10.48047/jocaaa.2024.33.08.159

MALWARE DETECTION USING MACHINE LEARNING THROUGH CLASSIFICATION TECHNIQUES

Radhika Singh[1], Priyank Sirohi[2] Gur Sharan Kant[3]

[1] Research Scholar, M.Tech (CSE), SCRIET, Chaudhary Charan Singh University, Meerut, U.P, India.

[2] Research Supervisor, M.Tech (CSE) Assistant Professor, Information technology Department, SCRIET, Chaudhary Charan Singh University, Meerut, U.P, India.

[3] Research Supervisor, M.Tech (CSE) Assistant Professor, Computer Science Department, SCRIET, Chaudhary Charan Singh University, Meerut, U.P, India.

Mail id- [1] radhikasingh172001@gmail.com, [2] priyanksirohi01@gmail.com,

[3] gskant9319@gmail.com

ABSTRACT

In current digital age, when crucial tasks are carried out online, cybersecurity is a top worry for both individuals and organisations. The majority of cyber security problems are caused by malware, which also generates a lot of it as a tool. For taking advantage of flaws in installed software and systems. It is necessary to identify and distinguish these malicious programs from innocuous software. This thesis offers low-processing-power malware detection methods that are reliable and effective. For malware identification, the XGboost machine learning (ML) method is used to a sizable malware dataset spanning a considerable amount of time. This machine learning model has been enhanced. Building the malware detection model with little processing resources and increasing malware detection efficiency to 98.5% accuracy and 0.999 AUC by feature selection based on feature relevance in the XGBoost algorithm. To mimic unknown future zero-day malware, three datasets with malicious and benign samples created at different times are employed. Ensemble bagging and boosting machine learning techniques are used to train machine learning models, which are then evaluated on unidentified future data. The traits that contribute most to malware detection are then identified using the Shapley value, which is based on game theory. Lively bar graphs and waterfall plots are used to illustrate these key characteristics. The machine's Shapley values for its characteristics. In order to directly connect with the results of bagging and boosting machine learning models, learning models are transformed into probability scales. The total of the features' contributions on a probability scale can be used to describe the result of a machine learning model that has been trained using a bagging or boosting method for malware. Inductive rules are developed by analysing these measurable, explicable contributions of ML model characteristics. These guidelines can be applied to change the incorrectly identified zero-day malware class. All incorrectly identified

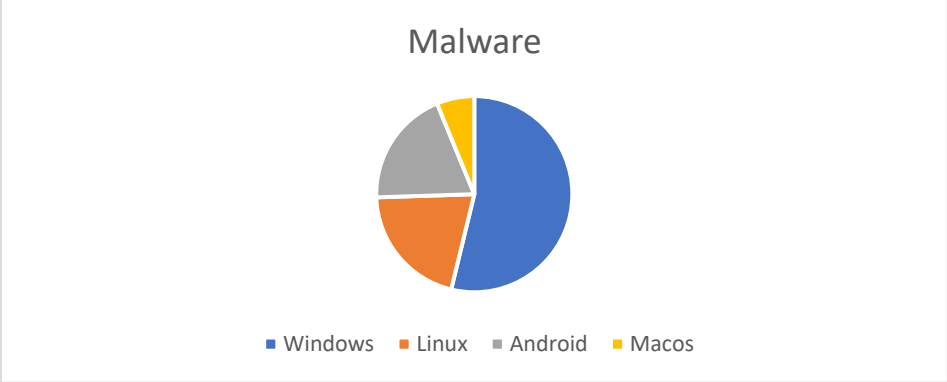
10.48047/jocaaa.2024.33.08.159

samples may be removed using these inductive principles in a controlled surroundings, attaining a perfect score. The model may be made resilient by using these inductive principles to re-verify correctly categorised samples in a controlled setting.

Keywords: *Cybersecurity, Malware, Machine Learning, Malware detection, Explainable ML, Shapley value.*

INTRODUCTION

Software is created by computer programs. The software links people and machines, increases efficiency, and assists users in completing tasks. The Operating System (OS), device drivers, graphics, database management system, browsers, movie players, PowerPoint, spreadsheets, word processors, and more are examples of software that is widely used. Antiviral software, etc. Some software is malevolent and does things that users aren't supposed to do. Malware is software designed to do harm. Malware circumvents access rules, impacts computer processes, and accesses and steals a variety of user and organisational data. Malware may alter and manipulate data in systems that make a wide range of choices, including authorising people and granting permissions for particular tasks. Any operating system, including Microsoft Windows, Android, Mac, Unix, Linux, Ubuntu, Fedora, and others, can contain malware as an application. Nonetheless, throughout the history of compu ter systems for laptops, desktops, and servers, Microsoft Windows OS has been the most often attacked operating system. In the years to come, this pattern will persist. The widespread usage of Microsoft OS in computer systems is one factor contributing to this trend. The virus in Microsoft OS is shown in Figure 1.1 at (AV-TEST-The Independent-ITSecurity-Institute 2023) in comparison to Unix, Android, Mac, and other operating systems. Hackers and malware developers work harder into breaching the system with a higher chance of success, regardless of the operating system's security features.



10.48047/jocaaa.2024.33.08.159

Fig. 1.1 Comparison of malware in Windows, Linux, Android, and Mac operating system

1.1 Malware varieties and propagation

All forms of harmful software are collectively referred to as malware. Malware is categorised according to the way the software behaves. This section explains the characteristics of malware, the particular category they fall under, and the several ways. The malicious software spreads and then infects the computers.

