IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 17, 2022, accepted April 1, 2022, date of publication April 20, 2022, date of current version April 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3168794

Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model

ASMA A. AL-HASHMI[1], FUAD A. GHALEB[2,3], A. AL-MARGHILANI[4], ABDULSAMAD E. YAHYA[4], SHOUKI A. EBAD[1], MUHAMMAD SAQIB M. S.[1], AND ABDULBASIT A. DAREM[1], (Member, IEEE)

[1]Department of Computer Science, Northern Border University, Arar 91431, Saudi Arabia
[2]School of Computing, University Teknologi Malaysia (UTM), Johor Bahru, Johor 81310, Malaysia
[3]Department of Computer and Electronic Engineering, Sana'a Community College, Sana'a, Yemen
[4]College of Computer Science & Information Technology, Northern Border University, Arar 91431, Saudi Arabia

Corresponding author: Asma A. Al-Hashmi (asma.alhashmi@nbu.edu.sa)

This work was supported by the Deputyship for Research and Innovation, Ministry of Education, Saudi Arabia, under Project 1385.

ABSTRACT Every day, hundreds of thousands of new malware programs are developed and spread worldwide in cyberspace. Most of these malware programs are malware variants such as polymorphic and metamorphic malware, which are created from older versions of malware and able to change their structures and function flows to circumvent security solutions. The accuracy of malware variant detection is a crucial challenge. Many existing malware variant detections use static features extracted from the physical structure of malware file, such as opcodes and function flows. Unfortunately, the static features are subject to obfuscation and code shelling using simple obfuscation techniques. Although a malware variant can change its structure and function flows, it is widely believed that the malware variant cannot hide its malicious behavioral patterns during the runtime. Accordingly, dynamic, or behavioral analysis-based features were suggested by many studies to detect malware variants accurately. However, most of these studies are solely dependent on application-programmable interface calls (or API calls), which is not enough to accurately distinguish between malware and benign due to API-based obfuscation techniques. Therefore, a malware variant detection model that combines different behavioral activities can improve detection accuracy while reducing the false-negative rate. To this end, this study proposed a Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model using Sequential Deep Learning and Extreme Gradient Boosting Techniques. Different behavioral features were extracted from the dynamic analysis environment. Then, a feature extraction algorithm that can automatically extract effective representative patterns has been designed and developed to extract the hidden representative features of the malware variants using a sequential deep learning model. These features have been fed into a developed extreme gradient boosting-based classifier for decision making. Extensive experiments have been carried out to validate the proposed scheme. The results were compared to the other related techniques in the field. The results show that the proposed model is reliable, as it improves the detection rate while reducing the false-negative rate.

INDEX TERMS Malware detection, malware variants, multifaceted behavioral features, deep ensemble learning, sequential deep learning.

I. INTRODUCTION

Malicious software or malware programs have been rapidly growing in recent years. According to the AV-Test Institute, there are more than one billion malware worldwide, and 560,000 malware are detected every day [1]. Most malware developers do not develop malware from scratch [2]. According to [3]–[5], 50% to 80% of the existing malicious software are malware variants. The newly detected malware variants in 2020 have been increased by 74% compared to that identified in 2019. In their 2021 report, Webroot stated that 94% of all malicious executables are polymorphic [6]. Polymorphic malware can frequently change its appearance (e.g., every 20 seconds) in terms of code structure and logic flow, creating massive malware variants [6], [7]. These vast amounts of malware forced researchers to propose many

The associate editor coordinating the review of this manuscript and approving it for publication was Claudio Zunino.

methods to detect and naturalize the malware payload before compromising security.

Existing malware detection solutions can be categorized into two groups namely static and dynamic analysis. It is categorized based on the type of analysis and how their features are extracted [8], [9]. In the static analysis, features are extracted from malware executable files (e.g. .exe and .dll in MS Windows), without the need to execute the malware samples. Examples of static features include strings, imported libraries, and function calls, among many others. Many solutions have been proposed to detect malware variants using static analysis [7], [10]–[15]. Static analysis has been frequently reported for the detection of malware variants [7], [10], [11], [14], [15]. However, static features can be hampered by obfuscation techniques, such as polymorphic and metamorphic malware, that hide the malicious payload and make it indistinguishable [4], [16]. Some obfuscation techniques can prevent feature extraction and hinder the static analysis by dynamically loading the code during the runtime [13]. Therefore, static features are ineffective for malware variants that change their appearance frequently by modifying or hiding their malicious structure, function flow, or rewriting themselves from scratch.

In contrast, the dynamic analysis aims to extract the behavioral features by monitoring the behavior of the executable program during the runtime execution. Examples of behavioral features include API and system calls, log and auditing files, registry access, and network traffic. Because the malware variant is generated from old malware, malware variants usually have similar behavior to the original [2]. Therefore, behavioral analysis is key to accurately detecting malware variants. However, most existing behavioral-based malware detection solutions are based solely on API calls [4], [9], [17]–[21]. Although API calls traces can represent most of the malware variants, API calls alone are not enough to accurately distinguish between malware and benign. This is because most of the malware writers use the same APIs functions that are used for developing benign software. Thus, it becomes difficult to differentiate between malware and benign, depending solely on API calls.

Moreover, many malware writers deliberately inject unnecessary API calls to evade detection. In addition, not all malicious or benign software uses API calls to function; many of them write their codes without the use of the API. In this case, the subject file may be represented by a sparse vector, and thus, it is hard to distinguish the malicious behavior from the legitimate one. We argue that the absence of API traces does not mean that the subject file is benign. Accordingly, API call sequences become ineffective for accurate representation and detection.

Some solutions combined different types of features to detect malicious patterns accurately [4], [5], [22], [23]. However, most of these solutions combine different types of static features [4], [9] or combine static features with API call sequences that are extracted from the dynamic analysis [4], [22]. Although API-based features from the dynamic and static analysis can achieve high detection performance, API alone cannot reflect the malicious behavior of a malware sample. Other dynamic behavior such as file auditing, registry access, and network behavior can further improve the detection accuracy while reducing the false-negative rate. The hypothesis is that each type of behavioral characteristic can tell a part of the maliciousness or goodness of the investigated executable file. However, to the best of our knowledge, no model was found that combines different dynamic behavior to detect malware variants. Therefore, it is important to incorporate different behavioral patterns into the malware variant detection model to improve its performance. Designing a model that combines different behavioral features is challenging due to the overlapping nature of the patterns that may work as noise during constructing the classifier. Therefore, it is essential to effectively extract the representative features that distinguish between benign and malware patterns.

To this end, this study proposed a Multifaceted Deep Ensemble Behavioral-based Malware Variant Detection Scheme using sequential deep learning and the eXtreme Gradient Boosting algorithm (MDEB-MVDS-XGB). The proposed MDEB-MVDS-XGB combines multiple behavioral-based features extracted from dynamic analysis. Different behavioral features were extracted from dynamic analysis, such as API calls, log and file auditing, registry access, and network traffic. The sequential deep learning algorithm was designed and developed to extract the hidden representative malware features automatically. The activation values and weights of the last hidden layer of the trained deep learning model were used to develop an ensemble classifier using the eXtreme Gradient Boosting algorithm. Extensive experiments were carried out to evaluate the proposed model. The results show that the proposed MDEB-MVDS-XGB model can detect unseen malware variants effectively. This study made the following contributions.

