Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.041038

ARTICLE

Hybrid Malware Variant Detection Model with Extreme Gradient Boosting and Artificial Neural Network Classifiers

Asma A. Alhashmi[1], Abdulbasit A. Darem[1,*], Sultan M. Alanazi[1], Abdullah M. Alashjaee[2], Bader Aldughayfiq[3], Fuad A. Ghaleb[4,5], Shouki A. Ebad[1] and Majed A. Alanazi[1]

[1]Department of Computer Science, Northern Border University, Arar, 9280, Saudi Arabia

[2]Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, 91911, Saudi Arabia

[3]Department of Information Systems, College of Computer Science and Information, Jouf University, Sakaka, Aljouf, Saudi Arabia

[4]School of Computing, University Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, 81310, Malaysia

[5]Department of Computer and Electronic Engineering, Sana'a Community College, Sana'a, 5695, Yemen

*Corresponding Author: Abdulbasit A. Darem. Email: basit.darem@nbu.edu.sa

Received: 08 April 2023 Accepted: 01 July 2023 Published: 08 October 2023

ABSTRACT

In an era marked by escalating cybersecurity threats, our study addresses the challenge of malware variant detection, a significant concern for a multitude of sectors including petroleum and mining organizations. This paper presents an innovative Application Programmable Interface (API)-based hybrid model designed to enhance the detection performance of malware variants. This model integrates eXtreme Gradient Boosting (XGBoost) and an Artificial Neural Network (ANN) classifier, offering a potent response to the sophisticated evasion and obfuscation techniques frequently deployed by malware authors. The model's design capitalizes on the benefits of both static and dynamic analysis to extract API-based features, providing a holistic and comprehensive view of malware behavior. From these features, we construct two XGBoost predictors, each of which contributes a valuable perspective on the malicious activities under scrutiny. The outputs of these predictors, interpreted as malicious scores, are then fed into an ANN-based classifier, which processes this data to derive a final decision. The strength of the proposed model lies in its capacity to leverage behavioral and signature-based features, and most importantly, in its ability to extract and analyze the hidden relations between these two types of features. The efficacy of our proposed API-based hybrid model is evident in its performance metrics. It outperformed other models in our tests, achieving an impressive accuracy of 95% and an F-measure of 93%. This significantly improved the detection performance of malware variants, underscoring the value and potential of our approach in the challenging field of cybersecurity.

KEYWORDS

API-based hybrid malware; detection model; static and dynamic analysis; malware detection

1 Introduction

The sectors like mining and petroleum sectors are increasingly vulnerable to Malware attacks due to their reliance on information technology and their use of integrated systems and data. Malicious

software also known as malware, has increased rapidly. Malware has a damaging effect on computer systems, digital assets, and the community [1]. Malware can be used to spy on users, steal sensitive information, disseminate fake news, destroy cyber-physical systems, perform assassinations, and many other acts of evil [2]. The malware creation tools accessible to a wide range of malware developers, including amateurs and script kiddies, lead to a proliferation of malware. Various types of malware have been created, including viruses, worms, trojans, botnets, ransomware, and advanced persistent threats, to name a few [3]. Numerous malware samples emerge every day. McAfee and Symantec report that 69 new malware are detected every minute [4]. Most of this malware comprises malware variants [5]. Existing malware is often re-engineered to produce some evasive and obfuscated malware. Developers of malware often employ obfuscation strategies with the intent to disguise their true characteristics and activities.

Various methodologies have been proposed in the realm of malware detection, and these can be categorized according to the analysis type and the nature of features extracted during static and dynamic analyses [6,7]. The static approach pertains to the extraction of features from portable executable files without necessitating the execution of the malware. On the other hand, dynamic features are gleaned from the interactions occurring between the malicious software and the Operating System (OS). This interaction takes place within a controlled, isolated environment. Malware actions are captured and used to extract features that represent the malware behavior. Both static and dynamic analysis have been extensively researched. Dynamic features are more effective compared to static features due to the complexity of extracting representative features using static analysis and the vast amount of data in the malware files. Recently, many researchers have utilized deep learning techniques to automatically extract representative features and construct accurate classifiers using static analysis-based features. These techniques have substantially escalated the proliferation of new malware variants capable of penetrating existing protective solutions. Nonetheless, identifying obfuscated and evasive malware presents a formidable challenge, given their high resemblance to benign programs and the elusive characteristics of their behavior.

The majority of existing methodologies for malware detection rely predominantly on either static or dynamic analysis as the fundamental mechanisms for feature extraction [5,6,8,9]. These features can become sparse and inadequate to represent malware behavior due to the injection of random and benign API sequences within the malware files or the obfuscation of real behavior during runtime. A few ensemble and hybrid models have been proposed that use dynamic and static-based features, as in [5,10]. Most of the hybrid approaches have focused on combining different classifiers. Only a few models have focused on the use of hybrid features, such as [5,10]. Integrating static and dynamic features in one model is not a trivial task, as the decision of maliciousness becomes more challenging. Within the context of this investigation, features from both static and dynamic API-based analyses were harnessed to establish two variants of ensemble classifiers via the application of the XGBoost algorithm. The underlying justification for amalgamating API features drawn from the disparate analytical modes lies in the potential divergence between APIs discerned from static analyses *vs.* those identified through dynamic analyses-an outcome attributable to the variance in employed identification methods. The integration of static and dynamic analysis methodologies may yield a holistic understanding of the APIs that malware exploits, with each analytical approach contributing distinct insights and uncovering a unique set of problems. Because API-based features have been reported to be among the most effective representative features, the scope of this study focuses on such types of features. Similarly, XGBoost and Artificial Neural Networks (ANN) have been frequently reported to be effective learners. Both aspects are integrated within the architecture of the model proposed within this study. In this research, we are primarily driven by the rapidly

increasing incidence of malware, particularly the more sophisticated malware variants, which presents a significant threat to individual and organizational security. Traditional detection methods, whether behavioral or signature-based, often fall short of identifying these advanced threats due to their evasion and obfuscation techniques. This alarming situation motivated us to develop a more effective solution for malware variant detection, aiming to improve the security posture of vulnerable entities.

