End-to-End Implementation of Malware Detection Using Machine Learning

Vasu Bansal[1], Yerramalli Sai Sreekar[2]

School of Computer Science and Engineering,
Vellore Institute of Technology,
Kelambakkam, Chennai

***Abstract*— With the increasing sophistication and frequency of malware attacks, there is a pressing need for effective malware detection techniques. Machine learning has emerged as a promising approach for identifying and classifying malicious software, leveraging the power of algorithms and statistical models to analyse complex data and detect patterns that may be indicative of malware behaviour. In this project, we propose a web application that utilizes machine learning techniques to classify uploaded files as malware or not. Our system is based on a comprehensive dataset of known malware and non-malware files, which we use to train and test our machine learning algorithms. Our experimental results demonstrate that our system is highly effective at detecting malware, achieving high accuracy rates even with previously unseen samples. Overall, our project highlights the critical importance of machine learning for addressing the growing threat of malware attacks and provides a practical solution for identifying potentially harmful files.**

Keywords*—*Machine Learning, Malware Detection, Cyber Security, Malware, Feature Extraction, Random Forest, Principal Component Analysis, Adaboost, Gaussian Naïve Bayes, PE File

I. INTRODUCTION

The proliferation of malware attacks in recent years has posed a significant threat to individuals and organizations worldwide. These attacks not only compromise sensitive information but can also result in significant financial losses. Therefore, developing effective malware detection techniques has become a critical priority in the field of cybersecurity.

In this project, we propose a web application that employs machine learning algorithms to classify uploaded files as malware or not. These algorithms work by analysing various features of the uploaded files, such as their size, file type, and content, to determine whether they are malicious. We trained and tested our classifiers using a publicly available dataset of malware samples and non-malware files. Our experimental results show that our proposed system achieves high accuracy in malware detection, making it a powerful tool for identifying potentially harmful files.

Overall, our project offers an effective as well as fast solution for detecting malware that can be readily used to check for potential threats. With its high efficiency due to only stateless classification, our system performs malware checks faster than most available software. It can help mitigate the growing threat of malware attacks, enabling individuals and organizations to better protect themselves against cybersecurity threats.

II. LITERATURE REVIEW

A large amount of research has gone towards the use of Machine Learning as a method to determine whether a file has malware or not. Ivan Firdausi in [1] discusses the growing threat of malware and the limitations of manual heuristic inspection for malware analysis. The authors suggest that machine learning techniques can be effective in detecting malware. [2] discusses the exponential growth of malware over the last decade and its significant financial impact on organizations. The authors propose a method for malware detection using machine learning algorithms such as decision tree and random forest.

D. Gavrilut and D. Anton [3] proposed a versatile framework for distinguishing between malware and clean files using different machine learning algorithms. The authors aimed to minimize the number of false positives while successfully detecting malware. [4] discusses the effectiveness of machine learning algorithms in detecting known and unknown malware across various file formats, domains, and platforms. The authors emphasize the importance of feature extraction methodology in improving the performance of machine learning algorithms.

There has always been a debate between whether stateless or stateful methods are more effective. [5] and [6] present a detailed review of malware detection mechanisms used by researchers. The authors classify scientific works on malware detection according to applied methods of detection, accuracy of detection, revealing stateless methods of detection to be quite accurate as well.

III. TECHNIQUES AND ALGORITHMS USED

A. PEFile Module:

PEFile, which stands for Portable Executable File, is a file format used in Microsoft Windows operating systems to represent executables, DLLs (Dynamic Link Libraries), and other binary files. The PEFile structure follows a well-defined format that includes a header and various sections. The header

contains important information such as the file's signature, architecture, entry point, and size of the different sections. The sections in a PEFile store different types of data, including code, data, resources, and imports/exports. The PEFile structure is designed to be portable and provides information for the operating system to load and execute the binary file correctly. It also includes optional headers that contain additional information for advanced features such as exception handling and security. The PEFile structure is a fundamental component of Windows executables, and understanding its format is essential for analyzing, debugging, and manipulating binary files in the Windows environment.