Adware: When a user is surfing or online, the application shows ads for deals and goods of various kinds, including games, movies, watches, etc. It makes incentives that entice users to visit a certain website before gathering user data and/or the user's interests.

Virus: A virus is a piece of software that alters other programs by attaching itself to their code and running when the program is run. The virus assumes command of input and runs the program as it pleases. It reduces PC performance by taking control over the software it oversees.

Worm: The program replicates itself by acting similarly to a virus. Worms propagate and infect other linked machines over a computer network. External hard discs, USBs, and other storage devices are not necessary for it to infect other the network's computers are like viruses.

Ransomware: The program locks the user files and data by encryption. The user is not able to access the files and data stored in the computer. Dependence of users on important and/or confidential data and files move them to a state of helplessness.

1.2 Malware and information security challenges

Secure information sharing, storage, and access are critical given the rise in digital activities for organisations and individuals across a wide range of industries, including banking, education, design, manufacturing, transportation, energy, and more. Malware has to be screened and separated from safe software for information security. Malware is the process of detection involves locating the required pattern in code originating from an API that software calls for certain operations. Code patterns produce signatures, which are collections of hex values. This requires extensive time and resources and analysis by subject-matter specialists. The current rate of malware production cannot be met by this approach of extracting signatures from malware for detection. The most advanced techniques for identifying patterns that can be used to the quicker detection of malware at the comparable rate at which it is now created are artificial intelligence and machine learning. Features obtained from executable files, malware samples, and safe software are subjected to machine learning. The characteristics are obtained by either static or dynamic examination of an executable file, or both analyses together.

10.48047/jocaaa.2024.33.08.159

Different techniques used for malware static analysis produce different properties. Combining many characteristics from different approaches increases the effectiveness of malware detection. Over time, the malware adapts its methods to take advantage of weaknesses and avoid being discovered by antivirus software. Therefore, one requires malware for a long time in order to develop an efficient malware detection model. High processing power is needed to create a machine because of the combination of significant malicious characteristics and massive malware samples from a long time period model of learning. The developed machine learning model continues to classify incorrect data as false positives and false negatives. Machine learning algorithms are unable to eliminate all forms of misclassification, although they can lessen one at the expense of another. The Machine Learning (ML) model may overfit as a result of complexity and further training. To address these problems, approaches other than machine learning techniques must be used. An organisation with any degree of sophisticated security measures only needs one false negative misclassified sample under test (SUT) to raise concerns about cybersecurity. Users of the program may experience anxiety and discomfort as a result of a false positive misclassified sample. This is driven by these difficulties investigation.

CONTRIBUTION

1. After lowering the amount of characteristics to a tenth by information gain on a machine of low to medium processing capacity, a viable malware detection model based on the eXtreme Gradient Boost (XGBoost) machine learning technique was developed from a huge number of viruses.
2. Creation of the LightGBM explainable machine learning model, which employed Shapley value to provide context for the model's decision-making.
3. Creation of machine learning models for explainable boosting and bagging in order to detect zero-day malware.
4. Developed a unified framework for real-time malware prediction that leverages three distinct malware machine-learning algorithms and a Windows executable.

LITERATURE REVIEW

In the literature, a number of malware detection techniques have been put out. A methodical approach to malware detection based on malware analysis types, malware data representations, and their application to the detection of malware in widely used operating systems, such as

10.48047/jocaaa.2024.33.08.159

Windows and Android, is shown in Fig. 2.1 and explained through the literature review for this study specific references.

2.1 Static Analysis

Without actually running the binary file, static analysis of malware extracts characteristics from an executable program. It is a popular and successful malware detection technique.

- N-gram of bytecode (Yin et al. (2020), Tien et al. (2020), Masud et al. (2008),
- N-gram of Application Programming Interface (API) calls (Alazab et al. (2010), Zhang et al. (2019), Chen et al. (2022)),
- Entropy of file or a part of the file (Baysa et al. (2013), Baek et al. (2021), Yin et al. (2020), Xiao, Li, Chen and Li (2020), Lyda and Hamrock (2007)),
- Strings (Afek et al. (2019), Dib et al. (2021) Torabi et al. (2021)),
- Permission of various types (Namrud et al. (2022), Xiao, Chen, He, Feng and Xue (2020))
- Opcode encoding Zhang et al. (2019),
- API calls Mahindru and Sangal (2020b), • Bag of visual words (BoVW) (Bakour and Unver (2021 ¨ a), Bakour and Unver ¨ (2021b)),
- Images of bytes of program executable that may be gray (Hemalatha et al. (2021), Vasan et al. (2020)), color (Huang and Kao (2019), Naeem et al. (2020)), etc.
- Opcode encoding Zhang et al. (2019),
- API calls Mahindru and Sangal (2020b),
- Bag of visual words (BoVW) (Bakour and Unver (2021 ¨ a), Bakour and Unver ¨ (2021b)),
- Images of bytes of program executable that may be gray (Hemalatha et al. (2021), Vasan et al. (2020)), color (Huang and Kao (2019), Naeem et al. (2020)), etc.