The study presents a twofold main contribution. It extracts both static and dynamic API features to train predictive models using the XGBoost algorithm. These models consist of an ensemble of decision tree predictors, which sequentially score the sample class. The ensemble's output trains ANN classifiers based on a multilayer perceptron. XGBoost, a tree-based algorithm, excels in handling structured data, outliers, and missing values. On the other hand, ANN, a flexible and adaptable algorithm, deals well with unstructured data and learns complex patterns.

The results demonstrate that the proposed hybrid ensemble model surpasses non-hybrid and single ensemble-based classifiers. The study provides the following contributions:

1. Extraction of hybrid API features from dynamic and static analysis. This combination offers a more comprehensive and accurate understanding of malware behavior, leading to more effective detection and mitigation of threats.
2. Development of a hybrid ensemble learning-based model (API-HMVD) architecture. It stacks an ANN-based model onto the outputs of XGBoost classifiers to learn the correlation between patterns detected for each feature type. The XGBoost model extracts features and makes preliminary predictions, which then enhance accuracy when fed into the ANN model to learn from intermediate results.

Additionally, the study proposes a novel hybrid malware variant detection model that leverages an Application Programmable Interface (API). This model employs both static and dynamic analyses to extract API-based features. These features are utilized by two predictors built on eXtreme Gradient Boosting (XGBoost), providing a more comprehensive view of malware behavior and improving detection accuracy. The output of these predictors is then utilized by an Artificial Neural Network (ANN) classifier to make the final decision, effectively revealing the hidden relationship between behavioral and signature-based features.

The structure of this paper is laid out as follows: Section 2 provides a comprehensive review of the related literature. A detailed elaboration of the research's proposed model is laid out in Section 3. The experimental framework and evaluation metrics are delineated in Section 4. A thorough discussion and analysis of the results are found in Section 5. Lastly, Section 6 presents the concluding thoughts and notable findings derived from this study.

2 Related Work

The efficacy of malware detection models hinges on two key elements: feature extraction and model design. Features are typically classified based on their analysis type, either static or dynamic. Various types of features can be extracted through these analyses [11]. For instance, static features might be derived from Portable Executable (PE) headers and sections, byte codes, opcodes, flow graphs, strings, imported application programable interfaces (APIs), libraries and functions, and logical architectures [2,9,12]. On the other hand, dynamic features might come from the malware program's interaction with the Operating System's file system, registry, network, memory, Central Processing Unit (CPU), and API call sequences. The extraction of API call features from both dynamic

and static analysis has been a common practice in previous studies [3,13–21]. Table 1 summarizes key methods used for malware detection in recent literature.

Regarding model design, supervised machine learning techniques such as Support Vector Machine (SVM), Random Forest [3,22–24], Extreme Gradient Boost XGBoot [22,25–28], and Artificial Neural Network (ANN) [7,14,19,29–31] are often employed to understand the correlation between the input features and class label. Recently, deep learning has been utilized for training classifiers that can leverage various types of features. However, the majority of existing deep learning-based models primarily focus on static-type features [32]. Given the complexity of malware representation, recent studies have focused on integrating various feature types and classifiers to enhance detection performance. Zhang et al. [5] proposed a hybrid malware variant detection approach that uses static features such as opcodes and API calls. Their model consists of two classifiers: a computational neural network designed to train the first classifier based on opcode-based features, and the multilayer perceptron algorithm used to construct the second classifier, which relies on API calls extracted from static analysis. Al-Hashmi et al. [11] have proposed a comprehensive and composite model designed for the detection of malware variants. This model employs an assortment of behavioral features derived from dynamic analysis to establish a multitude of classifiers using deep sequential learning methodologies.

Table 1: Summarizes key methods used for malware detection in recent literature

Reference	Method	Features	Classification algorithm
Zhang et al. [5]	Static and dynamic analysis	Opcodes, API calls	CNN, ANN
Al-Hashmi et al. [11]	Static analysis	API calls, registry access, file access, and network traffic	XGBoost, SVM
Wang et al. [33]	Static analysis	String feature, structure feature	SVM, KNN, RV
Kang et al. [34]	Static and dynamic analysis	Byte sequence, API sequence run time behavior graph	Random forest, XGBoost
Rieck et al. [35]	Dynamic analysis	API call	(ANN), (DT), Naïve Bayes (NB), SVM
Tian et al. [36]	Static analysis	User interaction functions, coverage rate (CR), API	SVM, KNN, decision tree, R.F

On the other hand, Wang et al. [33] have advanced a technique specifically tailored for the detection of malware variants that capitalizes on static features. These encompass elements such as strings, permissions, specifications of hardware and software, intents, API calls, opcode, and the function call graph. The authors have further classified these features into two primary groupings-one which is string-based and another that is structure-oriented. Despite the potential advantages of these methods, their dependence on static features can be a limitation as malware authors often use basic obfuscation techniques to hide malevolent patterns within binary code, reducing the effectiveness of these detection methods. To address this issue, Kang et al. [34] suggested a holistic approach that

integrates both static and dynamic analysis features. These features are subsequently leveraged to construct an ensemble model, which exhibits the capability to effectively identify various malware variants.

In their research, Rieck et al. [35] delivered a thorough examination of the automatic analysis of malware behavior through the lens of machine learning. The team proposed a unique framework, dubbed "MALheur", which is adept at identifying and categorizing malware according to its behavioral patterns, utilizing dynamic analysis. The framework harnesses a blend of feature extraction and machine learning algorithms for the analysis of malware behavior. The researchers executed extensive experiments on a sizable malware sample set to evaluate their framework's effectiveness.

Tian et al. [36], on the other hand, proposed the use of a class-level dependence graph alongside a method-level call graph as representative depictions of an application. By doing so, they were able to extract static behavioral features, which they used to identify Android malware.