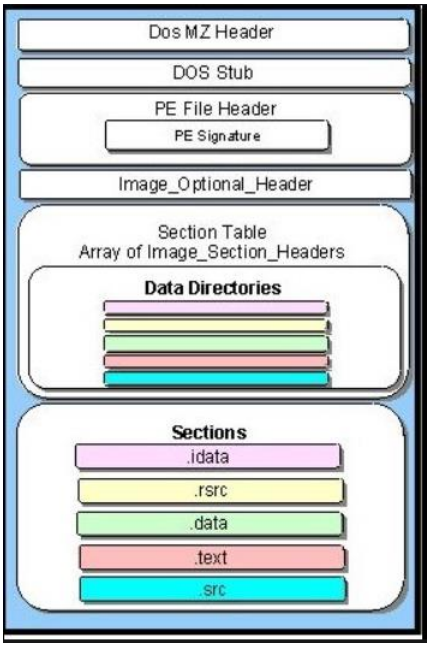

Fig. 1 PEFile Structure

B. *Gaussian Naïve Bayes:*

Gaussian Naïve Bayes is a probabilistic classifier that assumes that the features in the data follow a Gaussian (normal) distribution. It is a simple and efficient algorithm that works well with small to moderate-sized datasets. It calculates the conditional probability of each feature given the class label, and then uses Bayes' theorem to estimate the probability of an input data point belonging to a particular class. Despite its "naïve" assumption of feature independence, Gaussian Naïve Bayes can often achieve good accuracy in many classification tasks, especially when the features are continuous and normally distributed.

C. *Random Forest:*

Random Forest is an ensemble learning method that combines multiple decision tree classifiers to create a more robust and accurate model. It randomly selects a subset of features and data samples to build each decision tree, and then combines the predictions of all the trees to make the final classification decision. Random Forest is known for its ability to handle high-dimensional data, noisy data, and missing values. It can also handle categorical features and provides estimates of feature importance, which can help with feature selection. Random Forest is a popular algorithm for classification tasks due to its ability to reduce overfitting, improve generalization performance, and handle a large number of features.

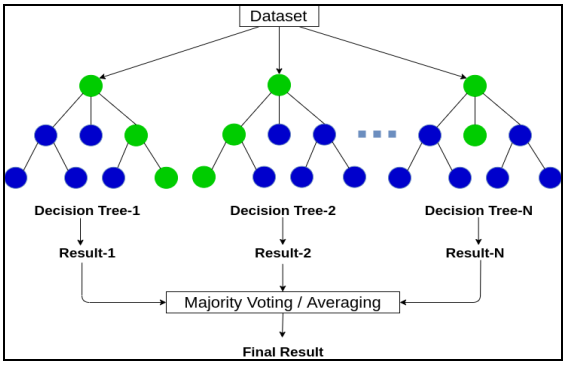

Fig. 2 Working of Random Forest

D. *Principal Component Analysis:*

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform a dataset with multiple correlated features into a lower-dimensional representation while preserving the most important information. PCA identifies the principal components, which are orthogonal directions in the feature space that explain the most variance in the data. The first principal component captures the largest amount of variance, and each subsequent component captures the maximum remaining variance orthogonal to the previous components. PCA is widely used for feature extraction, visualization, and data compression. It can help reduce the complexity of a dataset, improve model performance by removing redundant information, and provide insights into the underlying structure of the data.

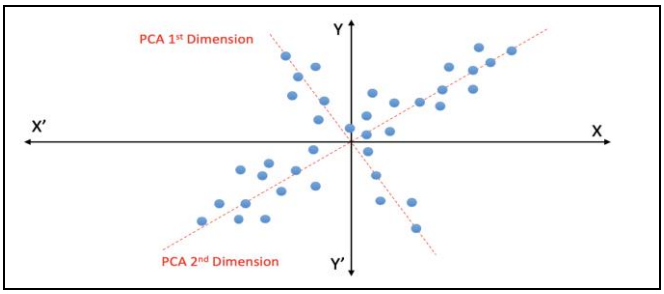

Fig. 3 Working of Principal Component Analysis

E. *Ensemble Voting Classifier:*

Ensemble Voting Classifier is an ensemble learning method that combines the predictions of multiple base classifiers by taking a majority vote. It can use different types of classifiers or the same type of classifiers with different hyperparameter settings. The idea is that by combining the predictions of

multiple classifiers, the ensemble can overcome the limitations of individual classifiers and achieve better overall performance. Ensemble Voting Classifier can be used for both classification and regression tasks, and it is known for its ability to improve accuracy, reduce bias and variance, and increase the robustness of the final predictions. It is particularly effective when the base classifiers have complementary strengths and weaknesses, and it can handle a wide range of data types and distributions.