2.2 Dynamic Analysis

In dynamic analysis, properties of the program are derived by running the software and observing and analysing its activity. If the sample program is malicious, its execution will infect the machine on which it runs. Therefore, the sample is run in an emulated environment like a sandbox (Anubis, CWsandbox, etc.) or a virtual environment like a Virtual Machine (VM) like Virtualbox, VMware, Cuckoo. The host computer is not infected by the virtual environment. The virus has the ability to generate, remove, and alter files. Malware can store data in certain secret places thanks to these behaviours. Therefore, the removal of dangerous data from one location enables the virus to continue operating on a computer by using files

10.48047/jocaaa.2024.33.08.159

from other locations. In order to alter operating system configuration, the virus creates or modifies registry entries. Malware can therefore launch itself whenever a machine boots up or when certain selected event takes place. Malware either starts new processes or joins ones that already exist. Malware can initiate contact with the authors' C&C centres or URLs in order to get commands or download further malware.

2.3 Hybrid Analysis

Malware that performs dynamic analysis can identify the virtual environment. Once a virtual environment has been identified, the virus ceases its nefarious activity. Malware may be regarded as innocuous software if it hides its harmful actions. Therefore, in order to prevent malware detection, the skilled malware analyst must examine the concealing conduct of the infection. Dynamic analysis of malware overcomes the constraints of static analysis, but it also introduces new problems that static analysis does not. It indicates that there is no chance to hide activity upon environment detection and that the executable file is examined without executing the file in static analysis. In this case, the greatest features that can more effectively identify malware are provided by hybrid analysis, which combines static and dynamic features. Three criteria may be used to classify the optimal feature selection for both static and dynamic analysis. i) Static analysis complements dynamic analysis by utilising its features and characteristics that were unable to be retrieved because to malware obfuscation. ii) The results of static analysis can be used by dynamic analysis to get over its drawbacks, such the malware's ability to identify the virtual environment and subsequently hide its destructive activity. iii) A hybrid model feeds the deep learning model a mix of static and dynamic analytical information. A deep learning model's more effective features are taken out and then supplied to machine learning models for enhanced detection of malware.

RESEARCH GAP

Finding research gaps in malware detection was made easier by the literature review, which also inspired this study to fill them. The following is a summary of these research gaps:

- In order to account for significant variations in malware kinds, the dataset selection for a reliable model to forecast malware should include samples from extended time periods, ranging from many years. The model for machine learning can Practice with zero-day, polymorphic, and metamorphic malware types. To create a reliable model, use a publicly available dataset that provides information on the number and quality of malware with a wide range of variables.

10.48047/jocaaa.2024.33.08.159

- In contrast to static, dynamic, and hybrid analysis, it must extract a wide range of characteristics from samples. The characteristics might come from strings that represent files, directories, URLs, registries, and PE headers of Windows executables.
- Samples are run using dynamic analysis to get beyond malware obfuscation and packaging. Packing and obfuscation cause a file's content to alter suddenly.
- lIt will take a lot of time and resources, including memory and a powerful CPU, to train a machine learning model with thousands of features and a significant number of data. If a standard computer is required to handle this, like as a laptop, then methods for feature selection that are efficient, astute, and optimised are needed, which can drastically cut down on the amount of features.

OBJECTIVES:

The following goals guide this study's development of a reliable, efficient malware detection and categorisation system.

1. To create a technique for creating a cutting-edge malware detection model, that has a wide range of characteristics from static analysis for efficacy and efficiency, and is able to learn from a huge number of malware samples created over an extended period of time.
2. To conduct a comprehensive empirical evaluation to achieve the above describe objectives and reduce overfitting through the application of ensemble boosting and bagging techniques.

METHODOLOGY

4.1 XGBoost machine learning models

1. DATASETS: This work makes use of a publicly accessible dataset that Roth (2019) released. There are 1.1 million samples in the collection. The dataset's samples span the period from December 2006 to December 2017. There are equal numbers of samples for benign (400,000) and malicious (400,000) kinds.
   In order to construct a supervised machine, unlabelled samples are not included in the dataset model of learning.

10.48047/jocaaa.2024.33.08.159

2. Selection of machine learning algorithm: In this portion of the chapter, supervised machine learning algorithms are examined for computational complexity, and the approach that can assist in creating a machine learning model with minimal computing resources is chosen. There are "n" samples in the dataset. and the "d" number of characteristics. There are n samples.

   Features/dimensions for every sample = d

   SVM, or support vector machine: O is the computational complexity. For a non-linear SVM ML model, max(n,d) min(n,d) 2 (Bottou and Lin, 2007; Haddadpajouh et al., 2021). The complexity of a big dataset with n = 600,000 training samples and d = 2351 features is (600,000*2351*2351) = 3,316.3206 x 109. Memory must be used to store the dataset. As a result, the SVM ML method requires a lot of memory and processing power, otherwise it will take longer.

   k-NN, or k-Nearest Neighbour: The k-NN ML technique requires a lot of memory to maintain all the distances for every sample, as well as O(knd) calculations for k neighbours. K-NN requires a lot of processing power and memory. Power and, as n grows, exhibits poor scalability with a huge dataset. Its complexity is (600,000*2351*2351(.5)) = 68.3959 x109 for a big dataset with n = 600,000 training samples, d = 2351 features, and closest neighbour = 2351(.5).

   ANN stands for artificial neural network.
   For models based on ANNs, the computational complexity is O.
   O n 5 for the backward propagation strategy and n 4 for the forward propagation approach. In addition to increasing the number of cells at the input, many features also demand a lot of memory. With n = 600,000 training samples and d = 2351 features, the complexity of a huge dataset is (600,0005) = 7,776x1020

   XGBoost, or eXtreme Gradient Boosting: The XGBoost ML method has a computational complexity of O(k d x log(n)). "x" is the number of current samples, and "k" is the number of subsequent models for the model's training. X is smaller than n in value. As a result, as compared to the decision tree ML model, the XGBoost further minimises computing cost. Assumin5% effective average samples, the complexity of a big dataset with n = 600,000 training samples and d = 2351 features is (100x2351x30,000xlog(600000) =93.8379x109.