In summary, numerous strategies have been proposed to improve malware detection model accuracy by integrating different feature sets. However, these approaches often overlook the correlations between the features due to the use of a single classifier for each feature type. Moreover, they predominantly rely on either static or dynamic features, which can be easily masked by malware authors employing obfuscation and evasion techniques. Addressing these identified gaps, this investigation encompasses the extraction of both static and dynamic API-based attributes to represent malware. The employed methodology incorporates a two-tiered classifier system. In the initial phase, latent characteristics from both dynamic and static attributes are independently extracted via the application of the XGBoost algorithm. Subsequently, in the second phase, a multilayer perceptron algorithm is utilized to decipher the correlations existing within these latent features. This process allows for the discovery of new hidden patterns characterizing malware variants, leading to improved detection performance. The following section provides a comprehensive description of our proposed model.

3 The Proposed Model

The proposed Application Programable Interface based Hybrid Malware Variant Detection (API-HMVD) model consists of four main phases: feature extraction using both static and dynamic analysis, feature representation, prediction model construction, and decision-making phase. As illustrated in Fig. 1, the structural framework of our proposed model is presented. Comprehensive insights into each phase of this model are elaborated upon in the respective subsections that follow.

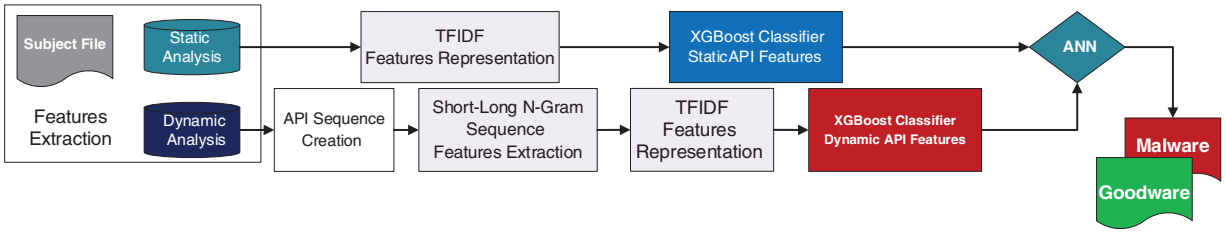

Figure 1: The proposed application programable interface based hybrid malware variant detection API-HMVD model

3.1 Features Extraction

During the initial phase, we engaged in a meticulous extraction of features. This extraction was guided by a two-pronged approach, incorporating elements of both static and dynamic analysis for a more comprehensive understanding of the dataset.

3.1.1 Static Analysis-Based API Features

In this step, the API features have been extracted from the PE files without executing their files. All the APIs that the suspect files intend to invoke are collected regardless of whether they are executed or not. API features are extracted from the import table of the PE files. Each program incorporates the API functions that are exported by other programs and libraries as part of code reuse for efficient and effective software development. Malware authors usually inject random APIs to evade detection by hiding malware patterns using polymorphic and metamorphic techniques.

3.1.2 Dynamic Analysis-Based API Features

Programmers, including malware authors, typically employ dynamic API loading to conceal API features from reverse engineers. Many operating systems support dynamic loading, e.g., Windows Operating System (OS) uses two well-known APIs for loading the APIs during the execution at runtimes, such as LoadLibraryA and GetProcAddress. Once such functions are used by malware, malware can access many API functions without writing directly to the PE file. Therefore, dynamic analysis is an important element of malware detection. In this study, API calls have been monitored in the sandbox environment. Sandbox records the APIs that are invoked by the program in a process called API hooking. When PE files are executed, any API calls are intercepted by the API hooking function. After the PEs are executed, a list of APIs is ordered according to their appearance in the hooking process. Thus, an API sequence can be formed to represent the behavioral activities of PE.

3.2 Representation Phase of Features

In this phase, we represent both the attack signature and the behavioral features. The N-gram technique was used to create the attack signature and the behavioral sequence of the extracted features using static and dynamic analysis. Then, a statistical technique called term frequency/inverse document frequency (TF/IDF) is used to represent the API features [19,31]. TF/IDF can convert APIs from textual format to their equivalent numerical weights. It can evaluate how important an API function is to a PE sample compared to the other samples. TF/IDF is applied to a single API function that is extracted from static analysis and applied to a sequence of APIs consisting of two or three APIs that occurred in order. In this study, the short sequence consists of one API function, and the longest sequence consists of four API functions. Each sequence is treated as a term. The *tf/idf* can be calculated as follows.

$$tf\left(API_f\right) = \frac{number\ of\ times\ API_f\ in\ a\ sample}{Total\ number\ of\ API\ functions\ in\ the\ sample} \tag{1}$$

$$idf\ \left(API_f\right) = \log\left(\frac{number\ of\ a\ sample}{number\ of\ documents\ that\ have\ API_f}\right) \tag{2}$$

$$tf/idf\left(API_f\right) = tf\left(API_f\right) \times idf\ \left(API_f\right) \tag{3}$$

3.3 Prediction Models Construction Phase

In this phase, two predictive models were constructed, each of which has been trained based on the API functions extracted from the static and dynamic analysis. The Extreme Gradient Boosting XGBoost algorithm [25] was used to construct predictive models. The XGBoost algorithm constructs a set of weak learners using boosting and decision tree algorithms. In the boosting, the trees are constructed sequentially in such a way that the residual error generated from the previous tree is considered during the building of the subsequent tree. Each tree is trained based on the lessons learned from its predecessors. That is, each tree tries to reduce the error resulting from the previous trees in the sequence. The grown trees are relatively small regression trees. Therefore, the model is highly interpretable due to the limited number of splits. Nodes and splits in the tree are chosen based on a similarity score (formula (4)). The following steps explain how XGBoost constructs the trees.