1) A Multifaceted and Deep Ensemble Behavioral-based Malware Variant Detection Scheme using sequential deep learning and the eXtreme Gradient Boosting algorithm (MDEB-MVDS-XGB) are designed and developed. Different behavioral features were extracted from dynamic analysis. These features were combined and used for the detection to overcome the malware variant obfuscation techniques.
2) A features extraction algorithm based on a sequential deep learning model is designed and developed for automatic extraction of the hidden malware patterns without human intervention. The weights and activation values of the neurons of the last hidden layer of the trained deep learning model are extracted and used as a new representative feature to train the ensemble model based on the extreme gradient boosting algorithm.
3) The fail-safe security principle is preserved by increasing the classification accuracy by minimizing the false-negative rate. Different ensemble models with single and multiple behavioral features were designed and

developed to evaluate the proposed model. The proposed model was validated and evaluated by conducting extensive experiments.

The rest of this study is organized as follows. The related work is presented in Section 2. A detailed description of the proposed model is explained in Section 3. Section 4 presents the performance analysis, including the description of the dataset, performance measures, and evaluation and validation procedures. Section 5 presents the results and a discussion. It also includes the limitations and future work of this study. This study is concluded in Section 6.

II. RELATED WORK

Malware authors constantly innovate ways to create new malware variants that circumvent security solutions while security analysts and researchers try to improve the security defenses and naturalize such threats. Many obfuscation techniques have been reproduced to create new malware variants that can evade detection. For example, polymorphic malware can modify its appearance in terms of structure and functions flow like the chameleon, which can change its color to disguise itself and hide from predators [7]. Another example is metamorphic malware which can rewrite itself from scratch [10], [12], [24]. Such malware is usually created from previous malware but with new characteristics. There are many solutions proposed to countermeasure malware variants [2], [4], [5], [7], [16], [25], [26]. Most of these solutions are for detection purposes.

Malware variant detection solutions can be grouped according to the type of analysis into two types: static and dynamic. In static analysis, representative features are extracted from the portable executable file (the *exe* files and *dll* files on MS Windows platforms) without executing these files. Static features are extracted from the file that includes strings [13], operation codes (opcodes) [12], dynamic link libraries, API calls [4], [14], [19]–[21], function calls [26], and requested permeations and intended correlations (in Android platforms) [27]. Meanwhile, in dynamic analysis, representative features are extracted by monitoring the behavior of malware during runtime in terms of its interactions with the operating system [17]–[19], [22], [28], [29], file systems, windows registry, and network traffic [15], [30]. Different behavioral features can be extracted, such as system calls, API call sequence, file-related behavior (access, created, modified, or deleted), registry access (creating or modification), and network traffic.

behavior Liu et al [31] proposed a malware detection model using an ensemble shared nearest neighbor (SNN) clustering algorithm. Three types of features were extracted through static analysis: opcode, control flow graph, and import functions which were represented by a grayscale image, directed graph, and term frequency, respectively. Features selection using information gain of the sequence was applied to extract 500 features among all features extracted using 3-gram. Features were combined, and different machine learning was trained for the decision-making Fan et al [32] developed a malware detection based on API call traces. Frequent subgraphs were used to represent the behavior of malware in the same family. The main drawback of this approach is the static features that are used to detect dynamic structure malware. Mahawer and Nagaraju [24] proposed a model for detecting metamorphic malware using a support vector machine with a histogram kernel. Patanaik and Barbhuiya [20] proposed a model using system calls to create a signature to detect malicious obfuscated programs. However, relying solely on interdependent system calls is ineffective to detect malware variants because such features can be evaded easily using simple obfuscation techniques such as API call reordering and garbage API call insertion. Huang et al [27] extracted representative features from a user interface that is associated with the top-level API function to detect stealthy behavior. For example, sending an email must be associated with a user interface to allow the user to create the message and send the button. However, the behavioral models designed based on API correlation with the user interface have many drawbacks. For example, in many automated services of benign programs, an API function does not need to have a corresponding user interface. Therefore, depending on static analysis only makes the solution vulnerable to polymorphic and metamorphic malware types.

Bai et al [26] developed a model that used a function call graph (FCG) to represent the malware variant. The signatures for the FCGs were created and stored in a database. A portable file with a match FCG signature in the database is recognized as malware. The final decision of whether a file is malware or not is based on the graph isomorphism algorithm [33]. The main disadvantage of the signature-based approach is its ineffectiveness in detecting new malware variants. Moreover, the graph isomorphism algorithm can be circumvented by polymorphic malware. Xiao et al [11] proposed a malware variant detection framework based on binary features that were extracted from portable executable file samples using the deep convolutional neural network. The malware binary is represented as an entropy, graph, and features were extracted using the convolutional neural network (CNN). Then, a classifier using the support vector machine (SVM) was trained for the final classification Cui, Xue [16] visualized the opcodes extracted from portable executable files by grayscale images and used CNN to train a model that can detect malware variants. Wang, Gao [13] proposed malware variant detection based on the Ensemble of String and Structural Static Features. Many types of features were extracted, including string, permissions, hardware and software requirements, intents, API calls, opcode, and the function call graph. These features have been grouped into two types string-based and structural-based features. These features were separately used for training. Three machine learning classifiers were used to train the proposed ensemble model, SVM, k-nearest neighbor (KNN), and random forest (RF) algorithm. The result of each classifier is weighted based on the features type. The main drawbacks of these solutions are their dependence on static features, which is

TABLE 1. Related work.

Author	Analysis	Features	Representation Technique	Detection Technique
Xiao et al.[11]	static	Binary	Image + CNN	SVM
Liu et al. [31]	static	opcode + CFG + import functions	grayscale image + directed graph + information gain of the sequence	Ensemble of classifiers based on SNN + RF + voting scheme
Huang et al. [27]	Static	API call traces related to the user interface	Intent correlation with terms frequency	Manual rule-based matching
Xiao et al. [11]	Static	Binary	CNN	SVM
Wang et al. [13]	Static	string features and structural features	***a)*** *String Features (Term Frequency) + structural features (Directed Graph)*	SVM, KNN, and RF
Mahawer and Nagaraju [24]	static	opcode	histogram intersection kernel	SVM
Bai et al. [26]	static	function call	FCG	signature matching using graph isomorphism algorithm
Fan et al. [32]	static	API calls	directed graph	signature-based using sub -graph similarity
Cui et al. [16]	static	opcode	grayscale images	(CNN)
Darem et al. [25]	dynamic	API calls	API call sequence TF-IDF	incremental sequential deep learning
Kang and Won [5]	static and dynamic	byte sequence + API sequences	For opcode grayscale image For API, term frequency	ensemble-based XGBoost
Sun et al. [2]	static and dynamic	suspicious system call + run-time behavior graph	collect runtime binder calls to build directed graph with system calls are edges, and components are vertices	signature-based using graph decoupling and similarity function
Zhang et al. [4]	static and dynamic	opcode + API calls	CNN for opcode and ANN for API calls	SoftMax

ineffective for detecting malware variants due to the simple obfuscation techniques that malware authors can use to hide malicious patterns in the binary code.

Darem *et al.* [25] present an adaptive mode for detecting malware variants based on API calls sequences and incremental deep learning. The API calls sequence were extracted using n-gram and represented using term frequency-inverse document frequency (TF-IDF). The main limitation of this approach is the need for human intervention to label the malware variant to update the model. Han, Xue [28] used API call sequences that were extracted from static and dynamic analysis to develop a malware detection framework. Dynamic and static API call sequences are correlated to construct a hybrid feature vector based on semantics mapping. A potential downside of this framework is that a malware author can maintain a correlation between the static API and the dynamic API by calling the injected static API during runtime. Thus, the correlation is preserved while the malicious program is executed.