4.2 Unified framework for malware detection

In the framework's analysis and filtering stage, it uses Windows executables as input and does static analysis to extract different kinds of characteristics.

10.48047/jocaaa.2024.33.08.159

Some characteristics are for detecting obfuscated contents of program executables, even if all features are generated from a program executable without actually executing the executable at different degrees of detail. These characteristics include (i) the bytogram, (ii) the file entropy histogram, and (iii) the entropy of each portion of the executable program samples. To keep malware researchers from understanding the purpose of the applications and identifying malware through static analysis, malware programmers obfuscate APIs, file locations, URLs, and registries. The APIs and several pathways for executing the executables are revealed by the dynamic analysis of malware. The paths correspond to printable character strings that are part of the executable. Every string in the executable that has five or more printable characters is extracted and hashed as part of the unified framework's feature extraction process. Each string's hashed values within the executable are utilised to insert the string into one of 104 containers. Large amounts of strings in all executables are uniformly reduced to 104 features for strings in each executable using this method. Depending on the executable's obfuscated content at different granular levels, one of the features that reflect the aforementioned features may contain the obfuscated paths of strings that were not directly extractable.

Every DLL and API call made by the executable uses this method as well. The characteristics that may detect obfuscated content of executables at different granular levels may represent the obfuscated APIs and DLLs that cannot be directly retrieved by static analysis. Features from dynamic analysis are not replaced by this method. However, it may be seen as the counterpart of dynamic analysis characteristics. This study project that employs features from the hybrid analysis is qualified by the combined use of both static analysis features and the equivalent of dynamic analysis characteristics.