1- Create the first tree that consists of the root node with a single-leaf node.
2- Use the tree that was created in the previous step to make the prediction and then calculate the residual error.
3- Use the following equation to calculate the similarity score.

$$Similarity\ Score = \frac{Gradient^2}{Hessian + \lambda} \tag{4}$$

where $Gradient^2$ is the square root of the residuals, *Hessian* denotes the number of the residuals, and λ is a hyperparameter to make regularization and prevent dividing by zero.
4- Select the node with the highest similarity score to achieve homogeneity.
5- Calculate the information obtained for each split using the similarity score as follows.
6- $$information\ gain\ = left\ similarity + right\ similarity - root\ similarity \tag{5}$$
7- Continue to construct the tree by creating the leaves and the decision nodes based on the information gained with pruning by adjusting the regularized parameter λ.
8- Use the constructed tree to predict the target class of the samples and calculate the residual error.
9- Compute the new residual as follows.

$$New\ Residual = Old\ Residual + \rho \sum predicted\ residual \tag{6}$$

where ρ denotes the learning rate.
10- Repeat these nine steps to construct the subsequent tree, and all trees are created.

The output of the decision trees for each trained model is aggregated using the following formula.

$$y_k = \sum_{i=1}^{m} f_i \tag{7}$$

where y_k is the aggregated output of the model k, m number of trees and f_i is the predicted value of the *ith* tree.

3.4 Decision-Making Phase

In this phase, the aggregated output of the predictive models from the previous phase was used to build a classifier for the final decision on the sample class. This work hypothesizes that the features used to represent benign and malicious samples might be sparse and contain outliers. Consequently, XGBoost may not fit well for such samples, leading to misclassifying such samples and causing an increase in false positive and negative rates. Therefore, a new classifier that uses the output of the week

classifiers as a new feature is needed to learn the hidden relationships between the XGBoost predictions and the output class.

In this study, the artificial neural network, namely, the multilayer perceptron algorithm [37], has been trained to recognize fake patterns. Neural networks offer the advantage of self-learning capabilities, which can detect hidden patterns within a given training dataset [38]. To train the Multi-layer Perceptron (MLP) classifier for decision-making, the feed-forward with backpropagation algorithm was used. The established model operates on a three-layered structure which includes an input layer, an intermediary or hidden layer, and finally, an output layer. Each predictor stemming from the XGBoost algorithm is linked to respective neurons present within the output layer, thus forming a structured network of information processing. The chosen activation function for each neuron residing in the hidden layer is the Rectified Linear Unit (ReLU) function, renowned for its efficiency and performance in deep learning contexts. To identify the optimal number of neurons to be included in this hidden layer, a process of iterative testing and refinement was undertaken, ultimately allowing for a balance between complexity and predictive power within the model. The sigmoid function was used for the final decision in the output layer. The MLP model is fully connected, which means that the neurons in one layer are connected to all the neurons in the next layer. Weights are allocated to the neurons in both the input and output layers. The optimization algorithm's role is to find the best or optimal set of weights that improve classification accuracy. Fig. 2 shows the structure of the MLP model. If the predicted value p_o is greater than 0.5 then the input features belong to a malware sample; otherwise, it is a benign sample. The output decision was determined based on the following formula.

$$p_o = \frac{1}{1 + e^{-\sum_{i=1}^{m} y_i}} \tag{8}$$

where y_i is the aggregated output of the XGBoost predictors and m number of the predictors.

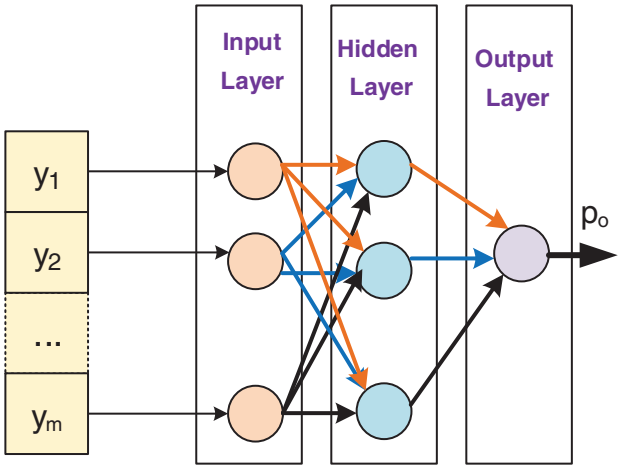

Figure 2: Structure of the multi-layer perceptron model

4 Experimental Setup

The model employed in this research is constituted of two primary components: feature extraction and classification processes. The feature extraction stage involves the extraction of API-centric attributes from malware exemplars, utilizing both static and dynamic techniques. The motivation behind this is to encompass both the configurationally and operational aspects of the malware. The

second segment of the model is dedicated to classification, organized in a bi-level structure. In the initial level, the XGBoost algorithm is leveraged to discern the association between the designated features and their corresponding class tags. The selection of hyperparameters for the XGBoost algorithm was accomplished through a methodical grid search procedure. Parameters selected include a 'max_depth' of 5, 'min_child_weight' of 1, and 'gamma' of 0, while 'subsample' and 'colsample_bytree' were both configured to 0.8. The XGBoost model generates an output referred to as 'malicious scores', which subsequently serve as input for the second level of the classification process. In this stage, a classifier based on an Artificial Neural Network (ANN) is utilized to identify the obscured correlations between behavioral and signature-based characteristics. The configuration of the ANN model comprises a feed-forward network design with three distinct layers. The initial layer, the input layer, is designed with a number of nodes equivalent to the total number of features. Following this, the hidden layer incorporates 10 nodes and employs the Rectified Linear Unit (ReLU) as the activation function. The final layer, the output layer, consists of a single node representing the anticipated class label. The model's learning rate was configured at 0.01 and training was conducted for a total of 100 epochs with a batch size of 32. The model's architecture and parameter settings were carefully chosen to ensure optimal performance, while maintaining computational efficiency.

