UTM UNIVERSITI TEKNOLOGI MALAYSIA
INTERNATIONAL JOURNAL OF INNOVATIVE COMPUTING
ISSN 2180-4370
Journal Homepage : https://ijic.utm.my/

Performance Comparative Study on Zero Day Malware Detection Using XGBoost and Random Forest Classifiers

Ahmad Faris Aiman Arizal[1], Marina Md-Arshad[2], Adlina Abdul-Samad[3*], Maheyzah Md Sirat[4], Siti Hajar Othman[5]
Faculty of Computing,
Universiti Teknologi Malaysia,
81310 UTM Johor Bahru, Johor, Malaysia
Email: af.aiman@graduate.utm.my[1], marina@utm.my[2], adlina6@graduate.utm.my[3], maheyzah@utm.my[4] hajar@utm.my[5]

Submitted: 21/4/2024. Revised edition: 12/8/2024. Accepted: 19/8/2024. Published online: 25/11/2024
DOI: https://doi.org/10.11113/ijic.v14n2.449

***Abstract*—Zero-day malware is a significant threat to cybersecurity as it is unknown to antivirus systems and can cause significant damage before being detected. Traditional malware detection methods rely on signatures and patterns specific to known malware, but these methods are ineffective against zero-day malware that has not been previously encountered. Machine learning has shown promise in detecting and classifying unknown threats, including zero-day malware. In this research, we propose using machine learning classifiers to detect and classify zero-day malware. The selected classifiers are Random Forest and XGBoost, well-known and widely used in machine learning. To evaluate the effectiveness of our approach, we first collect and pre-process a dataset of known malware. The dataset, Meraz'18, is used to train and test the selected classifiers. This dataset contains PEHeaders with static analysis performed with each section calculated on its entropy. The dataset contains a representation of benign and malicious files that has been use in previous studies for zero-day malware detection, namely during the payload injection phase. To prevent overfitting, 10-fold-cross-validation is utilized. The performance metrics of these classifiers such as F1-score, accuracy, Cohen's kappa, precision and recall analyzed on the known malware dataset and evaluate their ability to detect and classify zero-day malware. Hyperparameter tuning is used to tune each model to give the best performance of each model. The results show that the proposed classifiers perform extremely well, both achieving up to almost 98% accuracy. Using machine learning classifiers for zero-day malware detection and classification can significantly improve cybersecurity by providing a way to detect and protect against unknown threats. This work is an essential step towards the development of more robust cybersecurity systems that can effectively protect against unknown threats.**

***Keywords*—Machine learning, zero-day malware, hyperparameter optimisation, ensemble machin learning algorithms**

I. INTRODUCTION

Computers are becoming more accessible to the general populace as computers become cheaper to manufacture, thanks to advances in semiconductor manufacturing. In recent years, more individuals and families of all wage categories now own at least one computer. Computers simplify human lives by automating complex processes. For example, accounting, project management, Customer Relationship Management, and payment transaction can be automated using a computer with specialized software. As computers see widespread adoption, so does the development of malware. Malware is software designed to either damage, disrupt, or allow an adversary to gain access to an individual or organization's computer [1]. Since the early days of computing, malware has always presented a concern to individuals or organizations for data security and integrity as they usually render the affected computer unusable. To protect against this ordeal, anti-viruses are developed to counter malware.

Despite the development of anti-viruses, the development of malware is still increasing linearly, from 2005 to 2015 from 1 million to 400 million total malware [2]. However, malware nowadays employs zero-day vulnerabilities. Zero-day vulnerabilities refer to a vulnerability unbeknownst to programmers, system administrators and security analysts. It is often seen as a bug in a program's code. Once a malware developer finds a bug in the program's code, they will develop an application or script that will utilize this exploit to its fullest and start to deliver payloads that can cause damage to the computer. Recent examples of this are WannaCry and Petya; both of these malwares used an exploit in Microsoft's Server

Message Block Protocol, which was only discovered after WannaCry and Petya were deployed to the masses. As malware nowadays more frequently using zero days vulnerabilities to infect computers rather than social engineering, it has become a concern for security analysts and anti-virus companies.

To safeguard organization and individual data, a new type of malware detection needs to be developed. In recent years, many scholars have proposed using Machine Learning to detect malware. In the field of malware detection, machine learning algorithms have shown promising results, especially in combating zero-day malware threats. Two notable classifiers that have proven effective in this area are Random Forest and XGBoost. Research by [3] demonstrates that ensemble methods, specifically Random Forest and XGBoost, exhibit superior accuracy, precision, and recall in malware detection compared to other methods. Similarly, [4] found that Random Forest displayed the best accuracy rates with zero false positives and false negatives in detecting zero-day malware. To evaluate a machine learning model, several metrics are set. These metrics are used in the evaluation methodologies of machine learning, such as confusion metrics, accuracy, precision, recall, and F1-Score. Machine learning is an excellent foundation as malware response requires rapid response, especially for zero days, as it is hard to detect malware employing zero days.