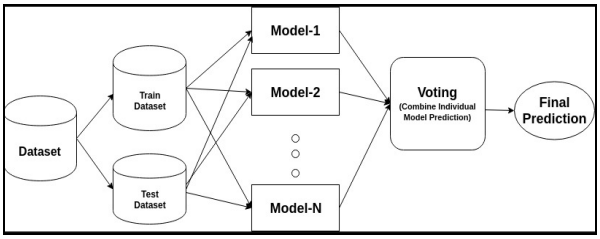

Fig. 4 Working of Ensemble Voting Classifier

F. Adaboost Classifier:

Adaboost (Adaptive Boosting) is an ensemble learning method that combines the predictions of multiple base classifiers by assigning different weights to the training samples. It initially assigns equal weights to all samples and then iteratively increases the weights of misclassified samples to focus on the samples that are difficult to classify. Adaboost adjusts the weights of the training samples at each iteration to emphasize the misclassified samples and improve the performance of the base classifiers. The final prediction is based on a weighted vote of the base classifiers, with higher weights given to more accurate classifiers. Adaboost is known for its ability to improve the accuracy of weak classifiers and produce strong classifiers that can handle complex and noisy data. It is widely used for binary classification tasks and can be combined with different base classifiers to achieve improved performance.

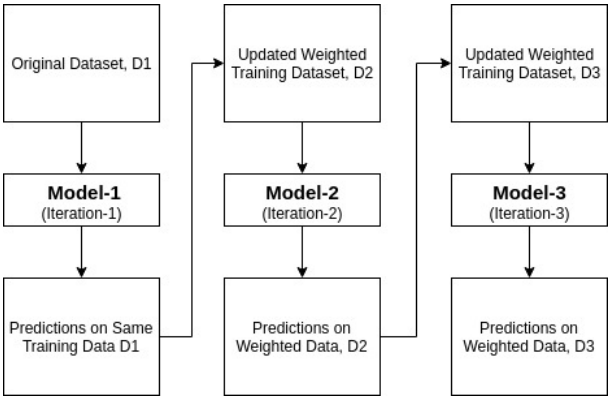

Fig. 5 Working of Adaboost Classifier

IV. PROPSED METHODOLOGY

We propose a web application where you can upload a suspected malware to check if it is a malware or not. The backend implements a Python program which functions as shown in Fig. 6. The dataset we have used for model training is taken from [7]. It consists of 78 columns representing different features extracted from benign or harmful viruses and 19611 rows, each of which contains data of a particular virus abstracted from VirusShare. One of these columns contains information on whether the given sample is a virus or not in binary format.