The collections of data leveraged for both the training and evaluation of the proposed model incorporate two distinct classifications of software. These include benign, or non-harmful, software, as well as malicious or malware software instances. Malicious samples were collected from the VirusShare web portal (https://virusshare.com/), while benign samples were collected from freshly installed OS and trussed repositories. The dataset encompasses a variety of malware samples, including viruses, Trojan horses, ransomware, backdoors, and others. Most malware samples are variants that are generated from old malware versions. To avoid detection, malware authors employed obfuscation and evasive techniques. The final data set comprises 11,712 samples, including 6,877 malware and 4,835 benign software samples. The dataset was divided into two subsets, with 60% allocated for training and 40% for testing. Table 2 shows the statistics of the dataset used in this study.

Table 2: Number of samples in the data set

	Malware	Benign
Training	4,126	2,901
Testing	2,751	1,934
Total	6,877	4,835
Dataset size	**11,712**	

The static features were extracted directly from the PE, while the dynamic features were obtained from the cuckoo sandbox. The model is validated using six performance measures: accuracy (ACC), recall, precision, F-M, FPR, and false FNR. In the context of our research, we operationalize the concept of 'accuracy ratio' as the proportion of samples that have been correctly classified, relative to the overall quantity of samples in the testing set. The metric known as 'Recall' is quantified by the division of the quantity of accurately classified malware by the comprehensive count of malware present within the test set. The measurement of 'Precision' is arrived at by determining the fraction of accurately predicted malware over the total count of predicted malware instances. The F-measure (F-M) is a metric that evaluates the overall performance of the model. The F-measure, often used as a balanced performance indicator, represents the harmonic mean of precision and recall. A higher

F-measure generally indicates decreased error rates in the classification process. The FPR, a key parameter in error assessment, is derived by taking the ratio of benign samples that have been inaccurately classified to the entirety of actual benign samples. Similarly, the FNR is ascertained by calculating the proportion of malware samples that have been erroneously classified over the total count of actual malware samples.

$$Accuracy = \frac{Total\ number\ of\ samples\ which\ are\ correctly\ classified}{Total\ number\ of\ samples} \tag{9}$$

$$Precision = \frac{Totalnumber\ of\ positive\ samples\ which\ are\ correctly\ classified}{Total\ number\ of\ positive\ classified\ samples} \tag{10}$$

$$Recall = \frac{Total\ number\ of\ positive\ samples\ which\ are\ correctly\ classified}{Total\ number\ of\ positive\ samples} \tag{11}$$

$$F - \text{Measure} = \frac{2 * Precision * Recall}{Precession + Recall} \tag{12}$$

$$FPR = \frac{\text{number of misclassified benign samples}}{number\ of\ actual\ benign\ samples} \tag{13}$$

$$FNR = \frac{\text{number of misclassified malware samples}}{number\ of\ actual\ malware\ samples} \tag{14}$$

The presented model's performance was evaluated relative to two other internally developed models and three state-of-the-art models from the literature. The two internally developed models utilized the XGBoost algorithm for training. The first was informed by API functions derived from static analysis, while the second relied on API functions from dynamic analysis. Subsequently, a comparative analysis was also conducted against the state-of-the-art models documented in [39–41]. The model referenced in [39] employed dynamic analysis for API feature extraction, and multiple classifiers-Random Forest (RF), Support Vector Machines (SVM), Logistic Regression (LR), and Naive Bayes (NB) were used for training. LR emerged as the most effective classifier in this case. In [40], an ensemble model was created using SVM as the base classifier, with API features represented using the doc2vec approach. Lastly, the model in [41] extracted API calls through dynamic analysis and harvested Indicator of Compromise features, representing them through the Term Frequency-Inverse Document Frequency (TF/IDF) technique. The LR algorithm was employed to formulate the detection model in this instance.

5 Results Analysis and Discussion

Table 3 shows the training performance of the proposed model compared to other models. In terms of overall accuracy performance, the proposed hybrid model achieved better training compared to the others. Figs. 3 and 4 present the performance comparison between the models studied.

The results from testing the model after fitting, which are shown in Table 4, Figs. 5, and 6, reveal that the proposed model outperforms the others. This suggests that hybrid features derived from both static and dynamic analysis surpass individual feature types in effectiveness. This is clear from the results obtained by the individual classifiers. The model performs better when using dynamic features for classification, compared to relying solely on static API-based features. This is because malware authors inject some unusable APIs to hide static analysis-based detection. Injecting random APIs into

the malicious files increases the sparsity of the feature vector and thus reduces the learning ability, as can be observed in Tables 3 and 4 see the API-static row. Concerning dynamic-based features, the model performs better than static features. This is because the behavioral activities of the malware can be easily captured. The injected features in the PE files may not be executed during execution. Furthermore, many malware attempts to upload API functions during runtime. Such features will be hidden from the static features vector and will appear in the dynamic-based features vector.

Table 3: The training performance of the proposed model compared to other models

	ACC	Recall	Precision	F-M	FPR	FNR
API-dynamic	0.96	0.93	0.92	0.92	0.03	0.03
API-static	0.94	0.89	0.89	0.89	0.04	0.04
The proposed model	0.96	0.96	0.95	0.95	0.02	0.04