Therefore, this research aims to identify and extract relevant features from Microsoft Windows PE Header files for zero-day malware detection and classification. Subsequently, the study will train and compare the performance of Random Forest and XGBoost algorithms in detecting zero-day malware, evaluating their effectiveness using precision, accuracy, recall, and F1-score metrics.

II. Problem Background

The severity of a malware attack could no longer be compared to the early 80s and 90s as there is now more emphasis on extortion. The main objective of these computer viruses at the time was to mess with the end user's computer, and at best, they must send their computer to a computer repair shop. For example, a zero-day malware developed in the late 90s dubbed Chernobyl or Spacefiller renders the target computer useless as the BIOS (Basic Input Output System) is overwritten with zeros or junk. Additionally, to prevent any data recovery efforts, the first megabyte or boot sector in a storage medium is overwritten with zeros, preventing Windows 9x class systems from booting up at the time. Compared to current zero-day malware such as WannaCry in 2018, the malware will render the computer useless as all documents with a selected extension name will be encrypted using a proprietary cryptography algorithm written by the malware developers. Then, a countdown timer will be displayed to the end user regarding how the keys there will be destroyed, and a price will be set by the malware developers detailing how to pay the ransom. Not only will the computer be rendered useless, but documents are far more valuable than a computer in specific use case scenarios.

Organisations data is a precious asset to malware developers as they can hold the data ransom in exchange for monetary value. The development of such ransomware as WannaCry has been estimated to cause damages to corporations at around $4 Billion [5] Not only has it crippled corporations but also public services, such as the NHS UK. NHS UK had to pay [6] to free their data from ransom. An investigation was launched by the National Audit Office of the UK; an estimated 19,000 appointments were cancelled in total due to the severity of WannaCry. Since the WannaCry ransomware types of malicious software emerged, more types of similar malicious software have emerged. One such example is aptly named the Rensenware joke virus, where victims must play a bullet hell the game and reach a specific score to recover their data. The evermore-encompassing world now requires cybersecurity as cyber-attacks are becoming a cause for concern, as it spares no mercy for those involved.

In essence, it is essential to conduct this research to safeguard organisational and individual data. By detecting and classifying malware using machine learning algorithms, a computer can evaluate the malware for itself rather than wait for virus definition updates from an anti-virus company. Additionally, for countries with slow internet bandwidth, a machine learning-based detection and classification method can prove helpful as virus definition updates are downloaded very slowly, thanks to the bandwidth.

III. Literature Review

A literature review is conducted in order to understand the current trends in zero-day malware detection, famous zero-day malware and assessing the use of current algorithms in machine learning used to classify.

A. Review on Zero Day Malware, Malware Analysis and Detection Techniques

This section discusses the current malware detection techniques employed by anti-viruses of the current decade. It is essential to understand these techniques as analysis is a step towards detection and justification of using machine learning algorithms over these methods.

1) Static Analysis

Static analysis is method where an executable is examined for any sign of malicious intent. The examination process is done by reverse engineering the code of the executable to determine if the executable has malicious code or not. For Windows based computers, we can examine the opcode, pe-header, string and API calls for any malicious code. This is done by using tools that break down or test an executable. For instance, tools that commonly used are for malware analysis are debugger, disassembler, decompiler and source code analysers.

2) Dynamic Analysis

Dynamic analysis on the other hand, is a method where a behaviour of an executable is examined. These executables are run in controlled environments, usually in virtual machines (VM) to evaluate the behaviour of the executable. Data then is collected from the executables such as invoked system API calls, Network, Registry, File data are collected for evaluation. Tools that are commonly used for dynamic analysis are usually process hacker, wireshark and cuckoo sandbox.

3) Hybrid Analysis

Hybrid analysis combines Static and Dynamic approaches receive the benefits of both approaches. To illustrate an example, different tools are used to collect static and dynamic features to create hybrid feature sets based on several types of data like string, opcode and API calls [7].

4) Signature-based detection

In an executable, there are sequences of code which we can classify the malware which we call a signature. Each malware is unique, however there are patterns that we can detect in order to classify it as a malware [8] This methodology is used by many anti-virus developers to detect malware that has been discovered previously.

5) Behavioral-based detection

A behavioral-based detection classifies a behavior of an executable if its part of a malware family. It is a set of rules defined by an anti-virus developer to determine its maliciousness. Usually, this method uses a sandbox to evaluate the behaviour of the suspected malicious code. Certain examples where a behavior is flagged as suspicious or malicious are: disabling anti-virus or security controls, registering for autostart, shutting down or installing new system services, altering or deleting system files and in generally performing any action that may seem highly abnormal [9].