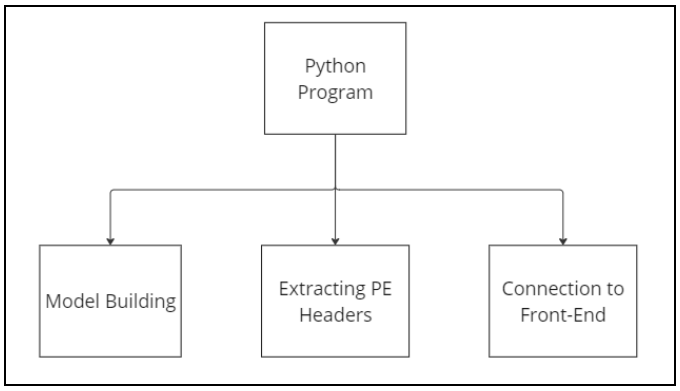

Fig. 6 Functioning of the Python Program

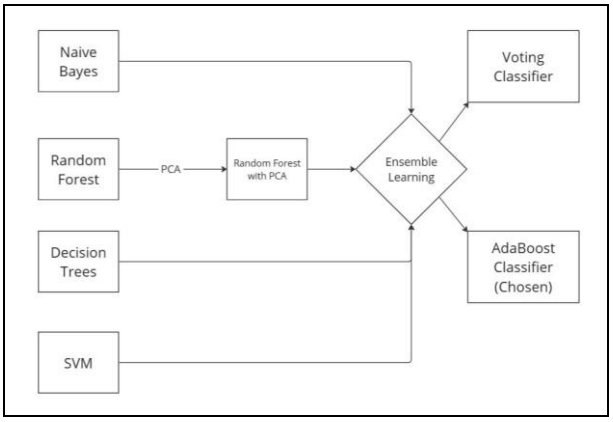

Fig. 7 Model Building

Once a suspected file is uploaded, it is sent to the backend where required features are extracted using PEFile module of Python. The structures defined in the Windows header files are accessible as attributes in the PE instance. Each step as shown in Fig. 7 is explained in detail here:

A. Dataset Pre-processing

The dataset consists of a total of 78 columns out of which only 69 contain accessible or useful information. Unnecessary columns like name of the file, e_magic, e_cblp and some other error values are dropped in order to maintain high accuracy of the expected model.

B. Feature Importance

Feature importance in machine learning is a critical aspect that helps determine the relative significance of different

features in predicting the target variable. It allows practitioners to identify the most influential features in the model's decision-making process. By understanding feature importance, practitioners can make informed decisions regarding feature selection, model interpretation, and optimization. This information helps in building more accurate and interpretable machine learning models, leading to better insights and predictions. The following features were found to be the most important in the analysis we performed (also shown in Fig. 6)

1. *e_ss*: This refers to the "Stack segment" field in the PE file header, which specifies the size of the stack segment in the executable file. It can be relevant in analyzing malware to understand the memory layout and usage of the program, as abnormal or suspicious stack sizes may indicate potential buffer overflow or stack-based attacks.
2. *Characteristics:* This field in the PE file header contains flags that describe various characteristics of the executable, such as whether it is a DLL or an executable, if it is a console or GUI application, whether it is a 32-bit or 64-bit binary, etc. These characteristics can provide insights into the intended behavior of the program and help identify potential anomalies or deviations from expected behaviors, which could be indicative of malware.
3. *SizeOfInitializedData*: This field in the PE file header specifies the size of the initialized data section, which contains data that is initialized by the operating system when the executable is loaded into memory. Malware may manipulate this section to store malicious code or data, or to hide its presence by modifying the expected size of this section.

4. *e_lfanew*: This field in the PE file header is an offset to the PE header, which contains information about the structure and layout of the PE file. The value of this field is used by the operating system to locate the PE header. Malware may tamper with this field to evade detection or alter the expected execution flow of the program.

5. *SizeOfHeapCommit*: This field in the PE file header specifies the size of the committed heap memory that is allocated when the executable is loaded. An unusually large value in this field may indicate potential heap-based attacks or abnormal memory allocation behavior, which could be indicative of malware.

6. *SizeOfUninitializedData*: This field in the PE file header specifies the size of the uninitialized data section, which contains data that is not initialized by the operating system when the executable is loaded into memory. Malware may manipulate this section to store encrypted or obfuscated code or data.

7. *Magic*: This field in the PE file header is a signature that identifies the file as a valid PE file. Different values of this field indicate different types of PE files, such as executable, DLL, etc. Analyzing the magic value can help determine if the file is a legitimate PE file or potentially a disguised malware.

8. *PointerToSymbolTable*: This field in the PE file header points to the symbol table, which contains information about the symbols (e.g., functions, variables) in the executable. This field can be relevant in analyzing malware to identify potential obfuscation techniques, such as missing or manipulated symbol table entries, which may indicate attempts to hide the true purpose or behavior of the program.

9. *AddressOfEntryPoint*: This field in the PE file header specifies the entry point of the executable, which is the first instruction that is executed when the program starts. Malware may modify this field to alter the expected execution flow of the program or to inject malicious code at the entry point, which can be indicative of malware.