Kang and Won [5] combined features extracted from static and dynamic analysis to train an ensemble model for detecting malware variants. Opcode-type features were extracted using static analysis, while API calls-based features were extracted using dynamic analysis. The opcode-based feature was represented as a grayscale image, while the API calls are represented by their term frequency. Random forest, XGBoost, and different deep learning algorithms were used for classification. XGBoost was reported as the best classifier for the combined features. Sun et al [2] proposed a malware variant detection model based on both static and dynamic analysis structured features. The suspicious system call set (SSS) and runtime behavior graph (RBG) were used as behavioral features. The static behavior graph (SBG), which is a subgraph of RBG was used to represent malware static behavior while the system calls were used to represent its dynamic behavior. Although the model generates the signature from malware runtime behavior, the model is signature-based, where the runtime behavior signatures of known attacks are stored for matching. A new malware variant is detected based on the similarity of its RBG and SSS with the existing signature. Zhang et al [4] proposed a hybrid malware variant detection system based on the combination of statistically extracted features with dynamically extracted features. More particularly, the operation code and API calls were used to construct two models using CNN for the opcode-based features and artificial neural network (ANN), the back-propagation neural network (BPNN) for the API calls-based features. The hidden features extracted from the hidden layer of BPNN were combined with the SoftMax features extracted from the SoftMax layer of the CNN model to construct the hybrid feature vector. Then, a SoftMax classifier, which uses the cross-entropy loss, was used to train the malware variant classifier. Although such model has improved the classification accuracy to some extent, there is room for improvement, especially if a single type of behavioral features was used.

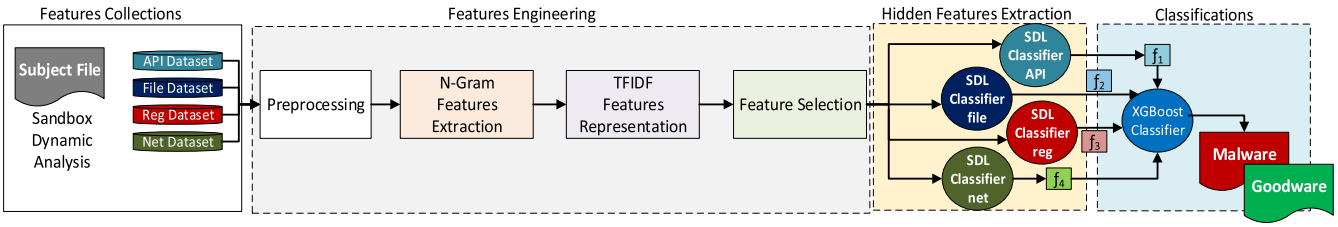

FIGURE 1. Overview of the proposed MDEB-MVDS-SDLXGB model.

In summary, many solutions were proposed to detect malware variants. As shown in Table 1, these solutions were grouped based on the type of analysis into static, dynamic, or hybrid (static and dynamic). Static analysis was frequently reported for malware variant detection. However, static features can be hampered by obfuscation techniques such as polymorphic and metamorphic malware [4], [16]. A polymorphic malware changes its appearance frequently by modifying its structure or flow, while metamorphic rewrites itself from scratch, generating a new malware variant. Some obfuscation techniques can prevent feature extraction and hinder the static analysis by dynamically loading the code during the run time [13]. API call sequences from both dynamic and static analysis were commonly used to represent the malware variants. However, depending on API calls is ineffective for many reasons. First, malware authors usually use the API calls that are used to develop benign software. Thus, it becomes difficult to differentiate between malware and benign depending solely on the API calls. Secondly, the malware author injects unnecessary API calls to hide the malicious patterns into different benign patterns to evade the detection. Thirdly, not all malicious or benign software use API calls to the function. In this case, the subject file may be represented by a sparse vector, and thus, it is hard to distinguish the malicious behavior from the legitimate one.

Many solutions have been suggested to combine different types of features to represent the malware author. However, most of these solutions combine different types of static feature or API calls sequences extracted from dynamic analysis. Although API-based features from the dynamic and static analysis can achieve high detection performance, other dynamic behavior such as file auditing, register access, and network behavior can further improve the detection accuracy while reducing the false-negative rate. Unfortunately, combining different dynamic behavioral features to detect malware variants was not considered. This study proposes a Multifaceted Deep Ensemble Behavioral-based Malware Variant Detection Scheme using sequential deep learning and the eXtreme Gradient Boosting algorithm (MDEB-MVDS-XGB). The MDEB-MVDS-SDLXGB combines multiple behavioral-based features extracted from dynamic analysis. A detailed explanation of the proposed model is provided in the subsequent section.

III. THE PROPOSED MODEL

The proposed MDEB-MVDS-SDLXGB model consists of six main components: raw behavioral data accusation, data preprocessing, features extraction, features representation, features selection, deep multifaceted hidden features extraction, and ensemble-based classification. Figure 1 shows an overview of the proposed model. As can be seen in Figure 1, four types of features were extracted namely the API-, File-, Registry-, and Network-based features. After the preprocessing, the extraction of features sequences using n-gram, the representation using TF/IDF, and the important features are selected, four types of hidden features are extracted using sequential deep learning. Four sets of hidden features were extracted denoted by f1, f2, f3, and f4 for API-, File-, Registry-, and Network-based, respectively. The hidden features are merged and used to train a classifier for decision-making using the XGBoost algorithm. The detailed description of each component in Figure 1 is provided in the following subsections.

A. BEHAVIORAL DATA EXTRACTION PHASE

In this step, different types of behavioral features are collected about the subject executable file, such as network traffic, file access (read, write, create, or delete), registry access, and system call sequence (or API call traces). These features are extracted during the runtime by submitting the subject file to a dynamic analysis environment to extract behavioral features automatically. When the subject file is executed (usually in an isolated environment such as Windows Sandbox) different behavioral data can be captured.

B. DATA PREPROCESSING

Data preprocessing plays an essential role in machine learning-based models, especially in malware detection, where the malware can compromise the system in case of misclassification. Data preprocessing helps to eliminate the effect of unnecessary content that contributes to classification to maximize accuracy. Most of the data collected in the previous step are unstructured text data. It may be dumped from different types of acquisition tools with different formats and structures. Such data is usually contained redundant and unnecessary features, has missing values, and contains noise such as symbols, XML or HTML tags, punctuations,

and stop words. Such unnecessary content or inconsistencies should be removed because it can produce misleading results. Therefore, in the preprocessing step, the data is cleaned by removing unnecessary content to help the machine learning algorithm find a correct and representative malware pattern that is distinguishable from the benign pattern. After special characters, stop words, punctuation, and unnecessary symbols are removed, the data are converted into lowercase characters for consistency.

C. FEATURES EXTRACTION

Feature extraction aims to create new informative features sets in which the malware variant can be represented better than using the original features. In this step, a technique called n-gram is used to extract more features from each sample by concatenating the subsequent words (also called terms) in the group of n subsequent words that occurred in the sample. In n-gram, each subsequent word starting from 1 to n is used as a unique feature. For example, in one-gram, every single word is considered one feature, while in two-gram, every two subsequent words are considered one feature. N-gram has been commonly used in text data mining applications. N-gram is also used by many malware studies [8], [10], [34] to extract features from API sequence, strings, and file auditing. The higher is the n value, the more features that can be extracted. However, too many features lead to high dimensionality, noises, and overfitting problems. In this study, the n is set to a range of one and two so every two subsequent features are combined to represent and then added to the extracted single features [25], [38], [39]. The reason for selecting n-gram is that a single feature in malware is not harmful compared to feature sequence which is more representative [38]. The use of a short sequence consisting of one or two features sequence is found to be better than using a three-gram in terms of performance as reported in [25], [39].