6) Heuristic-based detection

Hybrid analysis combines Static and Dynamic approaches receive the benefits of both approaches. To illustrate an example, different tools are used to collect static and dynamic features to create hybrid feature sets based on several types of data like string, opcode, and API calls.

B. Zero Day Malware Background

Zero Day Malware is malicious software using zero-day exploits found in software or hardware to in order to execute a payload in a victims computer. Generally, black-hat or grey-hat hackers take advantage of this zero-day vulnerability as they allow them to again access to a computer easier when compared to a malware where it relies on social engineering, giving the user full administrative rights when running the malicious executable. These zero-day vulnerabilities are a cause for concern as they are very unpredictable and hard to detect. Since zero-day vulnerabilities are present in all software, it is hard to patch it by the end-user unless an update is issued by the developer. Usually, these executables are found in Microsoft Products, namely the Microsoft Office Suite and Adobe software as they are the most used applications in the whole world.

1) Stuxnet

Stuxnet is a malware that is speculated to be developed by the United States and Israeli intelligence agencies respectively [8] with the purpose to delay the Iranian Nuclear Weapons programme. It is classified as a zero day namely because it uses stolen digital certificates made by Siemens, which control the Supervisory Control and Data Acquistion or (SCADA) to gather data. In this case, it was to control and collect data from the nuclear reactor for Iran.

2) Hafnium

Hafnium is another example of zero day malware. It uses a zero day exploit on Microsoft Exchange Servers where the web shell of the malicious script is used by an attacker to maintain persistent access to an online application that has already been compromised. [11]. Using the ProxyLogon vulnerability (CVE-2021-26855), Hafnium web shells were deployed as part of an APT assault to learn more about the companies managing the impacted systems.

C. Review on Machine Learning Algorithm in Malware Detection

For unsupervised learning, it is a type of machine learning in which an algorithm is trained on an unlabeled dataset, where the correct output is not provided. The goal of unsupervised learning is to discover patterns and relationships in the data just like in Fig. 1. Unlike supervised learning, where the algorithm is provided with labeled training examples and learns to make predictions based on that data, unsupervised learning algorithms must discover patterns and relationships in the data on their own. However, in this research, we had applied the supervised learning algorithm such as random forest and XGBoost in malware detection.

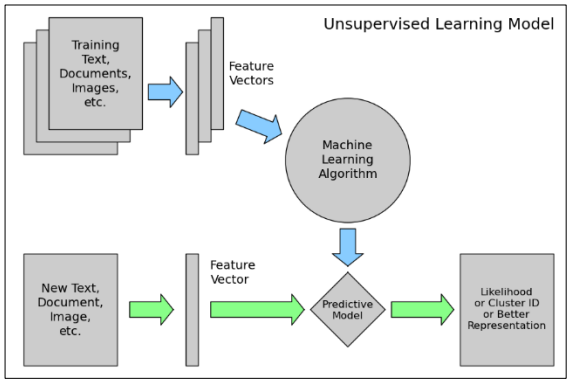

Fig. 1. Unsupervised Machine Learning Model

1) Random Forest

Random Forest is a popular machine learning algorithm used for classification and regression tasks based on Fig. 2. It is an ensemble method that combines multiple decision trees to make a final prediction. Random Forest is known for its robustness, accuracy, and ability to handle large datasets with a high number of features.

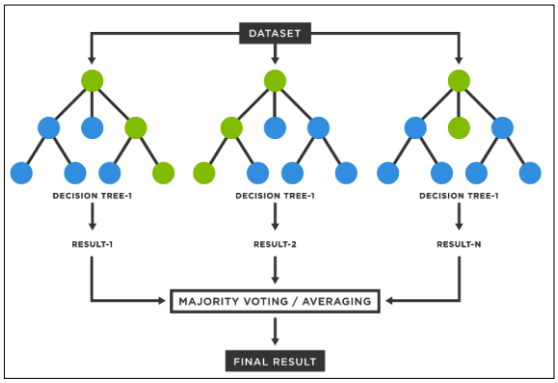

Fig. 2. Random Forest Diagram

The algorithm works by creating multiple decision trees using a random subset of the features and a random subset of the training data. The predictions of all the decision trees are then combined using a voting mechanism to make a final prediction. In a classification task, the majority vote of the decision trees is used to determine the class label, and in a regression task, the average of the predictions is used to determine the final prediction.

2) XGBoost

XGBoost is a distributed gradient boosting library that has been developed to be very effective, adaptable, and portable. It uses the Gradient Boosting framework to implement machine learning algorithms. A parallel tree boosting method called XGBoost (also known as GBDT or GBM) is available to address a variety of data science issues quickly and accurately based on Fig. 3.