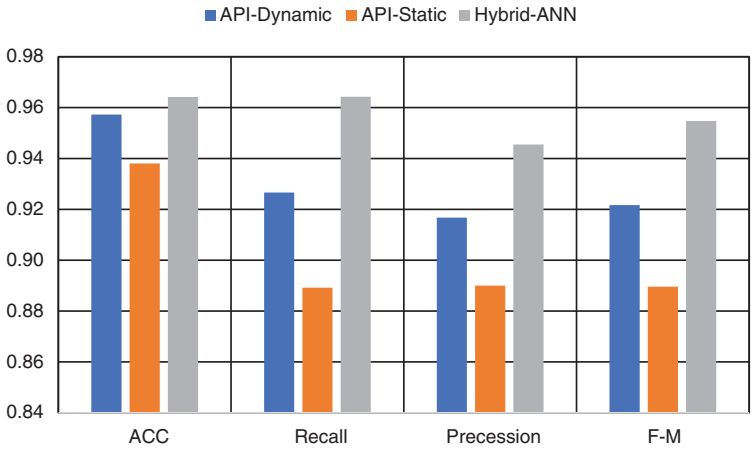

Figure 3: Training accuracy, recall, precision, and F-measure comparison

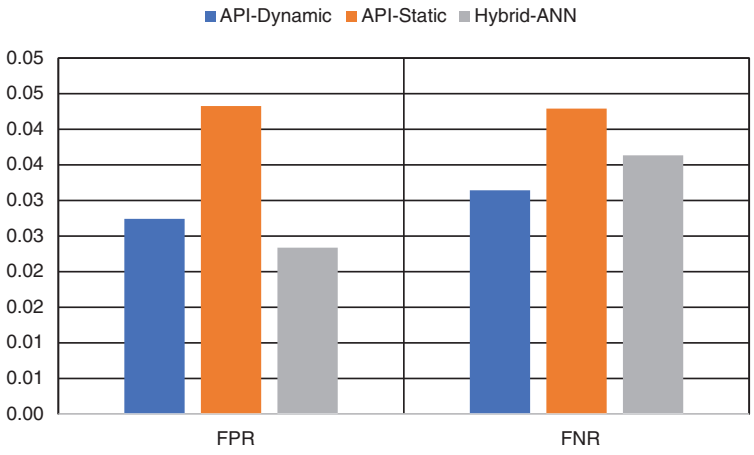

Figure 4: Training errors (FPR and FNR) comparison

Table 4: Compares the proposed model's test performance to that of other models

	ACC	Recall	Precision	F-M	FPR	FNR
API-dynamic	0.94	0.89	0.89	0.89	0.04	0.04
API-static	0.91	0.85	0.83	0.84	0.06	0.07
Finder 2022 [39]	0.86	0.94	0.9	0.92	0.11	0.16
Tran 2017 [40]	0.91	0.91	0.94	0.92	0.07	0.09
Ali 2020 [41]	0.87	0.90	0.92	0.91	0.17	0.10
The proposed API-HMVD	0.95	0.94	0.93	0.93	0.04	0.05

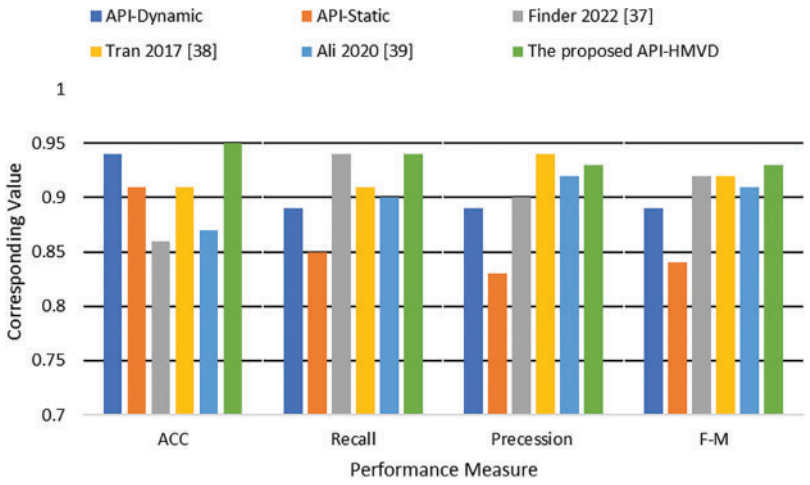

Figure 5: Classification accuracy, recall, precision, and F-measure comparison

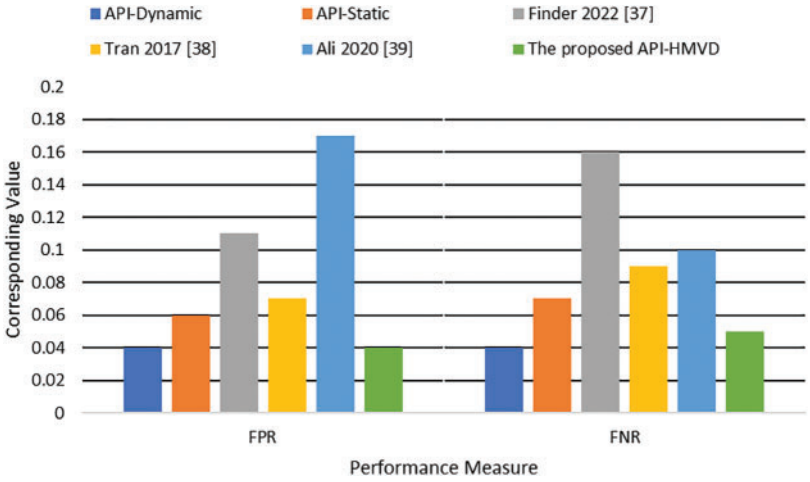

Figure 6: Classification errors comparison

Tables 3 and 4 and Fig. 3 through 6 also illustrate that combining XGBoost with an ANN-based classifier enhances both the training and classification performance. The proposed model achieves an

overall classification performance of 93% in terms of the F-measure compared to 89% achieved by the model that was constructed using dynamic features with individual classifiers using XGBoost. This is because the MLP model uses the output of the XGBoost classifiers to train a model that identifies correlations between the features extracted from both static and dynamic analyses. This integration enables the MLP to identify hidden patterns representing correlations between static and dynamic features. Thus, the proposed hybrid significantly boosts overall performance. As can be seen in Table 4 and Figs. 5 and 6, the proposed model outperformed the related work. It achieved the best accuracy and overall performance (see Fig. 5) with the lowest classification error (see Fig. 6). Although the model proposed in Tran 2017 [40] achieved higher precision, it compromised this precision with a reduced detection rate. Both the false positive and negative rate are high as compared to the proposed model (See Table 4).