10. *TimeDateStamp*: This field in the PE file header contains a timestamp indicating when the executable was compiled. Analyzing this field can provide information about the age of the executable and help identify potential discrepancies, such as executables with future timestamps or unusually old timestamps, which may indicate suspicious activity or attempts to evade detection.

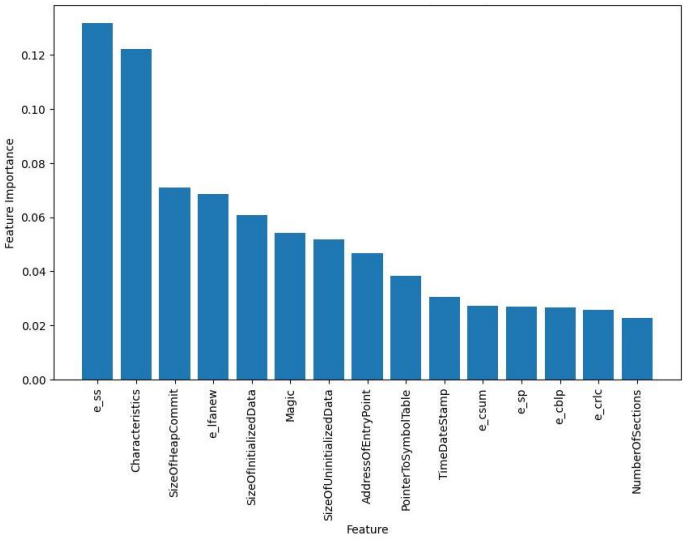

Fig. 8 Top Features Sorted on Basis of Importance.

C. Model Training

80% of the data is used for training. In order to find the best accuracy, various models like Gaussian Naïve Bayes, Random Forest, Adaboost Classifier and an Ensemble model were used.

D. Model Testing and Selection

The remaining 20% of the data was used for testing. The accuracy. precision, recall and F1-score of each of the model was calculated and the model with the best score was chosen.

E. Feature Extraction from File

The required PEFile header features need to be extracted and passed to the trained model for prediction. The implemented code is able to extract 69 out of the 78 features available in the dataset.

F. Prediction

Once the features are extracted, the chosen model is used to predict whether the uploaded file is a malware or not. Based on the output, the user is redirected to different pages to further show the required output and provide some guidance to user considering a malware does exist.

The flow of the code will go something like this - The python code is kept running live using 'uvicorn', which is an ASGI (Asynchronous Server Gateway Interface) server implementation for Python. The user accesses the home page and uploads the file they feel requires some checking. The file is passed onto the backend and Python functions in the backend predict whether the given file contains a virus or not. If the uploaded file has a risk of having a virus, the user is suggested some remedies on how to check if their PC has been damaged. If not, they get a green light on opening and using the file.

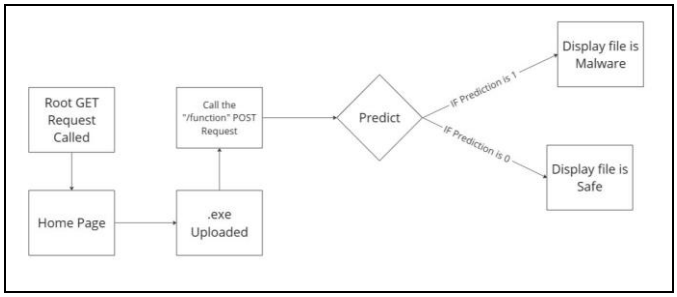

Fig. 9 Working of the Proposed Application Backend.

V. IMPLEMENTATION & RESULTS

The various models tested along with the scores obtained in each of them are shown in Table 1. As we can see, Random Forest with Principal Component Analysis shows the highest accuracy of 99.41%.