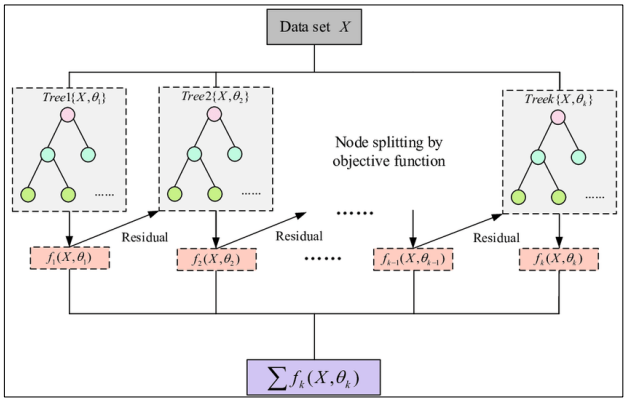

Fig. 3. XGBoost Diagram

The basic idea of gradient boosting is to train weak learners (simple models) in sequence and aggregate their predictions. At each iteration, the algorithm looks at the previous predictions and fits the new model to correct the errors made by the previous models. In XGBoost, the weak learners are decision trees and the algorithm uses a more sophisticated technique called gradient-based one-side sampling to fit the new trees. This helps to reduce overfitting and improve the accuracy of the final model.

D. Related Studies

Zero-day malware is a type of malicious software that exploits unknown vulnerabilities in computer systems. It can be particularly challenging to detect and classify due to its novel nature. Machine learning has emerged as a promising approach for addressing this challenge, and numerous studies have been published on the use of machine learning for zero-day malware detection and classification.

Abri *et al*. investigated the performance of machine learning algorithms, simple neural networks with single layer/multiple layers/multiple layers with a large value for epoch. The result is that all machine learning algorithms except for Gaussian Naïve Bayes and Quadratic Discriminant Analysis (QDA) performed exceptionally well, achieving up to 99 percent accuracy. However, validation is only done using 10-fold cross-validation only [12]. He *et al*. proposed a solution by using Hardware-Supported Malware Detection (HMD) that utilises Hardware Performance Counters (HPCs). It's feature selection is based on Recursive Feature Elimination (RFE) to filter the weakest feature to train the ML algorithm. Boosted-Random Forest gained the highest F1-score at 0.92 [13].

In mobile devices such as android, Yuan et. Al. proposed associating static and dynamic feature analysis to characterise android malware using deep learning algorithms. At different ratios (the ratio of benign/malignant), the overall accuracy increases from 94 percent to 99.54 percent [14]. Nikam et. al. Extracted static features in the AndroidManfiest.xml and Classes.dex then preprocessed the static features into the SciKitLearn library. Then, machine learning algorithms are trained to analyse static features. XGBoost received the highest accuracy at 98.72 percent in classifying malware [15].

Gavrilut et. al. used a Perceptron or known as a Single Layer Neural Network for the detection of malware. To train the perceptron, three datasets were used. One for training, testing and a scale-up dataset. The result of this study is that at the scale-up dataset, the perceptron loses accuracy by quite a margin from 96.16% to 88.52% [1]. Another study Mahajan et. al. proposed using kNN, Naïve Bayes, Neural Networks, Random Forests, Decision Trees and SVM for malware classification. The dataset is self-generated and comprises of 5800 malware samples. Results are using confusion matrix and cohen's kappa. Random Forest achieved the highest accuracy at 94.2 percent and cohen's kappa at 93.8 percent. [16]

Singh *et al*. proposed a system or an application comprised of three modules, the user interfaces, training module, and malware test module. The proposed module is used to extract the data (or static analysis). The dataset used is from Kaggle, specifically the Microsoft Malware Classification Challenge or (BIG 2015). The experiment result is that Decision Tree achieved the highest accuracy at 99.04 percent [17]. In another study, Gandotra et. al. generated their dataset and extracted features from the malware. Evaluation is done using the standard Confusion Matrix and 10-fold cross-validation. All the machine learning classifiers achieved 99 percent accuracy [18].

IV. DATASET

The Meraz'18 dataset is selected for this study as it provides up-to-date samples of malware, with a mixture of zero-day malware. The Microsoft Malware Classification Challenge (MMCC) dataset, while used prominently, is no longer up to date and was last used in 2015. Since then, new malware variants and techniques have emerged, making the MMCC dataset less relevant to the current threat landscape. Using up-to-date dataset such as Meraz'18 can improve the effectiveness of malware detection.

The lack of a publicly available dataset on zero-day malware is a challenge in the field of cybersecurity. The main issue for the scarcity of such a dataset is the fact that most researchers in this field create their own dataset for their specific studies, making it difficult to assemble a comprehensive and standardized dataset for wider use.