D. FEATURES REPRESENTATION

In this step, each sample (malware or benign) is represented by sets of unique terms (vocabulary). These unique terms were used to create a corpus. Then, all unique words in the corpus were used as feature vectors that will be used to generate the representative feature of each sample. The aim is to transform the text into numerical values so that machine learning algorithms can deal with it. The Term Frequency-Inverse Document Frequency (TF-IDF) is used to represent each unique term in the sample features. Thus, for each sample, every term is the feature vector is represented by its TF/IDF equivalent value. The TF-IDF is calculated as in the following formula.

$$tf\ (x) = \frac{Number\ of\ times\ x\ occur\ in\ a\ sample}{No.\ of\ terms\ in\ the\ sample} \tag{1}$$

$$df\ (x) = Number\ of\ documents\ that\ has\ x \tag{2}$$

$$x_{tf-idf} = tf\ (x) \times \ln\left(\frac{N}{df\ (x)+1}\right) \tag{3}$$

where x is the term, $tf\ (x)$ is the term frequency, $df\ (x)$ is the document frequency where x has occurred, N denotes the number of samples in the given dataset.

The TF-IDF can make general-purpose terms and specific terms distinguishable. For example, API terms that are frequently called by many samples are given scores lower than API calls that are specific for a particular sample or class. The general-purpose terms, which frequently occur in many samples from different classes, do not add any information about the target class. Therefore, the term is ranked when it is frequently used by a class and not frequently used by the other classes.

E. FEATURES SELECTION

One of the challenges of classifying malware is the high dimensionality of the features extracted from the behavioral data of the malware. The large number of features that can be extracted by the n-gram technique can lead to either an overfitting or an underfitting problem. Redundant features are a common problem in API calls due to the use of the same API functions for different functions in the program by both benign and malware authors. In addition, the correlated features make the gradient descent algorithm in machine learning-based models oscillate and slow the convergence. Moreover, the correlation between the features and the variance of the loss is high even with a small average value. Thus, the learner is misled and converges in inaccurate coefficients. Furthermore, some features are very specific to a particular sample, and others are very general. Both types of features make noises that affect the accuracy of the detection. Therefore, feature selection is an important step in eliminating redundant features and improving detection performance. The eXtreme Gradient Boosting Algorithm (XGBoost) was utilized to select the important features in this study. XGBoost can estimate the importance of the features during training by measuring how each feature was useful in the construction of the boosted decision trees. The feature is ranked based on the number of the split points that contribute to the decision tree. This technique is called the Gini impurity or Gini index. In the Gini index, a feature is more important than the others if its GI_f is lower than the other compared features. Gini index GI_f for a feature f can be calculated as follows.

$$GI_f = 1 - \sum_{j=1}^{n} p_j^2 \tag{4}$$

$$p_{j(f\geq t)} = \frac{\left|F_{(j)} : f \geq t\right|}{|F : f \geq t|} \tag{5}$$

where p denotes to the proportion of samples of each class in a split at point t, n denotes the number of the classes in each split, F denotes the set of all values in feature f that are in the split $f \geq t$, and $F_{(j)}$ denotes the set of all values belonging to the class j that is in the split $f \geq t$. For example, if we have two classes and a feature f is split at point t, then the

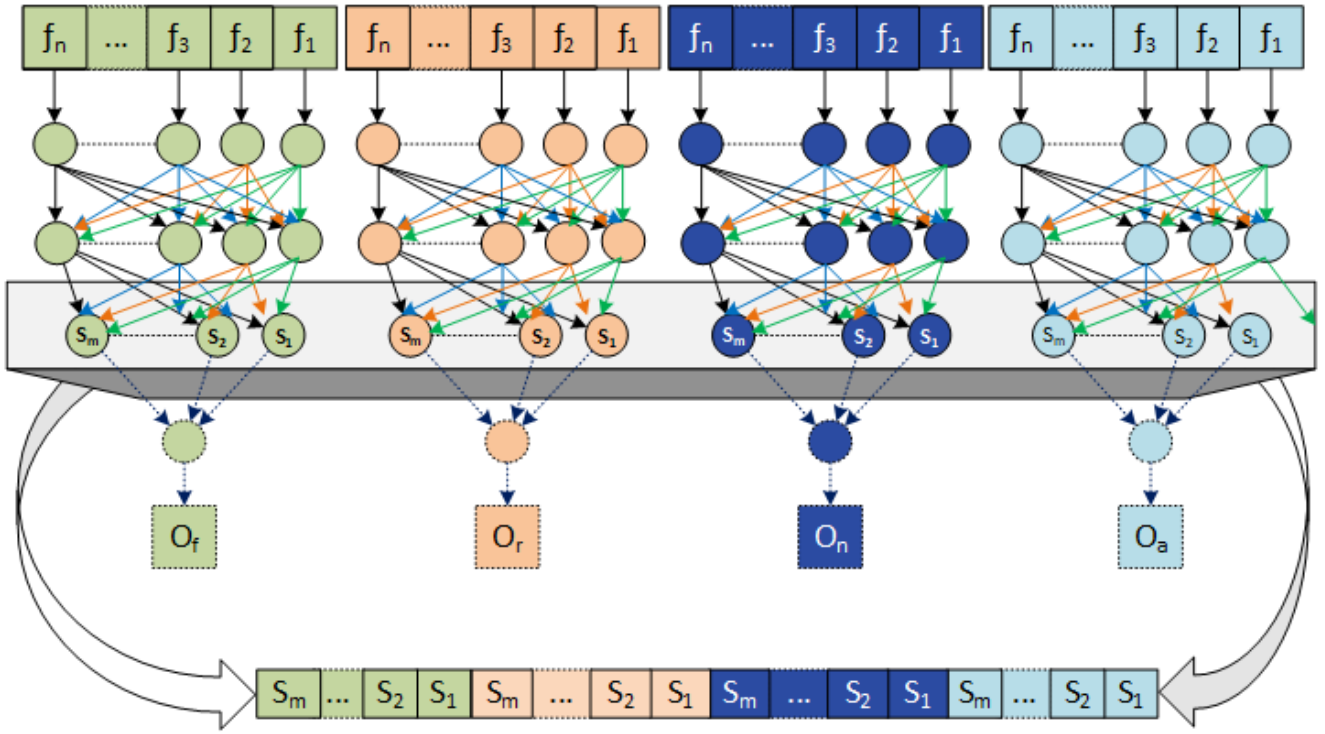

FIGURE 2. Shows multifaceted sequential deep learning hidden features.

feature importance (*fi*) is calculated as follows.

$$fi = 1 - \sum_{j=1}^{2} \left(\frac{|F_{(j)}|}{|F|} \right)^2 \tag{6}$$

F. DEEP MULTIFACETED HIDDEN FEATURES EXTRACTION

This phase aims to extract the hidden features representing the subject concerning its different behavior in terms of network, file access, API calls, and registry access.

These features are extracted from the last layer of the trained deep neural sequential model. They are the activation values of the last hidden layer with the weights of each neuron of this layer in the deep learning model. In this phase, four feature vectors are extracted, each representing different malware behavior for each subject. These features are used to learn hidden behavioral patterns. Figure 2 illustrates the multifaceted feature vector extracted from the hidden layer of the trained sequential deep learning model. Two activities were conducted to develop these multifaceted features vectors, one for training and the other for online operation or testing. In the training phase, the datasets containing features representing each type of behavior have been split into two subsets, 60% of the data is for the training, and the rest is for testing. In the training phase, sequential deep learning (SDL) is constructed, trained, and validated. The constructed sequential model consists of five dense layers: one input layer consists of the number of selected features, three hidden layers with size 64, 32, and 16 neurons in each hidden layer, respectively, and one output layer consists of one neuron to evaluate the learning performance. The activation function used in the input and hidden layer is the ReLu function while the sigmoid function is used in the output layer for decision making. To minimize the error and update the weights, the Adam optimizer, which is an extension of the stochastic gradient descent technique, was employed. It's a form of adaptive gradient that uses an adaptive moment estimation technique to estimate a dynamic learning rate. The model is trained, and then the activation values of the last layer are extracted and used as input features for the XGBoost classifier.