TABLE I
CLASSIFICATION REPORT OF GAUSSIAN NAÏVE BAYES

	Precision	**Recall**	**F1 score**	**Support**
0	1.00	0.27	0.42	3635
1	0.10	0.99	0.18	288
Accuracy			0.32	3923
Macro Average	0.55	0.63	0.30	3923
Weighted Average	0.93	0.32	0.41	3923

TABLE II
CLASSIFICATION REPORT OF RANDOM FOREST MODEL

	Precision	**Recall**	**F1 score**	**Support**
0	0.96	0.99	0.97	961
1	1.00	0.99	0.99	2962
Accuracy			0.99	3923
Macro Average	0.98	0.99	0.98	3923
Weighted Average	0.99	0.99	0.99	3923

TABLE III
CLASSIFICATION REPORT OF RANDOM FOREST MODEL USING PCA

	Precision	**Recall**	**F1 score**	**Support**
0	0.99	0.99	0.99	970
1	1.00	0.99	1.00	2953
Accuracy			0.99	3923
Macro Average	0.99	0.99	0.99	3923
Weighted Average	0.99	0.99	0.99	3923

TABLE IV
CLASSIFICATION REPORT OF ENSEMBLE MODEL

	Precision	**Recall**	**F1 score**	**Support**
0	0.94	0.96	0.95	962
1	0.99	0.98	0.98	2961

Accuracy			0.98	3923
Macro Average	0.97	0.97	0.97	3923
Weighted Average	0.98	0.98	0.98	3923

TABLE V
CLASSIFICATION REPORT OF ADABOOSTER ALGORITHM

	Precision	**Recall**	**F1 score**	**Support**
0	0.97	0.98	0.97	970
1	0.99	0.99	0.99	2953
Accuracy			0.99	3923
Macro Average	0.98	0.98	0.98	3923
Weighted Average	0.99	0.99	0.99	3923

TABLE VI
SUMMARY OF REPORTS OF ALL MODELS USED

Model Used	**Accuracy (%)**	**Precision**	**Recall**	**F1 score**
Gaussian Naïve Bayes	32.24	0.93	0.32	0.41
Random Forest	98.72	0.99	0.99	0.99
Adaboost	99.29	0.99	0.99	0.99
Ensemble	98.39	0.99	0.99	0.99
Random Forest with PCA	99.41	0.99	0.99	0.99

Random forest with Principal Component Analysis is used in the Web Application that we have made for the same project. The web application uses Python in the backend, FastAPI as the module for connection to the frontend and HTML, CSS and JS to make a user-friendly UI in order for any person to use as can be seen in Fig. 10.

Fig. 10 Home Page of Web Application

VI. CONCLUSION

In conclusion, our project on end-to-end implementation of malware detection using multiple machine learning models has provided us with a deep understanding of the complexity involved in the detection of malware in the modern digital landscape. After experimenting with models mentioned in Table 1, we selected Random Forest with Principal Component Analysis as our final model due to its ability to improve the accuracy of the detection process and reduce the risk of false positives. We believe that if stateless inspection of the PE header file gives such a high accuracy (99.41%), it is worth using rather than stateful inspection which takes a lot of time.

By automating the process of malware detection, we can help individuals and organizations to detect malware infections early, prevent data breaches, and reduce downtime. These benefits can significantly enhance the security of sensitive information and maintain business continuity. Our project can serve as a foundation for future research in the field of malware detection, and we hope that it can inspire others to build upon our work and contribute to the development of more robust and accurate machine learning models. Ultimately, our project aims to provide individuals and organizations with a powerful tool to stay safe and secure in an ever-evolving digital landscape.

REFERENCES

[1] "*Malware Detection using Machine Learning*," ResearchGate. [Online]. Available: https://www.researchgate.net/publication/224089748_Malware_detection_using_machine_learning
[2] "*IEEE Xplore Abstract Record*," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5675808
[3] "IEEE Xplore Document Record," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/9673465/
[4] "*Expert Systems with Applications*," ACM Digital Library. [Online]. Available: https://dl.acm.org/doi/10.1016/j.eswa.2023.119615
[5] "*IEEE Xplore Document Record*," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/9620415
[6] "IEEE Xplore Document Record," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/9445322/
[7] "*An Emerging Malware Analysis Techniques and Tools: A Comparative Analysis*," International Journal of Engineering Research

& Technology (IJERT). [Online]. Available: https://www.ijert.org/research/an-emerging-malware-analysis-techniques-and-tools-a-comparative-analysis-IJERTV10IS040071.pdf

[8] "*Chapter 21 - Malware Detection and Prevention Techniques*" SpringerLink. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-6648-6_21