The dataset contains Microsoft Portable Executable Headers with calculated entropy for each section. The Table I below shows the features present.

TABLE I. Features present in the Meraz'18 dataset [16]

Fields	Description
Machine	The executable's target computer's CPU architecture is identified by this number.
SizeOfOptionalHeader	The optional header's length
Characteristics	a flag that indicates the file's characteristics, including its executability, whether it's a system file rather than a user programme, and a number of other details.
MajorLinkerVersion	The linker's major and minor version numbers.
MinorLinkerVersion	
SizeOfCode	This column stores the size of the code (.text) part or, if there are many sections, the sum of all the code sections.
SizeOfInitializedData	This column stores the size of the initialised data (.data) section or, if there are many sections, the sum of all initialised data sections.
SizeOfUninitializedData	This variable stores the size of the uninitialized data (.bss) section or, if there are many sections, the total size of all uninitialized data sections.
AddressOfEntryPoint	The entry point is given a Relative Virtual Address (RVA) as soon as the file is loaded into memory. According to the documentation, this relative address relates to the launch method for device drivers as well as the start address for programme images. An entry point is not required for DLLs; instead, the AddressOfEntryPoint field is used.
BaseOfCode	An RVA displaying the start of the code section is generated when the file is loaded into memory.
BaseOfData	An RVA of the start of the data section when the file is loaded into memory.
ImageBase	This field stores the first byte of the image's desired address when it is loaded into memory (the preferred base address)
SectionAlignment	The value (in bytes) stored in this field is used to align memory chunks along boundaries that are multiples of this value.
FileAlignment	This field stores a value (in bytes) that is utilised, like SectionAligment does, to align the raw section data on the disc. The leftover chunk is padded with more data if the size of the actual section data is smaller than the FileAlignment value.
MajorOperatingSystemVersion	These parts of the structure specify the major version number and minor version number of the needed operating system, the major version number and minor version number of the required image, the main version number and minor version number of the required subsystem, and so forth.
MinorOperatingSystemVersion	
MajorImageVersion	
MinorImageVersion	
MajorSubsystemVersion	
MinorSubsystemVersion	
SizeOfImage	The size, in bytes, of the picture file, including all headers. Since this number is needed to load the picture into memory, it is rounded up to a multiple of SectionAlignment.
SizeOfHeaders	The image file's overall size in bytes, including all headers. This amount is rounded up to a multiple of SectionAlignment because it is necessary to load the image into memory.
CheckSum	At the moment the image is loaded, it is utilised to verify the picture and acts as a checksum for the image file.
Subsystem	This parameter defines, among other things, whether the executable image file is NX-compatible and if it can be relocated while in use.
DllCharacteristics	This parameter specifies several properties of the executable image file, such as whether it is NX compatible and if it may be moved while in use.
SizeOfStackReserve	The size of the stack to commit, the size of the stack to reserve, and the amount of local heap space are all specified by these parameters.
SizeOfStackCommit	
SizeOfHeapReserve	
SizeOfHeapCommit	
LoaderFlags	A reserved field
NumberOfRvaAndSizes	Size of the DataDirectory array.

V. FRAMEWORK FOR ZERO-DAY MALWARE DETECTION

This study is structured into three distinct phases depicted in Fig. 4. In the initial phase, a comprehensive review and analysis of the procedures and qualities are conducted, alongside the implementation of data pre-processing techniques and feature engineering techniques. The second phase focuses on the practical application of selected machine learning algorithms through training with hyperparameter optimisation techniques that are selected. Lastly, the final phase involves a detailed examination and discussion of the research findings.

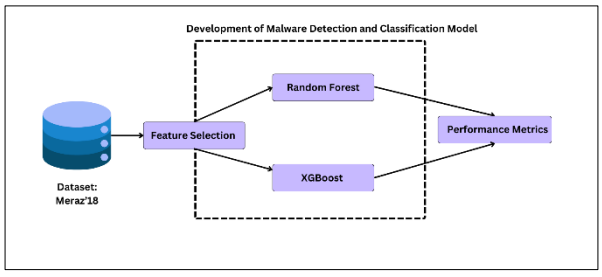

Fig. 4. Framework for Zero-Day Malware Detection

A. Phase 1: Identification of Dataset, Feature Engineering to Train selected Machine Learning Algorithms

In the starting phase of this project, a literature review is conducted on zero-day malware. Existing research namely focuses on regular malware and not zero-day malware. The dataset from Meraz'18 obtained from Kaggle, is chosen and will be used in this experiment. Statistical analysis was done on these files which mainly constituted the extraction of PE information and calculation of entropy of different sections of these files. Python and Scikit-learn has been selected as they are widely used among researchers. Feature extraction and selection is performed in this phase using the ExtraTreesClassifier. These features are then used in Phase 2 for training.