As can be seen in Figure 2, the important features that were selected in the feature selection phase which is donated by $f_1, f_2, \ldots f_n$ where n is the number of selected features from the four extracted features sets. The selected features are used as input to four SDL classifiers and the outputs are the hidden features $S_1, S_2, \ldots S_m$ where m is the number of the neuorns in the last hidden layer of the SDLs classifiers. Figure 3 represents the methodology of the constructed multifaceted sequential deep learning model.

Let F is the set of input features selected using the features importance and f is an element in F, L is the set of all layers in the deep learning model and l is an element in L, $a_i^{(l)}$ is the activation value of the node i in level l and $w_{ij}^{(l)}$ is the weight of input node i for node j in level l and g is the activation function. The activation score $S_i^{(l)}$ of the last hidden layer of the train and the deep sequential model can be calculated as follows.

$$a_i^{(0)} = g\left(\sum_{j=1}^{n} w_{ij}^{(1)} f_i^{(0)} + b^{(0)} \right) \tag{7}$$

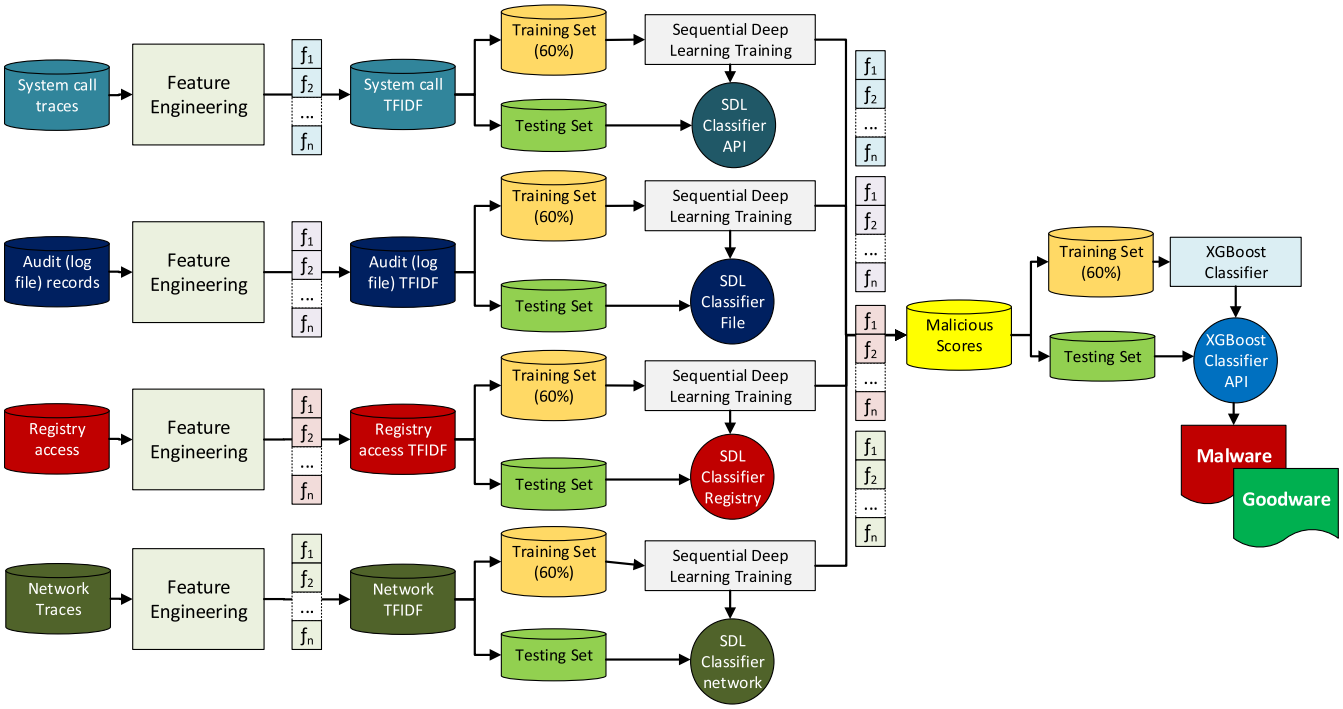

FIGURE 3. The methodology of the proposed MDEB-MVDS-SDLXGB model.

$$a_i^{(l)} = g\left(\sum_{j=1}^{k} w_{ij}^{(l)} a_i^{(l-1)} + b^{(l-2)}\right) \tag{8}$$

$$S_i^{(l)} = w_{ij}^{(l)} \sum_{j=1}^{m} w_{ij}^{(l-1)} a_i^{(l)} + b^{(l-1)} \tag{9}$$

where m is the number of nodes in the last hidden layer, k is the number of nodes in the hidden layer before the last, and n is the number of input features. The function $g()$ is the activation function. In this study, the ReLU function was used as the activation function of all nodes in the hidden layers.

G. ENSEMBLE BASED CLASSIFICATION

This phase aims to make the final decision about whether it is malware or benign. In this phase, the feature vector obtained from the previous phase is used as input features for the Extreme Gradient Boosting algorithm for decision making. The XGBoosting algorithm has been used to train a model based on the scores made by the Multifaceted Sequential Deep Learning model. The gradient boosting method used in the XGBoost algorithm incrementally creates new decision trees that consider the error made by the previous decision trees. The gradient descent algorithm is used to reduce the error when a new tree is added. XGBoost uses Taylor expansion to calculate the cost function. The trees are gradually built and added to the ensemble. A regularization term is used to prevent the tree from being complex. Figure 3 shows the structure of the proposed MDEB-MVDS-SDLXGB model. The hidden features are extracted from the trained model and used for classification in the online operation.

H. ONLINE OPERATION

The subject file is submitted to the sandbox environment for dynamic analysis. The subject is executed, and its different behavior in terms of API calls, network traffic, file access, and registry access is logged. The raw text data collected are preprocessed using the aforementioned data preprocessing steps. Then, more features are extracted using the n-gram technique. Then using the trained TF-IDF vectorization method, the representative numerical features are created. Using the trained feature selection model, only important features are selected and used as input to the sequential deep learning model.

The sequential deep learning model gives a score between zero and one. For each subject, there are four scores to represent its behavior in terms of API calls, network traffic, file access, and registry access. Finally, these scores are used as input features to the trained XGBoost model to decide whether the subject file is malware or benign. Algorithm 1 and Figure 4 summarize the online operation of the proposed MDEB-MVDS-XGB model.

IV. PERFORMANCE EVALUATION

This section describes the evaluation process of the proposed model. It also describes the setup of the experimental environment, the used dataset, and the performance measures.