As discernible from the data presented in Tables 3 and 4, our proposed model exhibits superior performance when juxtaposed against individual static and dynamic API-based models as well as contemporary state-of-the-art models. The hybrid model under consideration delivered elevated performance measures such as accuracy, recall, precision, and F-measure, during both the training and testing phases. This enhanced performance can be attributed to the amalgamation of static and dynamic analyses that captures a broader spectrum of the software's behavioural dynamics. The achieved accuracy of 95% by the proposed model, as illustrated in Table 4, underscores its proficiency in correctly classifying a substantial majority of the malware specimens. Moreover, the model's precision of 93% signifies that in cases where software is identified as malware, there exists a high probability that the designation is accurate. The recall measure of 94% further emphasizes the model's robustness in identifying positive instances, in this scenario, being malware specimens. This implies that the model was successful in accurately identifying 94% of all malware instances within the dataset. The model's F-measure, a harmonic mean of precision and recall, stands at 93%, signifying that the model maintains an equilibrium between precision and recall.

Within the realm of cybersecurity, both FPR and FNR are deemed paramount. A diminished FPR indicates the model's low propensity to erroneously classify benign software as malicious, thus mitigating unnecessary alerts. Our proposed model exhibits an FPR of a mere 4%. Similarly, a lower FNR implies the model's efficacy in correctly identifying malware, thereby reducing the potentiality of cyber threats. The model under consideration exhibits an FNR of only 5%. When juxtaposed against individual static and dynamic models, our proposed hybrid model evidently surpasses them, as signified by a 1%–3% enhancement across all performance measures. These advancements suggest that the integration of static and dynamic analyses provides a more effective mechanism for malware identification. In relation to state-of-the-art models, our proposed model demonstrated superior performance, thereby substantiating the efficacy of our proposed hybrid approach and the amalgamation of XGBoost and ANN classifiers. In Figs. 3 and 4, the superior training performance of the proposed model is graphically demonstrated. Similarly, Figs. 5 and 6 visually represent the superior classification performance of the proposed model. These graphical representations serve to further underscore the robustness and reliability of our proposed model.

6 Conclusion

This study presents the design and development of an API-based Hybrid Malware Variant Detection Model using Extreme Gradient Boosting and Artificial Neural Network Classifiers. The proposed model seeks to enhance malware variant detection by extracting hybrid API features from both dynamic and static analysis and integrating these features with an ensemble of XGBoost learners

and an MLP classifier. The hybrid features were leveraged to counteract the obfuscation and evasion techniques employed by malware authors for generating malware variants. The XGBoost predictors were created to discover and extract hidden relationships between static and dynamic API-based features. The MLP classifier was developed to learn the hidden patterns that correlate these sets of features. The proposed model's effectiveness is demonstrated through its comparison with non-hybrid models. The principal limitation of the proposed model is its requirement for an in-depth analysis of API features that enhance performance and result in misclassifications, a topic beyond the scope of this study. There exists potential for enhancing detection performance by exploring the ideal number of hidden neurons in the Artificial Neural Network (ANN) model. The utility of the single hidden layer feedforward neural network, as proposed in reference [38] for three-way decisions, can be explored for achieving this aim. Moreover, additional machine learning algorithms like SVM and RF, among others, could also be investigated to enhance detection performance. Future investigations could encompass an in-depth examination of the model's performance in detecting malware employing sophisticated evasion and obfuscation techniques. Malware developer might utilize complex obfuscation and evasion tactics to disguise their genuine behaviors, thereby eluding both static and dynamic analyses. This highlights the significance of integrating a diverse array of features extracted from both static and dynamic analyses into the hybrid model. Further research could delve into creating more resilient models that can effectively identify these advanced obfuscation techniques, thereby contributing to a more secure cyber landscape.

Acknowledgement: The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University for funding work.

Funding Statement: This work was supported by the Deanship of Scientific Research at Northern Border University for funding work through Research Group No. (RG-NBU-2022-1724).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Asma A., Fuad A., Abdulbasit A.; data collection: Sultan M., Abdullah M., Majed A.; analysis and interpretation of results: Shouki A., Asma A., Fuad A., Abdulbasit A.; draft manuscript preparation: Bader A., Asma A., Abdulbasit A., Sultan M., Abdullah M. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: We are committed to promoting transparency and open access to our research. All data and materials used in this study will be made readily available upon request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Kim and S. Cho, "Obfuscated malware detection using deep generative model based on global/local features," *Computer & Security*, vol. 112, pp. 102501, 2022.

[2] A. Kakisim, M. Nar and I. Sogukpinar, "Metamorphic malware identification using engine-specific patterns based on co-opcode graphs," *Computer Standards & Interfaces*, vol. 71, pp. 103443, 2020.

[3] N. Kumar, S. Mukhopadhyay, M. Gupta, A. Handa and S. Shukla, "Malware classification using early-stage behavioral analysis," in *2019 14th Asia Joint Conf. on Information Security (AsiaJCIS)*, Kobe, Japan, pp. 16–23, 2019.

