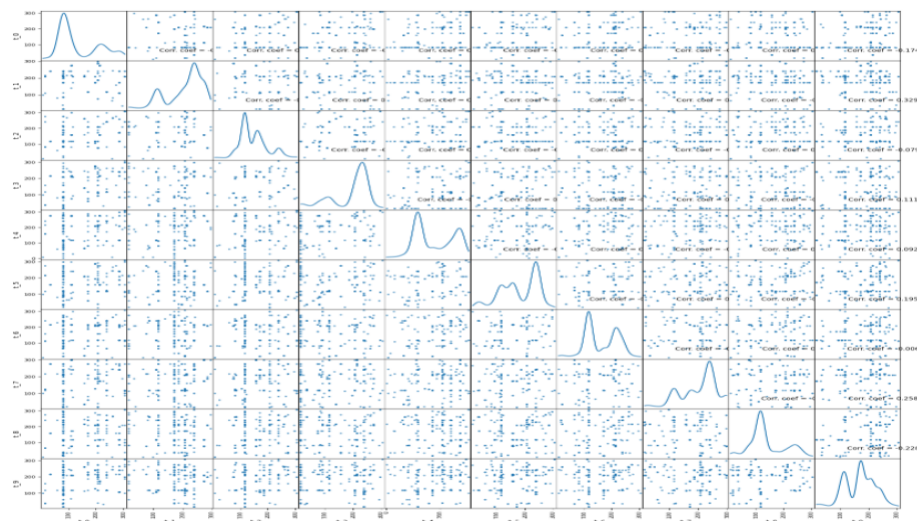


FIGURE 4: SCATTER AND DENSITY PLOT

4.3) DYNAMIC ANALYSIS

Dynamic analysis in the context of malware detection refers to the process of examining the behavior and execution of software or applications in a controlled environment. It involves running the software or malware samples in a sandbox or virtualized environment and monitoring their actions, such as API calls, system interactions, network communications, file access, and memory operations.

In the case of the CUCKOO sandbox for example dynamic analysis was performed on Android applications. The sandbox environment executed the applications while recording the APIs accessed by each application. The resulting dataset contains information about the APIs called by different applications, represented by rows with unique hash values as identifiers.

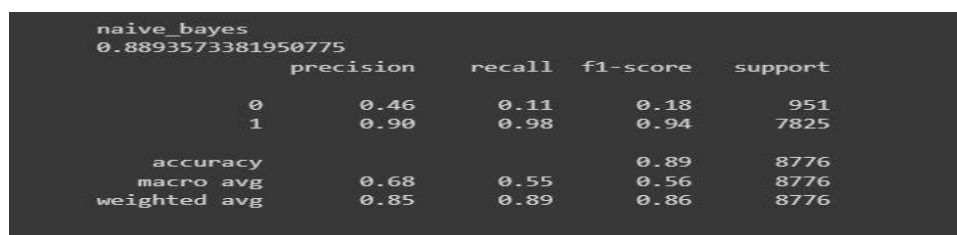
Dynamic analysis provides valuable insights into the runtime behavior of applications or malware. It helps in identifying potentially malicious activities, such as unauthorized network connections, file modifications, or suspicious API calls. By analyzing the dynamic behavior, security researchers and analysts can detect and classify malware, identify patterns or signatures of malicious behavior, and develop effective detection and prevention strategies.

In the above CUCKOO sandbox example, the dynamic API call sequences were processed and fed into the model for training and classification, aiming to accurately distinguish between malware and goodware .

In our code We then perform binary classification using different machine learning models on a dataset containing dynamic API call sequences for malware samples.

Gaussian Naive Bayes Model: -

- The algorithms uses fundamental Gaussian Probability concepts to generate model which uses conditional probability for classification.
- Precision refers to the number of true positives divided by the total number of positive predictions, it indicates the quality of a positive prediction made by the model.
- The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples
- F1 score is defined as the harmonic mean between precision and recall. It is used as a statistical measure to rate performance. It can be used to compare performace of classifiers.
- The support is the number of samples of the true response that lie in that class.



naive_bayes				
0.8893573381950775				
	precision	recall	f1-score	support
0	0.46	0.11	0.18	951
1	0.90	0.98	0.94	7825
accuracy			0.89	8776
macro avg	0.68	0.55	0.56	8776
weighted avg	0.85	0.89	0.86	8776

FIGURE 5: NAIVE BAYES ACCURACY

K Neighbors Model:

- The algorithm treats each column as independent attributes of each data, with which the algorithm tries to find another data with similar attributes.
- Tuning value of k, represents the nature of algorithm ie the number of neighbours to be taken into consideration during classification.

```

kneighbors 3
0.9843892433910666
precision    recall  f1-score   support

   0         0.45    0.89    0.60      113
   1         1.00    0.99    0.99     8663

 accuracy          0.98     8776
  macro avg         0.72    0.94    0.79     8776
  weighted avg      0.99    0.98    0.99     8776

kneighbors 6
0.9847310847766636
precision    recall  f1-score   support

   0         0.44    0.93    0.60      108
   1         1.00    0.99    0.99     8668

 accuracy          0.98     8776
  macro avg         0.72    0.96    0.80     8776
  weighted avg      0.99    0.98    0.99     8776

kneighbors 9
0.9846171376481313
precision    recall  f1-score   support

   0         0.44    0.93    0.59      107
   1         1.00    0.99    0.99     8669

 accuracy          0.98     8776
  macro avg         0.72    0.96    0.79     8776
  weighted avg      0.99    0.98    0.99     8776

```

FIGURE 6: K NEIGHBORS ACCURACY

Decision Tree Algorithm

- The algorithm constructs a tree based splitting data such that the attributes completely classify the data. The tree is then used to classify data based on classifier present in dataset,

```

DecisionTreeClassifier()
0.9832497721057429
precision    recall  f1-score   support

   0         0.66    0.68    0.67      221
   1         0.99    0.99    0.99     8555

 accuracy          0.98     8776
  macro avg         0.83    0.83    0.83     8776
  weighted avg      0.98    0.98    0.98     8776

```

FIGURE 7: DECISION TREE ACCURACY

4.4) BEHAVIORAL ANALYSIS

We are doing a behavioural analysis using CNN-LSTM which refers to a methodology that utilizes Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to analyse and understand the behavioural patterns of a given input sequence. In the context of malware analysis, this approach involves feeding the API call sequence of a malware sample into a CNN-LSTM model. The CNN layers in the model extract spatial features from the input sequence, while the LSTM layers capture temporal dependencies and sequence information. By training the model on a dataset of known malware samples, it learns to recognize and classify the behavioural patterns indicative of malicious activity. This technique allows for the detection and classification of malware based on its unique behavioural characteristics, providing a more effective means of identifying and protecting against malware threats.

The steps involved for behavioural analysis:

1) Data Preprocessing:

- The code begins by importing necessary libraries, including numpy, pandas, seaborn, scikitplot, and matplotlib.
- The dataset is loaded from a CSV file using pandas and stored in the "data" variable.
- Irrelevant columns ("hash" and "malware") are dropped from the dataset, and the resulting data is stored in the "used_data" variable.
- The first few rows of the "used_data" are displayed using "used_data.head()".
- A count plot is generated using seaborn to visualize the distribution of malware classes in the dataset.

2) Data Splitting:

- The dataset is split into training and testing sets using the train_test_split function from scikit-learn.
- The input features (X) and target labels (y) are assigned based on the split.

3) Model Architecture:

- The CNN-LSTM model is defined using the Sequential API of Keras.
- The model consists of layers such as Embedding, BatchNormalization, Conv1D, MaxPool1D, LSTM, and Dense.
- The model summary is printed to provide an overview of the network architecture.
- Model Compilation and Training:
- The model is compiled with the Adam optimizer and binary cross-entropy loss function.
- The model is trained using the training data, with a validation split of 0.2, for 30 epochs and a batch size of 512.
- The training history is stored in the "history" variable.

4) Model Evaluation and Visualization:

- The trained model is saved to a file named "behavioral-malware-detection-based-on-api-calls_model.h5".
- Two subplots are created using matplotlib to display the training and validation loss, as well as training and validation accuracy over epochs.