10.48047/jocaaa.2024.33.08.159

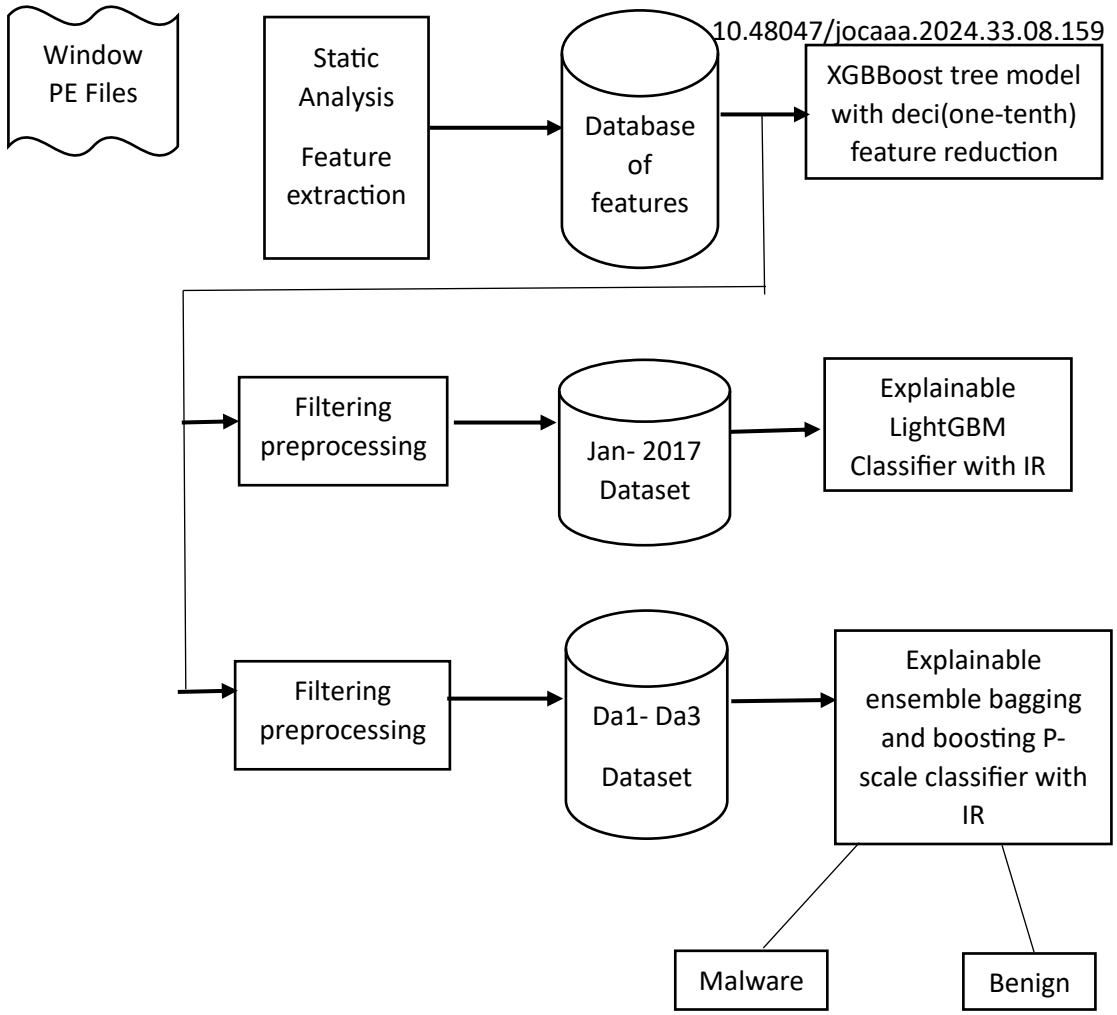


Fig 1.2: Unified Framework for the research work

PROPOSED SYSTEM

Shapley value: It is impossible to comprehend machine learning models. It is challenging to determine the rationale behind a machine learning model's specific prediction for an input sample. Machine learning models that are interpretable provide the rationale behind their predictions. A decision tree-based machine learning model may be interpreted using the decision route, information gain, and heuristic value to characteristics like gini impurity. In this study, a derivative of the Shapley value as a heuristic value makes an ensemble GBDT machine learning model interpretable. The derivative of the Shapley value, which derives from cooperative game theory, concurrently meets the three qualities of local precision, consistency, and missingness (Lundberg et al., 2020). For a feature's contribution, Lundberg et al. (2020) employed the interactive SHapley Additive exPlanation (SHAP) value, which is provided in Eqn. Among the several decision trees created by the boosting machine learning technique, R is the set of characteristics for a certain decision tree algorithms. For every feature, the f(Y) is

10.48047/jocaaa.2024.33.08.159

calculated at random. For each feature in the feature group, this calculation must be performed again.

$$fy(R) = E[\ f(Y)|do(YR = yR)])$$

R is the collection of features for a LightGBM tree construction, y is the input vector for the model's current prediction, and Y is a randomly chosen feature out of a dataset's N features. The exact calculation of the values is NP-hard. It was calculated by Lundberg et al. (2020) using approximation and a slight disturbance of feature value.

Dataset:

Only malware samples from December 2006 to December 2016 are available. During this time, there are no benign samples. Since the dataset is extremely uneven from an origin time, the samples from the era are not included viewpoint. Information on malware, benign, and unidentified samples in the January 2017 dataset is provided in Table 5.1. The experiments in this paper do not include the unidentified samples, which are labelless. Windows executables are used in all examples.

SL.NO.	Count of Samples	Types of sample	Originated in
1.	32761	Malware	2017-01
2.	17180	Benign	
3.	28606	Unidentified	

Table 1.1 types of samples

Feature from each sample in dataset

By splitting a file into blocks of 2048 raw bytes, these characteristics show the entropy of the example file. A window of size 1024 bytes is shifted to create the 2048-byte blocks. Every block's entropy is ordered in order to calculate 256 values in a 16x16 matrix. The unpredictability of a file's content is gauged by its entropy. If the file contains encrypted or obfuscated material, the content becomes even more random. Because of the virus's obfuscated content, features from static analysis of the malware are lacking. A byte histogram of the complete file and the entropy of a sample file at different levels can both make up for this shortfall. 256 features with the designations H1, H2, H3,...H256 are represented by the H1-H256 feature names. After being reduced by 1, the number that comes after H is transformed

10.48047/jocaaa.2024.33.08.159

into a hex value. It shows the number of bytes of that particular value that are contained in the file. Entropy is calculated. both at the section and file levels for the example file. A file's total entropy is calculated and assigned to one of the 50 bins. The names of these 50 characteristics are sec entropy 1 through sec entropy 50. Str1–Str104 are the names of the strings in the example executable that contain five or more printable characters.

Selection of boosting algorithm

In this paper, trials are conducted using the LightGBM Gradient Boosting Decision Tree (GBDT) machine learning technique. The following benefits are the reason the algorithm was selected: Quicker prediction, training and Computational simplicity and the ability to determine the model's feature relevance following training.

System block diagram

The Windows executable is run through Python code that extracts features from the PE header using the module for Instrumenting Executable Files (LIEF) Quarkslab (2019) module. Furthermore, features are taken out as Entropy functions are obtained from all parts of a file, blocks of 2048 bytes with a window of 1024 bytes, and the entire file. Additionally, histograms of bytes and printable text are generated for every executable sample. 2351 features in all are retrieved and sent to the Boosting LightGBM machine learning algorithm.