[4] E. Gandotra, D. Bansal and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 2, pp. 56–64, 2014.
[5] J. Zhang, Z. Qin, H. Yin, L. Ou and K. Zhang, "A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding," *Computer & Security*, vol. 84, pp. 376–392, 2019.
[6] D. Gibert, C. Mateu and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, pp. 102526, 2020.
[7] B. Al-rimy, M. Maarof and S. Shaid, "Ransomware threat success factors, taxonomy and countermeasures: A survey and research directions," *Computer & Security*, vol. 74, pp. 144–166, 2018.
[8] D. Ucci, L. Aniello and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computer & Security*, vol. 81, pp. 123–147, 2019.
[9] S. Sibi Chakkaravarthy, D. Sangeetha and V. Vaidehi, "A survey on malware analysis and mitigation techniques," *Computer Science Review*, vol. 32, pp. 1–23, 2019.
[10] S. Baek, "Two-stage hybrid malware detection using deep learning," *Human-Centric Computing and Information Sciences*, vol. 11, pp. 10–22967, 2021.
[11] A. Al-Hashmi, F. Ghaleb, A. Al-Marghilani, M. Saqib, A. Darem *et al.,* "Deep-ensemble and multifaceted behavioral malware variant detection model," *IEEE Access*, vol. 10, pp. 42762–42777, 2022.
[12] G. Xiao, J. Li, Y. Chen and K. Li, "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks," *Journal of Parallel and Distributed Computing*, vol. 141, pp. 49–58, 2020.
[13] U. Urooj, B. Al-rimy, A. Zainal, F. Ghaleb and M. Rassam, "Ransomware detection using the dynamic analysis and machine learning: A survey and research directions," *Applied Sciences*, vol. 12, no. 1, pp. 172, 2021.
[14] D. Angelo, M. Ficco and F. Palmieri, "Association rule-based malware classification using common subsequences of API calls," *Applied Soft Computing*, vol. 105, pp. 107234, 2021.
[15] Y. Ahmed, B. Koçer, S. Huda, B. A. Saleh Al-rimy and M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *Journal of Network and Computer Applications*, vol. 167, pp. 102753, 2020.
[16] E. Amer, I. Zelinka and S. El-Sappagh, "A multi-perspective malware detection approach through a behavioral fusion of API call sequence," *Computer & Security*, vol. 110, pp. 102449, 2021.
[17] K. Bylykbashi, E. Qafzezi, M. Ikeda, K. Matsuo and L. Barolli, "Fuzzy-based driver monitoring system (FDMS): Implementation of two intelligent FDMSs and a testbed for safe driving in VANETs," *Future Generation Computer System*, vol. 105, pp. 665–674, 2020.
[18] X. Liu, Y. Lin, H. Li and J. Zhang, "A novel method for malware detection on ML-based visualization technique," *Computer & Security*, vol. 89, pp. 101682, 2020.
[19] B. Al-Rimy, M. Maarof, M. Alazab, F. Alsolami, S. Shaid *et al.,* "A pseudo feedback-based annotated TF-IDF technique for dynamic crypto-ransomware pre-encryption boundary delineation and features extraction," *IEEE Access*, vol. 8, pp. 140586–140598, 2020.
[20] S. Yang, S. Li, W. Chen and Y. Liu, "A Real-time and adaptive-learning malware detection method based on API-pair graph," *IEEE Access*, vol. 8, pp. 208120–208135, 2020.
[21] J. Suaboot, Z. Tari, A. Mahmood, A. Zomaya and W. Li, "Sub-curve HMM: A malware detection approach based on partial analysis of API call sequences," *Computer & Security*, vol. 92, pp. 101773, 2020.
[22] L. Dhanya, R. Chitra and A. Anusha Bamini, "Performance evaluation of various ensemble classifiers for malware detection," *Materials Today: Proceedings*, vol. 62, pp. 4973–4979, 2022.
[23] L. Xiaofeng, J. Fangshuo, Z. Xiao, Y. Shengwei, S. Jing *et al.,* "ASSCA: API sequence and statistics feature combined architecture for malware detection," *Computer Network*, vol. 157, pp. 99–111, 2019.
[24] J. Grover, N. Prajapati, V. Laxmi and M. Gaur, "Machine learning approach for multiple misbehavior detection in VANET," in *Advances in Computing and Communications*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 644–653, 2011.

[25] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, San Francisco, USA, pp. 785–794, 2016.
[26] J. Palša, N. Ádám, J. Hurtuk, E. Chovancová, B. Madoš *et al.,* "MLMD—A malware-detecting antivirus tool based on the XGBoost machine learning algorithm," *Applied Sciences*, vol. 12, no. 13, pp. 6672, 2022.
[27] R. Elnaggar, L. Servadei, S. Mathur, R. Wille, W. Ecker *et al.,* "Accurate and robust malware detection: Running XGBoost on runtime data from performance counters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 7, pp. 2066–2079, 2022.
[28] S. Saadat and V. Joseph Raymond, "Malware classification using CNN-XGBoost model," in *Artificial Intelligence Techniques for Advanced Computing Applications*. Singapore: Springer Singapore, pp. 191–202, 2021.
[29] B. Al-rimy, M. Maarof and S. Shaid, "Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection," *Future Generation Computer System*, vol. 101, pp. 476–491, 2019.
[30] J. Bai, Q. Shi and S. Mu, "A malware and variant detection method using function call graph isomorphism," *Security and Communication Networks*, vol. 19, pp. 1–12, 2019.
[31] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli *et al.,* "Classification of ransomware families with machine learning based onN-gram of opcodes," *Future Generation Computer System*, vol. 90, pp. 211–221, 2019.
[32] A. Darem, "A novel framework for windows malware detection using a deep learning approach," *Computers, Materials & Continua*, vol. 72, no. 1, pp. 461–479, 2022.
[33] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu *et al.,* "DroidEnsemble: Detecting android malicious applications with an ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.
[34] J. Kang and Y. Won, "A study on variant malware detection techniques using static and dynamic features," *Journal of Information Processing Systems*, vol. 16, no. 4, pp. 882–895, 2020.
[35] K. Rieck, P. Trinius, C. Willems and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.
[36] K. Tian, D. Yao, B. Ryder, G. Tan and G. Peng, "Detection of repackaged android malware with code-heterogeneity features," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 64–77, 2017.
[37] I. Finder, E. Sheetrit and N. Nissim, "Time-interval temporal patterns can beat and explain the malware," *Knowledge-Based Systems*, vol. 241, pp. 108266, 2022.
[38] T. Tran and H. Sato, "NLP-based approaches for malware classification from API sequences," in *Proc. of the 2017 21st Asia Pacific Symp. on Intelligent and Evolutionary Systems (IES)*, Hanoi, Vietnam, pp. 101–105, 2017.
[39] M. Ali, S. Shiaeles, G. Bendiab and B. Ghita, "MALGRA: Machine learning and N-gram malware feature extraction and detection system," *Electronics*, vol. 9, pp. 1777, 2020.
[40] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou and M. Ettaouil, "Multilayer perceptron: Architecture optimization and training," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 1, pp. 26, 2016.
[41] S. Cheng, Y. Wu, Y. Li, F. Yao and F. Min, "TWD-SFNN: Three-way decisions with a single hidden layer feedforward neural network," *Information Sciences*, vol. 579, pp. 15–32, 2021.