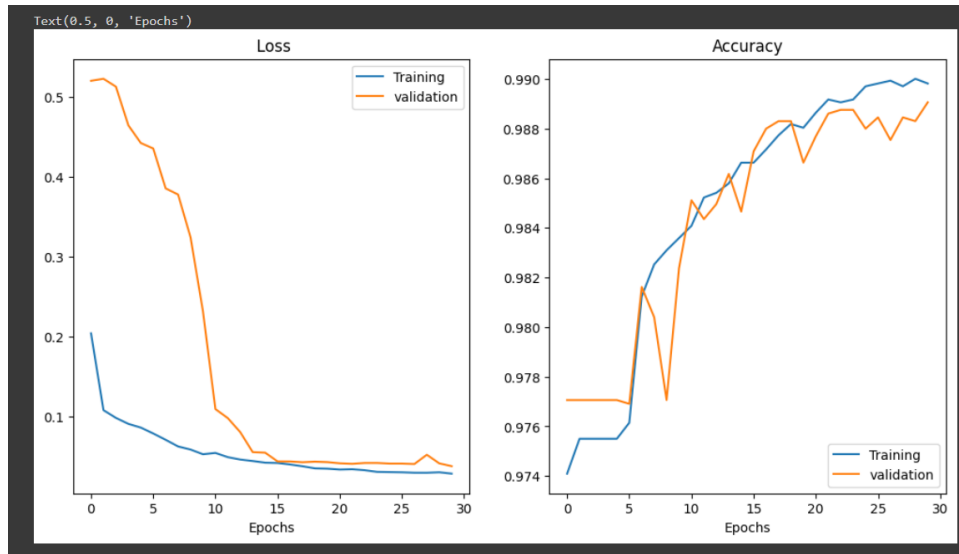


FIGURE 8: ACCURACY AND LOSS GRAPHS

5) Model Testing and Evaluation:

- The model is used to predict the labels of the testing data (X_{test}), and the predictions are stored in "y_pred".
- The probabilities of the positive class (malware) are calculated and plotted using scikit-plot for ROC (Receiver Operating Characteristic) and precision-recall curves.
- This code provides a comprehensive pipeline for training a CNN-LSTM model to detect malware based on API call sequences. The visualization of training metrics and evaluation plots allows for an assessment of the model's performance.

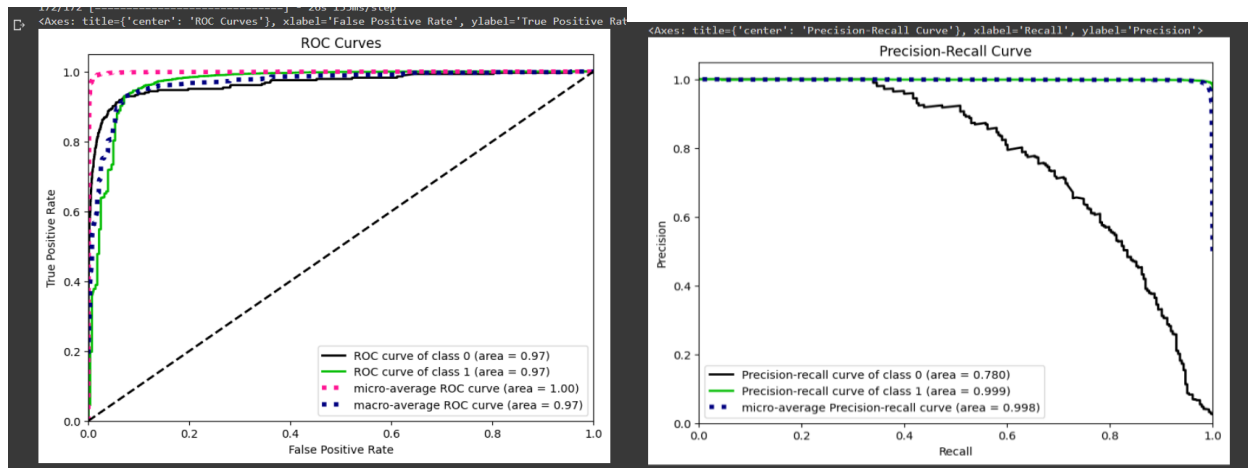


FIGURE 9: ROC AND PRECISION RECALL CURVE

4.5) MULTI-LAYER PERCEPTRON

1. Data Preparation:

- The dataset is loaded using pandas from a CSV file.
- The target variable "malware" contains multiple classes, as shown in the countplot.