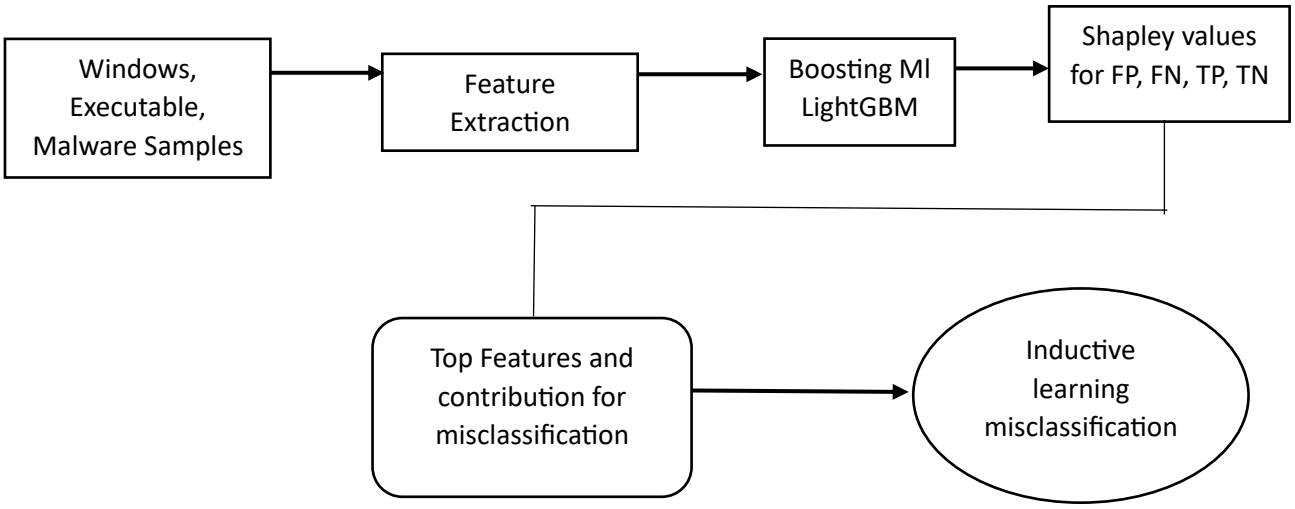


Fig1.3: System block diagram for ML Model

10.48047/jocaaa.2024.33.08.159

EXPERIMENT AND RESULTS

Hardware: A laptop with an i5 CPU and 8 GB of RAM is used for the tests that follow.

Features	Training dataset	Testing dataset	Count of features
Important and selected as in table	300,000 Malware 300,000 Benign	100,000 Malware 100,000 Benign	276

Table 1.2 Dataset of 276 important and selected features

Experiment with feature importance

The analysis of results revealed that although subsets have marginally lower performance inthe range of 2-6% for accuracy than all the features, important features exist in each subset and not in one subset. Some subsets may have a large set of important features and some may have a small set of important features such as subset #4 may have more important features than subset #6 which has the lowest accuracy among all the six subsets. These important features in each subset need to be identified. The mechanism to identify these important features from all the subsets is performed with the feature importance parameter of the XGBoost algorithm. To identify the characteristics that aid in selecting the XGBoost trees, the feature significance of a machine learning model comprising all 2351 features is examined. 2075 characteristics have zero values, whereas 276 features have values larger than zero. feature significance. It revealed that these 2075 attributes contribute zero (no) to the XGBoost algorithm's decision-making process in trees.

Hyperparameter tuning

Number of trees: n estimator

Hyperparameter tuning enhances the performance of the machine-learning model. n estimator and learning rate hyperparameter are used to improve the performance of the XGBoost ML model. Hyperparameter tuning with all features in the dataset takes a very long time. Hence, the XGBoost model built with set-4 features in Table 4.2 is used for hyperparameter tuning. Insight derived from hyperparameter tuning can be applied to all the XGBoost models with a similar dataset. Set-4 had 431 features from the PE header and strings of the executable. XGBoost has n estimator = 100 as a default parameter. n estimator = 200, 300, and 400 are used to measure the performance of the XGBoost model.