The dataset will be loaded using the Pandas library, and then preprocessed using the scikit-learn library using the SimpleImputer function for missing values. Feature engineering will be performed by using the ExtraTreesClassifier to identify the most prominent features.

1) Data Pre-procesing

The dataset is loaded using the pandas library, the variable data is holding the dataset. We drop non numerical values in the dataset, such as ID, MD5 and legitimate as they are not relevant for feature engineering. Then, by using the imputer library, missing values are filled using the strategy='mean' in the empty values each column. The .csv file contains a lot of rows, hence using the imputer library is useful as it produces meaningful results compared to a set value by the user.

2) Feature Selection using ExtraTreesClassifier

To perform the feature selection, an extra trees classifier is selected. The Extra Trees classifier is a useful method for feature selection as it allows for the identification of the most important features in a dataset for a given classification task. The process involves training the classifier on the dataset, and then using the feature importance attribute to determine the importance of each feature. The feature importance's are computed as the average decrease in impurity from splitting the data on that feature. The top k features with the highest importance score can then be selected and used to train the classifier. This method can be implemented using the scikit-learn library in Python by importing the ExtraTreesClassifier, fitting it to the training data, and accessing the feature_importances attribute. This method can be useful in high dimensional datasets where there are many features, and the goal is to reduce the dimensionality while keeping the most informative features. Figure 5 shows feature heatmap in Meraz'18 dataset.

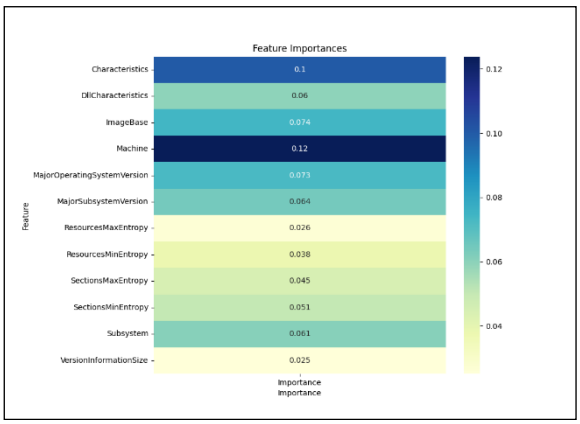

Fig. 5. Feature Heatmap in Meraz'18 dataset

B. Phase 2: Development of Malware Detection and Classification Model.

In this second phase of the research, which utilizes machine learning techniques, a series of crucial steps are undertaken, encompassing comprehensive training of the model and meticulous hyperparameter tuning using Bayesian Optimisation to optimise its performance and enhance its efficacy in identifying and classifying zero-day malware. Table II shows the hyperparameters selected for tuning Random Forest.

TABLE II. Hyperparameters selected for tuning Random Forest

Hyperparameter	Description
n-estimators	Numbers of tree in ensemble
max-depth	Maximum level of trees
min-samples-split	Minimum samples required to be split (in a internal node)
min-samples-leaf	Minimum number of samples in a leaf node
max_samples	Max samples to use for each tree in the ensemble.
max_features	This determines the max number of features to consider when looking for the best split.
class_weight	Handles class imbalance. When set to balanced, the algorithm adjusts the class weights inversely proportional to class frequencies.

The n_estimators, max_depth, min_samples_split and min_samples_leaf are selected based on Table III. These are the common values selected when tuning a random forest classification task. [20]. Then, an instance of BayesianOptimisation is created with parameters from param_bounds_rf and as well as the objective function, is called. Maximise is called in order to start the optimisation process and then the best parameters are selected and printed.

TABLE III. Hyperparameter selected for tuning XGBoost

Hyperparameter	Description
n-estimators	Numbers of tree in ensemble
max-depth	Maximum level of trees
min-samples-split	Minimum samples required to be split (in a internal node)
min-samples-leaf	Minimum number of samples in a leaf node

Similar to Random Forest, the methods are the same, with a few differences in hyperparameters, as XGBoost is a Boosting ensemble learning methods. Learning_rate, max_depth, subsample, colsample_bytree, gamma, scale_pos_weight are selected to be optimised based on the XGBoost. These XGBoost parameters are the most common to be optimised [19]. Then, an instance of BayesianOptimisation is created with parameters from param_bounds_xgb and as well as the objective function, is called. Maximise is called in order to start the optimisation process and then the best parameters are selected and printed.

C. Phase 3: Performance Comparison of trained machine learning models.

After training the selected machine learning algorithms, the performance of each machine learning model is evaluated. This evaluation aims to assess the trained machine learning models generalize to unseen data and to compare algorithms in terms of various selected performance metrics.