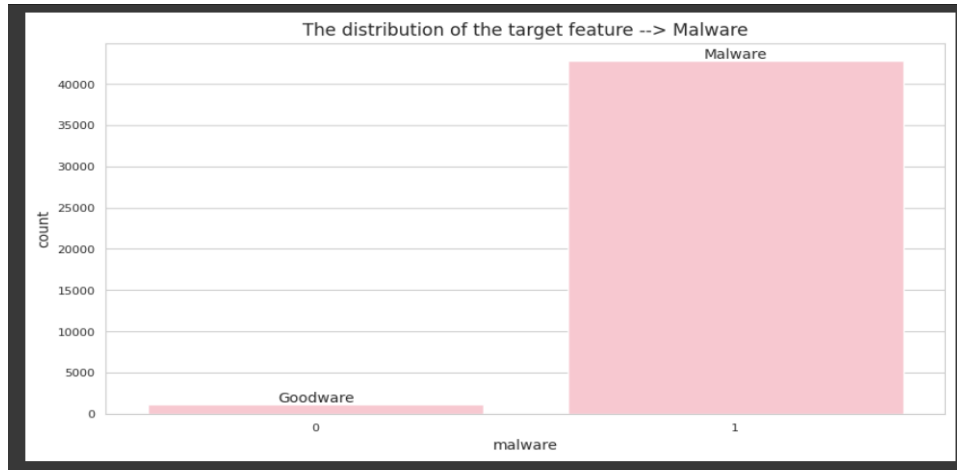


FIGURE 10: DISTRIBUTION OF MALWARE AND GOODWARE

2. Preprocessing:

- StandardScaler from scikit-learn is used to scale the input features.
- The dataset is split into training and testing sets using train_test_split.

3. Model Definition:

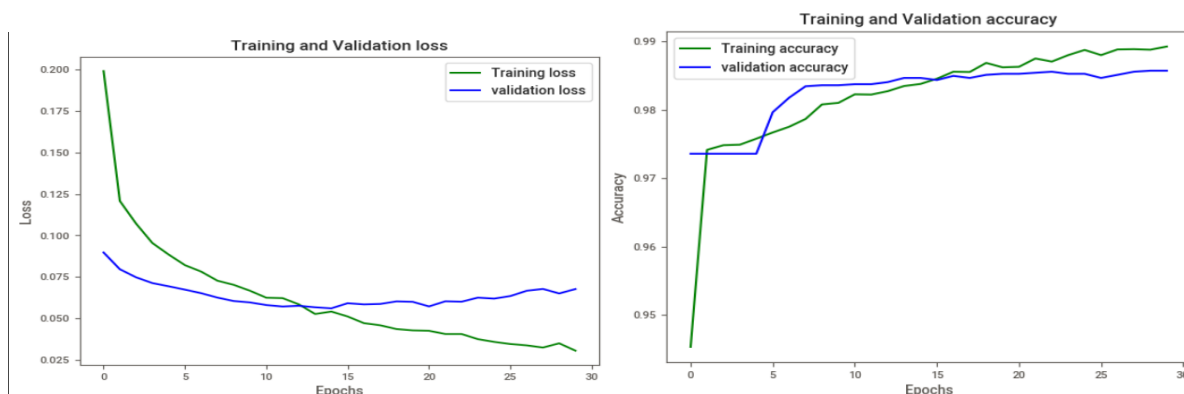
- A Sequential model is created using Keras.
- Dense layers with different activation functions (ReLU) are added to the model.
- Dropout layers are inserted to prevent overfitting.
- The final layer has units equal to the number of classes (1 in this case) and uses the sigmoid activation function.

4. Model Compilation and Training:

- The model is compiled with binary_crossentropy as the loss function and 'adam' as the optimizer.
- The model is trained using the training data with a validation split of 0.20, for 30 epochs and a batch size of 256.
- EarlyStopping is implemented to monitor the validation loss and stop training if it doesn't improve for a certain number of epochs.

5. Model Evaluation and Visualization:

- The model is evaluated on the testing data using the evaluate method.
- Training and validation loss are plotted to observe the model's learning progress.
- Training and validation accuracy are plotted to assess the model's performance over epochs.



11: ACCURACY AND LOSS GRAPH FOR MLP

FIGURE

RESULTS

Accuracies Achieved:

- Gaussian Naive Bayes Model: - 88.93 %
- K-nearest neighbours: - 99.4 %
- Decision Tree Algorithm: - 98.3 %
- CNN-LSTM: -98.91 %
- Multi-layer Perceptron: -98.57 %

CONCLUSION

Section 1 focuses on traditional machine learning algorithms (Naive Bayes, KNN, Decision Tree) and provides basic classification results.

Section 2 introduces a more complex model architecture combining CNN and LSTM for classification, along with additional evaluation metrics and visualizations.

Section 3 involves a multilayer perceptron neural network and offers more advanced training and evaluation techniques, including data visualization and analysis.

The choice of which code to use depends on the specific requirements and goals of the project. If simplicity and basic classification results are sufficient, first approach may be suitable. Second approach involves a more advanced model architecture, suitable for complex datasets and deep learning tasks and the third approach provides a more comprehensive analysis with a neural network approach.

This [GitHub](#) repository has all the notebooks used in this project.

REFERENCES

[Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks](#)

[API Call Sequence Malware Dataset](#)

[Kaspersky-Lab-Whitepaper-Machine-Learning](#)