A. EXPERIMENTAL SETUP

The four types mentioned above of behavioral features were extracted during runtime from a dynamic analysis environment. The dynamic malware analysis environment is

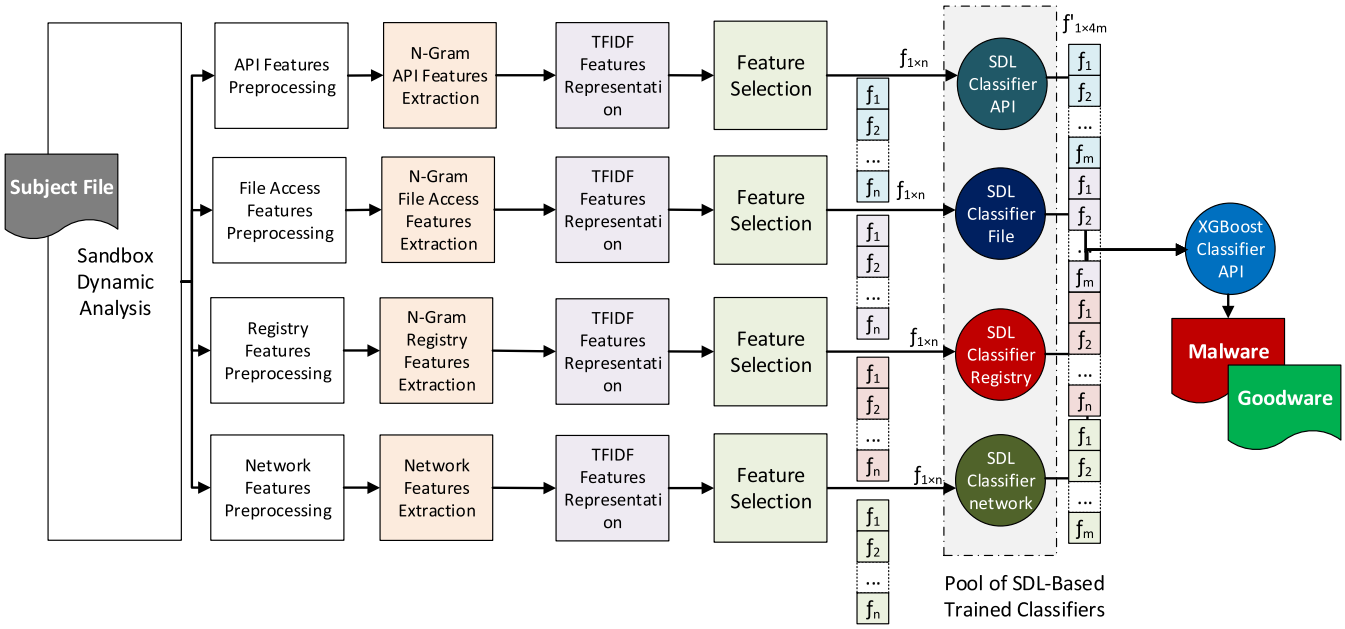

FIGURE 4. The online operation of the proposed MDEB-MVDS-SDLXGB model.

constructed in an isolated virtual environment with a host computer CPU Intel (R) Core i7 @ 3.20 GH, and the RAM is 16.0 GB. Cuckoo sandbox tools were used with the virtual box to build an isolated and controlled virtual environment for malware investigation. The host operating system is Linux Ubuntu 18.04 and Windows 7 as the guest operating system. Windows 7 was used as a victim machine. Several researchers commonly use sandboxes to extract behavioral features [8], [34], [35].

The sandbox was set up following the instructions presented in [29]. The guest Windows 7 operating system was installed in the virtual machine, and a configured and clean slate screenshot was made. Many applications have been installed, some dummy files and folders have been generated, and an internet connection has been enabled to make the guest operating system more realistic to the evasive malware sample. The cuckoo agent on the guest operating system runs the provided binary files and hooks their API calls, as well as logs the network traffic, file access activity, and register access behavior. The cuckoo agent on the virtual machine collects these behavioral features of the submitted file and sends them back to the host machine. The virtual machine is then restarted with the initial clean slate restored, allowing the new analysis to begin with a fresh copy of the guest operating system. Finally, the API call sequences were extracted from the cuckoo agent reports folder using Python programming packages.

B. DATASET DESCRIPTION

The malware binary files used in this study were downloaded from the public repository VX Heaven.[1] Previous malware detection researchers have already used this dataset [4], [8], [10], [21], [34], [36]. There are numerous distinct types of malware families in the malware dataset, including trojans, adware, backdoors, ransomware, viruses, and worms. The Vxheaven collection yielded a total of 19076 malware samples, which were chosen at random. The benign or benign binary files were obtained from a freshly installed Windows operating system. A total of 3994 benign executable and dynamic link libraries were collected. As a result, the dataset utilized in this study has 23070 samples, with 19076 malware samples and 3994 benign ones.

C. PERFORMANCE MEASURES

Multiclass performance metrics are commonly used for measuring and evaluating the quality of malware detection [37]. The same metrics have also been used to validate the proposed model in this study. These metrics include detection accuracy, detection rate (or recall), false-positive rate, precision, and F-measure. However, these performance measures are not enough for the evaluation because they do not consider the fail-safe security principle. We argue that malware detection should consider the false-negative rate more than the false positive rate. A false-positive leads to more investigation and analysis (increase human intervention), while a false-negative leads to compromise the security (the fail-safe principle is violated).

This study investigated the models based on the above performance measures, including the false-negative rate. Consequently, five main performance evaluation metrics were used to evaluate the effectiveness of the proposed model, namely, detection accuracy (ACC), false-positive rate (FPR), detection rate (or recall) (DR), and F measures (F1). The detection accuracy (ACC) is the percentage of the benign

[1] https://www.vxheaven.org

Algorithm 1. The Proposed MDEB-MVDS-XGB Mode

Input: $suspicious_{file}$, $TFIDF_{Transformer}$, S *the set of the trained deep learning model, and* $C_{XGBoosting}$

Output: $file_{class}$

Start

1: $suspicious_{file} \xrightarrow{submit} analysis_{envirnoment}$

2: $analysis_{envirnoment} \xrightarrow{Send} raw_{features}$

3: $raw_{features} \xrightarrow{preprocessing} \forall fs \in Fs : Fs = \{fs1, fs2, fs3, fs4\}$

4: $Fs \xrightarrow{N-Gram(features\ extraction)} Fn : Fn = \{fn1, fn2, fn3, fn4\}$

5: *Calculate* $TF - IDF$*using* $TFIDF_{Transformer}$

$\forall fi \in Fn using x_{tf-idf} = tf(x) \times \ln\left(\frac{N}{df(x)+1}\right) \xrightarrow{append} F$

5: $\forall fi \in Fn$ *Calculate* $feature_{importance}$ $fi = 1 - \sum_{j=1}^{2}\left(\frac{|F_{(j)}|}{|F|}\right)^2 \xrightarrow{Store} F$

6: $F = get\_topn\_features\_score(F)$

&: $\forall S\ j\ in\ S\ do$

7: $\forall$ *nueroninthe last layer* l *in* S *find the hidden features*

8: *Extract the activation value* $a_i^{(l)} = g\left(\sum_{j=1}^{k} w_{ij}^{(l)} a_i^{(l-1)} + b^{(l-2)}\right)$

9: *Compute the hidden feature* $f'^{(l)}_i = w_{ij}^{(l)} \sum_{j=1}^{m} w_{ij}^{(l-1)} a_i^{(l)} + b^{(l-1)} \xrightarrow{append} F'_j$

$F'_j \xrightarrow{append} F'$

10: $C_{XGBoosting}(F') \xrightarrow{classify} file_{class}$

End

samples correctly classified to all the classified samples. The detection rate (DR), also called recall, is the fraction of malware samples that are correctly classified. The false-positive rate (FPR) is the percentage of the incorrectly classified instances as malware samples. The false-negative rate (FNR) is the percentage of the instances that are incorrectly classified as benign samples. F-measures (F1) is the harmonic mean and calculated as in Equation (10), where the TP is the number of malware samples that are correctly classified, FP number of benign samples that are wrongly classified, and FN number of malware samples that are wrongly classified.

$$F_1 = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{10}$$

D. PERFORMANCE EVALUATION

The MDEB-MVDS-SDLXGB model proposed in Figure 2 has been evaluated by comparing its performance with the other five designed models as follows. MDEB-MVDS-SDLXGB is compared with the Multifaceted and Deep Ensemble Behavioral-Based Malware Variant Detection Scheme using Sequential Deep Learning Technique with Majority Voting Scheme (MDEB-MVDS-SDLMV). The Majority Voting Scheme has replaced the XGBoosting technique of the proposed model MDEB-MVDS-SDLXGB in the MDEB-MVDS-SDLMV. Meanwhile, in the Multifaceted Behavioral-Based Malware Variant Detection Scheme using Sequential Deep Learning (MB-MVDS-SDL), all the features from different domains have been combined in one feature vector (See Figure 5). The API calls features have been combined with the features extracted from registry access, file access, and network traffic. Then, sequential deep learning with four layers was trained for the classification. Like the MB-MVDS-SDL, the three other tested models were designed, but each model was trained using one of the following machine learning techniques, extreme gradient boosting for the MB-MVDS-XGB model, SVM is used for the MB-MVDS-SVM model, and random forest algorithm was used for the MB-MVDS-RF model.