10.48047/jocaaa.2024.33.08.159

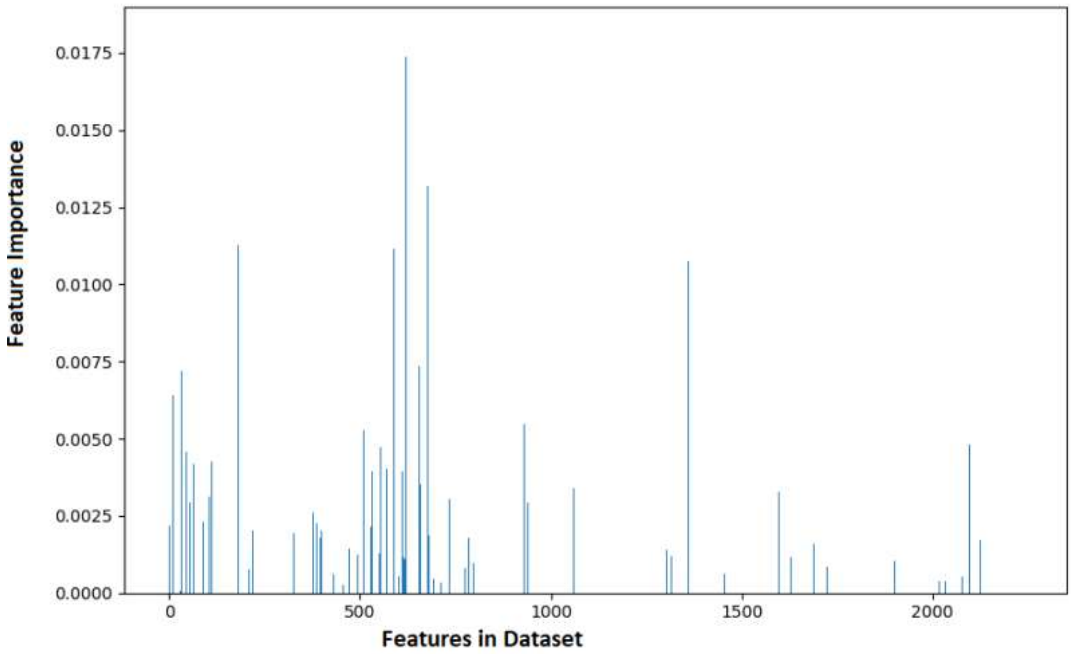


Fig. 1.4 Feature importance of 2351 features based on the XGB model

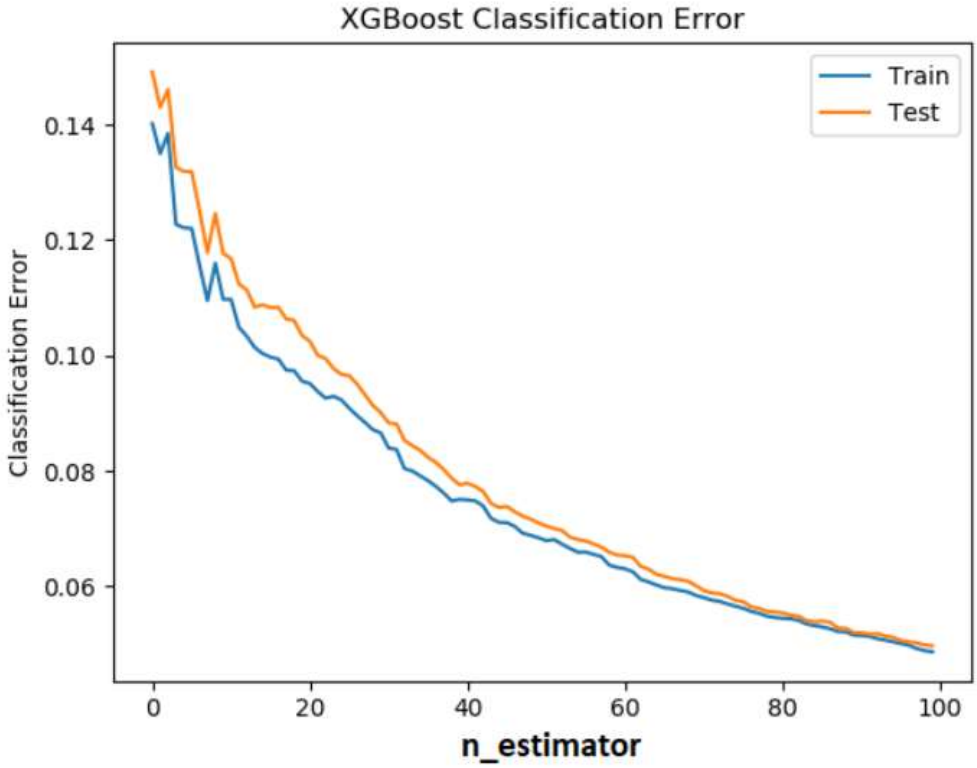


Fig. 1.5 Classification Error for 100 Trees

10.48047/jocaaa.2024.33.08.159

K Fold cross-validation and comparison with ML algorithm

A fresh dataset of 10,000 samples was created for cross-validation using the XGBoost ML algorithm by taking 5000 samples from the testing dataset and 276 selected features from the training dataset, as shown in Table 4.3. With this sub-dataset, cross-validation is carried out using additional machine learning methods such Decision Tree, Adaboost, Random Forest, Liner SVC, KNN, Gaussian Naïve Bayes, and Extratree, among others. Among the machine learning algorithms, XGBoost has the best performance. A comparison of XGBoost with other machine-learning algorithms is demonstrated using cross-validation at smaller sample sizes, which may be expanded for a larger dataset. t. The performance of the XGBoost model for the complete dataset is the same as the tenfold cross-validation at 98.5% vs. 98.22%. The time to complete the cross-validation is the least (47.62 Seconds) for the Extratree machine learning algorithm and the second lowest for XGBoost (61.44 Seconds) among the machine learning models. It is the maximum for Gaussian Na¨ıve Bayes (470 Seconds) machine learning algorithm.

Models	Recall	Precision	F1- score	Accuracy in (%)	Time in second
Extra trees	0.98	0.92	0.94	94.68	47.62
Gradient Boosting	0.98	0.89	0.93	93.16	72.83
Random forest	0.98	0.9	0.93	93.6	141.54
Decision Tree	0.94	0.85	0.89	89.62	177.43
AdaBoost	0.91	0.87	0.89	89.24	105.06
KNN	0.88	0.52	0.65	56.38	307.66
Gaussian Nacve Bayes	0.1	0.43	0.17	51.82	470.37
Linear SVC	0.99	0.48	0.65	49.98	115.62
XGBoost Model	0.98	0.99	0.98	98.22	61.44

Table 1.3 Performance comparison against ten machine learning algorithms

10.48047/jocaaa.2024.33.08.159

CONCLUSION

Every second in this day and age, a new dangerous application appears that might compromise the security of computer and mobile users. The users could be from a wide range of various groups or people. As a result, the infection impacts everyone kinds of organisations, including government agencies, commercial companies, and academic institutions, among others. The goal of malware is to evade detection by advanced methods. This starts a competition to develop sophisticated new malware that uses anti-analysis methods. Another factor is the rapid emergence of new malware varieties, which necessitates the development of new research methods to effectively and reliably identify more complicated malware at the same rate. Complex new malware is not entirely new. It may be a variant or part of the code of existing malware. These patterns may be learnt from existing malware by machine learning algorithms, which then utilise them to identify malware. This research thesis established goals in this regard and accomplished them. But malware detection Research must change as the race does.

The XGBoost ML algorithm had a 98.6% accuracy rate in malware identification. A new method based on information acquisition that influences the characteristics reduces them by 89%. processing power needed to get the machine learning model trained. A typical laptop may be used to construct the model, which has just 276 characteristics that are taken from a vast feature set that has 2351 features in total. In comparison to the computer resources required to create a malware detection model using the initial huge dataset, it is a modest computational resource. The dataset is openly accessible for additional study. It is also capable of detecting malware in real time.

FUTURE SCOPE

Malware is a lucrative business strategy for cybercriminals. Malware writers adapt and surprise users in order to develop assaults that take advantage of weaknesses. Instead, than creating entirely new software, malware developers create covert versions of already-existing malware to accomplish these goals. The methods created and illustrated in this Research will continue to be beneficial. Nonetheless, the method used to build the strategies in this study will open the door for future researchers to come up with a more methodical approach to malware detection in the following field.