VI. RESULTS

From the results in Table IV, it is found that Random Forest is the best performing model, having high accuracy, precision, recall, f1-score and Cohen's kappa. shows the comparison of XGBoost and Random Forest with hyperparameter tuning and without hyperparameter tuning.

Table IV: Algorithm comparison with defined performance metrics.

Metrics	Algorithms			
	Random Forest (%)	XGBoost (%)	Random Forest (%) w/o HP	XGBoost (%) w/o HP
Accuracy	98.6	98.50	98.1	98.48
Precision	98.41	99.1	98.45	98.8
Recall	98.2	98.59	98.62	98.84
F1-score	98.50	98.84	98.53	98.82
Cohen's Kappa score	96.75	96.72	96.0	96.0

The results in Table IV show that hyperparameter adjustment significantly affects how well machine learning algorithms perform. The results show that hyperparameter adjustment significantly affects how well machine learning algorithms perform. When comparing the tuned models, Random Forest outperforms XGBoost in all metrics. However, it is important to note that the differences are very minimal. With hyperparameter tuning, XGBoost achieves an Accuracy of 98.50%, Precision of 99.1%, Recall of 98.59%, F1-score of 98.84%, and Cohen's Kappa score of 96.72%. On the other hand, Random Forest with tuning achieves an Accuracy of 98.6%, Precision of 98.41%, Recall of 98.2%, F1-score of 98.84%, and Cohen's Kappa score of 96.75%.

Comparing the untuned models, XGBoost still performs better, with an Accuracy of 98.48%, Precision of 98.8%, Recall of 98.84%, F1-score of 98.82%, and Cohen's Kappa score of 96.0%. For Random Forest without hyperparameter tuning, the Accuracy is 98.1%, Precision is 98.45%, Recall is 98.62%, F1-score is 98.53%, and Cohen's Kappa score is 96.0%.

Hyperparameter adjustment improves model performance, however most metrics are similar between the two techniques. Hyperparameter tweaking affects datasets and hyperparameters differently. In certain circumstances, hyperparameter adjustment improves performance significantly, while in others it improves performance somewhat.

A. Performance Comparison scenario using Chrome Extension

A sample dataset of 15 samples was used in the performance comparison scenario utilizing the Chrome Extension. Ten of these samples were determined to be zero-day malware, and five to be benign. The effectiveness of the machine learning models in correctly identifying the samples was evaluated using the confusion matrix. Analysis of the confusion matrix's results reveals important information about the models' capacity to distinguish between malicious and benign samples in the real-world application of the Chrome Extension. Table V shows the comparison of XGBoost and Random Forest using the chrome extension, with models that utilised hyperparameter tuning and without hyperparameter tuning.

TABLE V. Comparison of selected algorithms with defined performance metrics used in the chrome extension context

Metrics	Algorithms			
	Random Forest (%)	XGBoost (%)	Random Forest (%) w/o HP	XGBoost (%) w/o HP
Accuracy	73.33	73.33	60.00	66.67
Precision	70.00	60.00	50.00	66.67
Recall	87.5	100.00	83.33	88.89
F1-score	77.78	75.00	60.25	76.19

VII. CONCLUSION

This research has demonstrated the feasibility of utilizing machine learning techniques, specifically Random Forest and XGBoost, for the detection of zero-day malware. By focusing on the extraction of critical features from PE header files, we have established a foundation for effective malware classification. The successful implementation of feature selection methods has underscored the importance of carefully selecting informative attributes for enhancing model performance.

Both Random Forest and XGBoost have exhibited promising results in detecting zero-day malware, as evidenced by the high accuracy, precision, recall, and F1-score achieved. While the performance of these ensemble methods was comparable in this study, further investigations into their strengths and weaknesses under different dataset distributions and malware types would be beneficial. This accomplishment illustrates the successful training and improvement of machine learning classifiers for precise zero-day malware identification and classification.

To summarise, machine learning is a good contender and both ensemble methods are somewhat close in terms of performance in classification. Cybersecurity personell can leverage machine learning algorithms in detecting zero-day malware in where rapid response is required.

VIII. Suggestion for improvement and future works

Bayesian optimization was used in this study as a method for hyperparameter tweaking, which was successful in enhancing the functionality of the machine learning classifiers. However, it might be advantageous to investigate alternate optimization methods, like Particle Swarm Optimization (PSO), for future work. PSO has showed promise in terms of hyperparameter optimization, and it could provide new information and possibly boost classifier performance. The robustness and generalizability of researchers' models can be further improved by taking into account various optimization strategies.