V. RESULTS ANALYSIS AND DISCUSSION

Figure 6(a) and Table 2 show a comparison between the performance of the proposed MDEB-MVDS-SDLXGB with the five designed models. As can be seen in Figure 6(a), the proposed MDEB-MVDS-SDLXGB outperforms all the other designed models. It achieved 99.23% accuracy, which is better than the performance achieved by the other designed models. For example, in terms of accuracy, MDEB-MVDS-SDLXGB achieved 0.8%, 0.5%, 1.2%, and 1.6% better than the other designed models MDEB-MVDS-SDLMV, MB-MVDS-SDL, MB-MVDS-SVM, and MB-MVDS-RF, respectively. Similarly, in terms of the recall, the proposed model MDEB-MVDS-SDLXGB achieved the best true positive rate compared to the other tested model, while the SVM-based model achieved the lowest true positive rate. In terms of precision, the majority voting-based ensemble MDEB-MVDS-SDLMV achieved the best precision, followed by the proposed MDEB-MVDS-SDLXGB model. Although the model with the majority voting scheme, MDEB-MVDS-SDLMV, achieved the best precision among all the tested models, such achievement is not praised in security and malware detection, which violate the fail-safe principle. When the precision is higher than the recall, that is an indication of a higher false-negative rate, which means increasing undetected malware, which makes the target vulnerable. Therefore, recall is more important than precision, and thus the MDEB-MVDS-SDLXGB model wins in this case. The results in terms of the F-measure confirm how the MDEB-MVDS-SDLXGB model has the better trade-off of precision and recall. It achieves 99.48% F-measure, which is better than the other designed models MDEB-MVDS-SDLMV, MB-MVDS-SDL, MB-MVDS-SVM, and MB-MVDS-RF by 0.51%, 0.33%, 0.7%, 1.1%, and 2.9%, respectively. Overall, the proposed MDEB-MVDS-SDLXGB achieved the best performance, followed by the MB-MVDS-SDL model. The MDEB-MVDS-SDLMV model has a trade-off recall by precision; because of this, its overall performance is lower than MB-MVDS-SDL. Although the combined features with the random forest-based model MB-MVDS-RF works well with the high definitional data, its performance archives the worst performance among studies models. There are two interpretations of this behavior. The

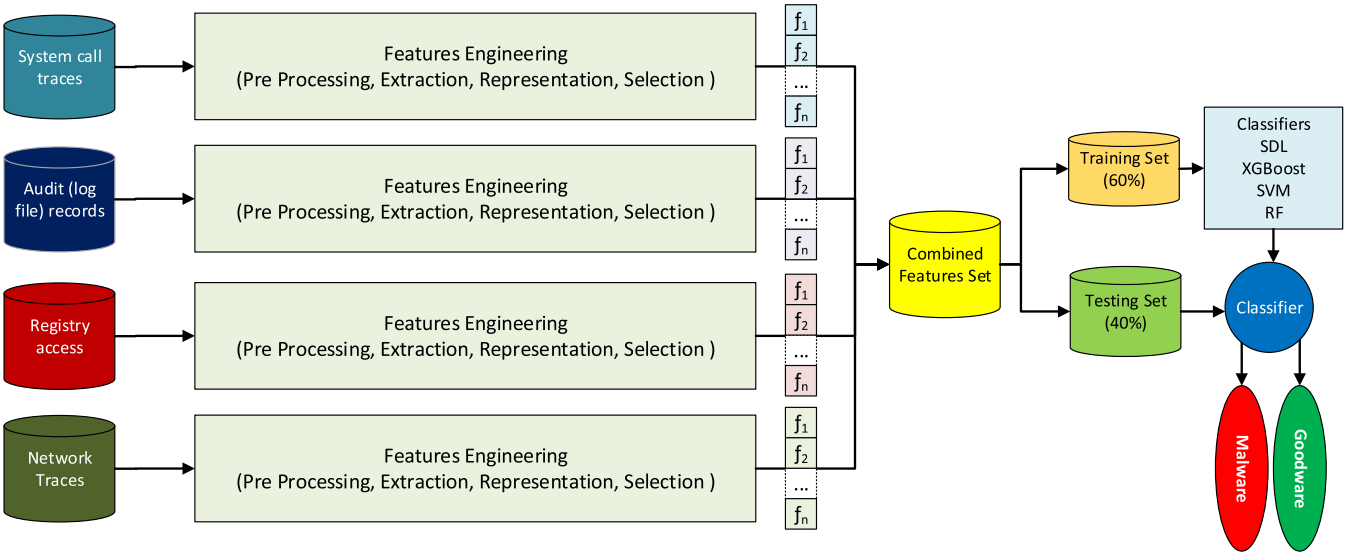

FIGURE 5. Combined behavioral features vector.

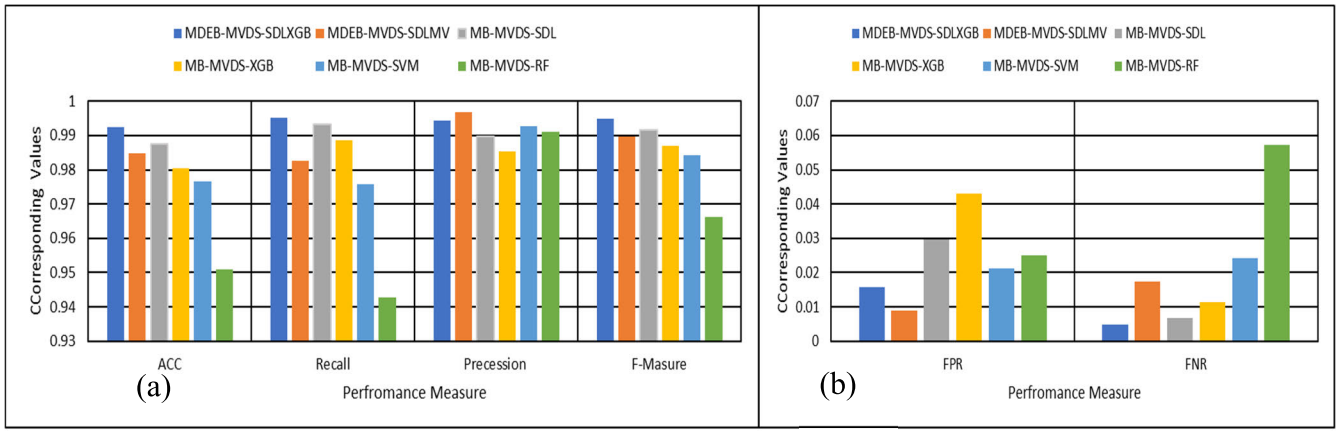

FIGURE 6. Overall performance comparison between the proposed multifaceted deep ensemble behavioral-based malware variant detection scheme using sequential deep learning technique with XGBoost (MDEB-MVDS-SDLXGB), and with the majority voting scheme (MDEB-MVDS-SDLMV), and multifaceted behavioral-based malware variant detection scheme using sequential deep learning (MB-MVDS-SDL), using XGBoost (MB-MVDS-XGB), using support vector machine (MB-MVDS-SVM), and using random forest (MB-MVDS-RF) in terms of (a) accuracy, recall, precision, and F-measure (b) FPR and FNR.