- Rules of induction: Better rules to identify undetectable malware (false negative) can be created by further refining the inductive rule-making approaches described in this study.

10.48047/jocaaa.2024.33.08.159

- Malware family: The method described may be used to a single malware family, such as ransomware, rootkit, or crytominer, to improve inductive guidelines to reliably and efficiently detect the malware related to the family type.
- Time savings: High-end computing can benefit from the methods for creating reliable and efficient malware detectors with a cheap computing device in order to free up time for training machine learning models and malware detection.
- Malware on different operating systems: These methods may be used to various operating systems, including Linux, Android, and others, to identify malware. It will assist in enhancing the security status of the organisations that employ them.
- The Internet of Things: The number of new Internet of Things (IoT) devices are growing quickly. The gadgets' low-end CPUs and minimal memory allow for limited computing.
- Machine learning malware detection methods that use adversarial machine learning are vulnerable to hostile assaults.

10.48047/jocaaa.2024.33.08.159

REFERENCES

Afek, Y., Bremler-Barr, A. and Feibish, S. L. (2019), 'Zero-Day Signature Extraction for High-Volume Attacks', IEEE/ACM Trans Netw 27(2), 691–706.

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M. and Giacinto, G. (2016), Novel feature extraction, selection and fusion for effective malware family classification, in 'Proceedings of the sixth ACM conference on data and application security and privacy', pp. 183–194.

Alazab, M., Venkatraman, S., Watters, P. and Alazab, M. (2010), 'Zero-day malware detection based on supervised learning algorithms of API call signatures', Conf Res Pract Inf Technol Ser 121(June 2014), 171–182.

Alcantarilla, P. F., Bartoli, A. and Davison, A. J. (2012), Kaze features, in 'Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12', Springer, pp. 214–227

Aldeid (2017), 'PE-Portable-executable', https://www.aldeid.com/wiki/PE-Portableexecutable. [Online; accessed 12-March-2023].

Almomani, I., Qaddoura, R., Habib, M., Alsoghyer, S., Khayer, A. A., Aljarah, I. and Faris, H. (2021), 'Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data', IEEE Access 9, 57674–57691.

Alzaylaee, M. K., Yerima, S. Y. and Sezer, S. (2020), 'DL-Droid: Deep learning based android malware detection using real devices', Comput Secur 89

AV-TEST-The Independent-IT-Security-Institute, T. I. I.-S. I. (2023), 'AVatlas Malware and PUA', https://portal.av-atlas.org/malware. [Online; accessed 12- March-2023].

Baek, B., Euh, S., Baek, D., Kim, D. and Hwang, D. (2021), 'Histogram Entropy Representation and Prototype Based Machine Learning Approach for Malware Family Classification', IEEE Access 9, 152098–152114.

Bakour, K. and Unver, H. M. (2021 ¨ a), 'DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques', Neural Comput Appl 33(18), 11499–11516.

10.48047/jocaaa.2024.33.08.159

Bakour, K. and Unver, H. M. (2021 ¨ b), 'VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques', Neural Comput Appl 33(8), 3133–3153.

Basole, S., Di Troia, F. and Stamp, M. (2020), 'Multifamily malware models', J Comput Virol Hacking Tech 16(1), 79–92. URL: https://doi.org/10.1007/s11416-019-00345-8

Baysa, D., Low, R. M. and Stamp, M. (2013), 'Structural entropy and metamorphic malware', J Comput Virol 9(4), 179–192.

Bottou, L. and Lin, C.-j. (2007), 'Support Vector Machine Solvers Large Scale Kernel Machines', pp. 1–27.

Ceschin, F., Pinage, F., Castilho, M., Menotti, D., Oliveira, L. S. and Gregio, A. (2019), 'The Need for Speed: An Analysis of Brazilian Malware Classifers', IEEE Secur Priv 16(6), 31–41.

Chen, X., Hao, Z., Li, L., Cui, L., Zhu, Y., DIng, Z. and Liu, Y. (2022), 'CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection', IEEE Trans Inf Forensics Secur 17, 788–803.

David, B., Filiol, E. and Gallienne, K. (2017), 'Structural analysis of binary executable headers for malware detection optimization', J Comput Virol Hacking Tech 13(2), 87– 93.

Dib, M., Torabi, S., Bou-Harb, E. and Assi, C. (2021), 'A Multi-Dimensional Deep Learning Framework for IoT Malware Classification and Family Attribution', IEEE Trans Netw Serv Manag 18(2), 1165–1177.

Egelman, S., Herley, C. and Van Oorschot, P. C. (2013), 'Markets for zero-day exploits: Ethics and implications', ACM Int Conf Proceeding Ser pp. 41–46.

Erfan, M. and Jalili, S. (2021), 'File Fragment Type Classification by Bag-Of-VisualWords', ISeCure 13(2), 101–116.

Fan, Y., Hou, S., Zhang, Y., Ye, Y. and Abdulhayoglu, M. (2018), 'Gotcha - Sly Malware! Scorpion: A Metagraph2vec based malware detection system', Proc ACM SIGKDD Int Conf Knowl Discov Data Min pp. 253–262.

Giampaolo, R. and Foundation, P. S. (2017), 'Psutil python library', https://pypi. python.org/pypi/psutil. [Online; accessed 12-March-2023].