Acknowledgment

We would like to express our heartfelt appreciation to Miss Marina Md Arshad, for her guidance and support throughout the research process. We are truly grateful for her mentorship and for the inspiration she provided during every stage of this research.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

References

[1] Gavrilut, D., Cimpoesu, M., Anton, D., & Ciortuz, L. (2009). Malware detection using machine learning. *2009 International Multiconference on Computer Science and Information Technology.* Doi:10.1109/imcsit.2009.5352759.

[2] Alenezi, M. N., Alabdulrazzaq, H. K., Alshaher, A. A., Alkharang, M. M. (2022). Evolution of malware threats and techniques: A Review. *International Journal of Communication Networks and Information Security (IJCNIS), 12*(3). Doi:10.17762/ijcnis.v12i3.4723 .

[3] Li, Z. (2024). Comprehensive evaluation of mal-api-2019 dataset by machine learning in malware detection. *IJCSIT, 2*(1), 1-9. https://doi.org/10.62051/ijcsit.v2n1.01.

[4] Ekong, A. (2023). Securing against zero-day attacks: a machine learning approach for classification and organizations' perception of its impact. *Journal of Information Systems and Informatics, 5*(3), 1123-1140. https://doi.org/10.51519/journalisi.v5i3.546.

[5] Cybertalk. (2022). 5 years after the first WannaCry attack. Retrieved from https://www.cybertalk.org/5-years-after-the-first-wannacry-attack/#:~:text=WannaCry's%20impact,roughly%20%244%20billion%20in%20damages.

[6] Alford, J. (2019). NHS cyber-attacks could delay life-saving care and cost Millions: Imperial News: Imperial College London. Retrieved January 15, 2023, from https://www.imperial.ac.uk/news/193151/nhs-cyber-attacks-could-delay-life-saving-care/.

[7] Mira, F. (2021). A systematic literature review on malware analysis. *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS).* Doi:10.1109/iemtronics52119.2021.9422537

[8] Inayat, U., Zia, M. F., Ali, F., Ali, S. M., Khan, H. M., & Noor, W. (2021). Comprehensive Review of Malware Detection Techniques. *2021 International Conference on Innovative Computing (ICIC).* Doi:10.1109/icic53490.2021.9693072

[9] Aboaoja, F. A., Zainal, A., Ghaleb, F. A., Al-rimy, B. A., Eisa, T. A., & Elnour, A. A. (2022). Malware detection issues, challenges, and future directions: A survey. *Applied Sciences. 12*(17), 8482. Doi:10.3390/app12178482

[10] Baezner, M., & Robin, P. (2018.). Stuxnet. Retrieved January 15, 2023, from https://www.researchgate.net/publication/323199431_Stuxnet.

[11] Gatlan, S. (2022). Microsoft: New malware uses windows bug to hide scheduled tasks. Retrieved January 15, 2023, from https://www.bleepingcomputer.com/news/security/microsoft-new-malware-uses-windows-bug-to-hide-scheduled-tasks/.

[12] Abri, F., Siami-Namini, S., Khanghah, M. A., Soltani, F. M., Namin, A. S. (2019). Can machine/deep learning classifiers detect zero-day malware with high accuracy? *2019 IEEE International Conference on Big Data (Big Data)*. Doi:10.1109/bigdata47090.2019.9006514.

[13] He, Z., Miari, T., Makrani, H. M., Aliasgari, M., Homayoun, H., Sayadi, H. (2021). When machine learning meets hardware cybersecurity: Delving into accurate zero-day malware detection. *2021 22nd International Symposium on Quality Electronic Design (ISQED).* Doi:10.1109/isqed51717.2021.9424330.

[14] Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Science and Technology, 21*(1), 114-123. Doi:10.1109/tst.2016.7399288

[15] Nikam, U. V., & Deshmuh, V. M. (2022). Performance evaluation of machine learning classifiers in malware detection. *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE).* Doi:10.1109/icdcece53908.2022.9793102.

[16] Mahajan, G., Saini, B., Anand, S. (2019). Malware classification using machine learning algorithms and Tools. *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*. Doi:10.1109/icaccp.2019.8882965.

[17] Singh, P., Kaur, S., Sharma, S., Sharma, G., Vashisht, S., & Kumar, V. (2021). Malware detection using machine learning. *2021 International Conference on Technological Advancements and Innovations (ICTAI).* Doi:10.1109/ictai53825.2021.9673465.

[18] Hesham, A. (2021). A dive into the PE file format - introduction. Retrieved January 17, 2023. https://0xrick.github.io/win-internals/pe1/.

[19] Hosseini, S. (2023). A practical guide to hyperparameter tuning of XGBoost models using Bayesian optimization and grid. https://medium.datadriveninvestor.com/introduction-31c985114aa1.

[20] Wicaksana, P. Y. (2022). https://medium.com/@prabowoyogawicaksana/hyperparameter-optimization-random-forest-classifier-550fd5ed8e14.