TABLE 2. Performance comparison between the proposed model and those evaluated by others.

	ACC	Recall	Precision	F-Measure
MDEB-MVDS-SDLXGB	**0.9923**	**0.9952**	0.9944	**0.9948**
MDEB-MVDS-SDLMV	0.9848	0.9827	**0.9969**	0.9897
MB-MVDS-SDL	0.9874	0.9933	0.9897	0.9915
MB-MVDS-XGB	0.9805	0.9887	0.9853	0.9870
MB-MVDS-SVM	0.9766	0.9758	0.9926	0.9841
MB-MVDS-RF	0.9510	0.9428	0.9910	0.9663

first is the use of majority voting for decision-making, and the second is the sparsity of the data.

Figure (6) and Table 3 present the performance in terms of FPR and FNR. The lowest FPR has been achieved by MDEB-MVDS-SDLMV, which achieved a 0.9% false-positive rate, followed by the proposed MDEB-MVDS-SDLXGB model, which archives a 1.56% false-positive rate. The worst false positive rate has been achieved by MB-MVDS-XGB where the features were combined and the XGBoost algorithm was used for classification. Although reducing the false positive rate is important, it is not critical for malware detection like reducing the false-negative rate. Reducing FNR is a critical security requirement in malware detection because it may lead to successful attacks. As can be seen in Figure 7 and Table 2, the proposed MB-MVDS-XGB model archives the best reduction in terms of FNR followed by the combined feature vector with the sequential deep learning MB-MVDS-SDL achieve a 0.67% false-negative rate. However, the MB-MVDS-SDL model has a trade-off of the FNR by the FPR, as can be observed in Figure 3.

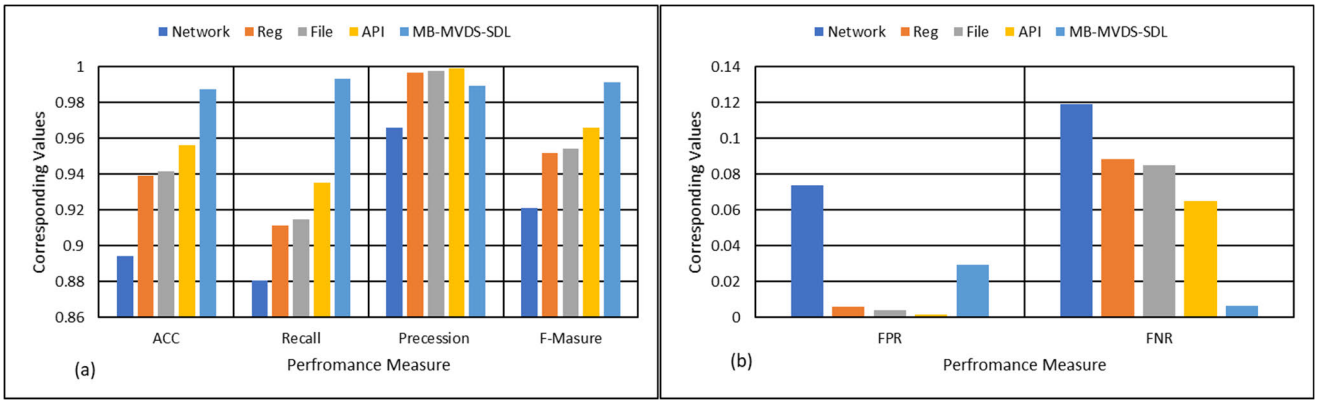

FIGURE 7. Performance measures using the sequential deep learning technique.

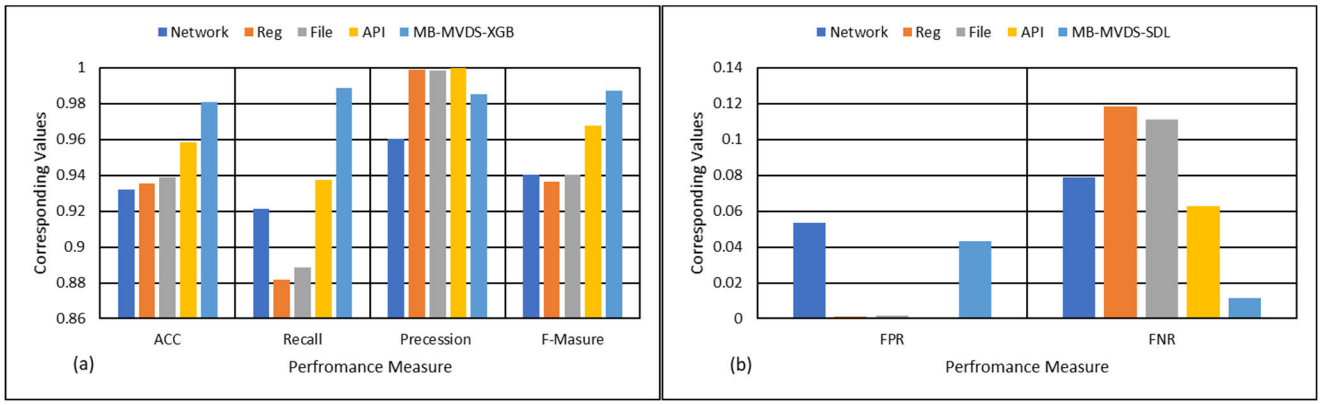

FIGURE 8. The performance measures using extreme gradient boosting algorithm.

Figures 7 (a) and (b) illustrate the performance of the Sequential Deep Learning Classification for four study types of behavioral features. Figure 7 (a) displays the accuracy, recall, precision, and recall, while Figure 8 (b) displays the FPR and FNR. As shown in Figure 7 (a), the API call sequence can effectively represent most malware variance. However, API call sequence-type features create relatively high FPR. A combined features vector with sequential deep learning performs better than a single type of behavioral feature. The results in Figures 7 (a) and (b) show that each type of behavior contributes to creating a more distinctive malware variant. It shows how the false-negative rate has been reduced using the combined behavioral vector compared to the FNR of the individual behavioral vector.

Figures 8 (a) and (b) show the performance of the trained models using the Extreme Gradient Boosting Algorithm. Figure 8 (a) presents the results in terms of accuracy, recall, precision, and recall, while Figure 8 (b) shows the FPR and FNR results. As can be seen in Figure 8 (a), the model trained using the combined features archives the best accuracy, detection rate (recall), F-Measure, among others, while the model designed based on the API call sequence features archives the best performance in terms of precision and FPR. However, the API call sequence type features create a relatively high false-negative rate FNR = 6.2%, which is the main drawback of this model.

TABLE 3. Results in terms of FPR and FNR.

	FPR	FNR
MDEB-MVDS-SDLXGB	0.0156	**0.0048**
MDEB-MVDS-SDLMV	**0.0090**	0.0173
MB-MVDS-SDL	0.0297	0.0067
MB-MVDS-XGB	0.0432	0.0113
MB-MVDS-SVM	0.0213	0.0242
MB-MVDS-RF	0.0250	0.0572

Figures 9 (a) and (b) illustrate the performance of the trained models using the SVM technique. Figure 9 (a) shows the performance in terms of accuracy, recall, precision, and recall, while Figure 9 (b) shows the FPR and FNR results. Similar to the XGBoost model in Figure 9, the model trained using the combined features archives the best accuracy, detection rate (recall), F-Measure compared with the other studied models. Meanwhile, the model designed based on the API call sequence features archives the best performance in terms

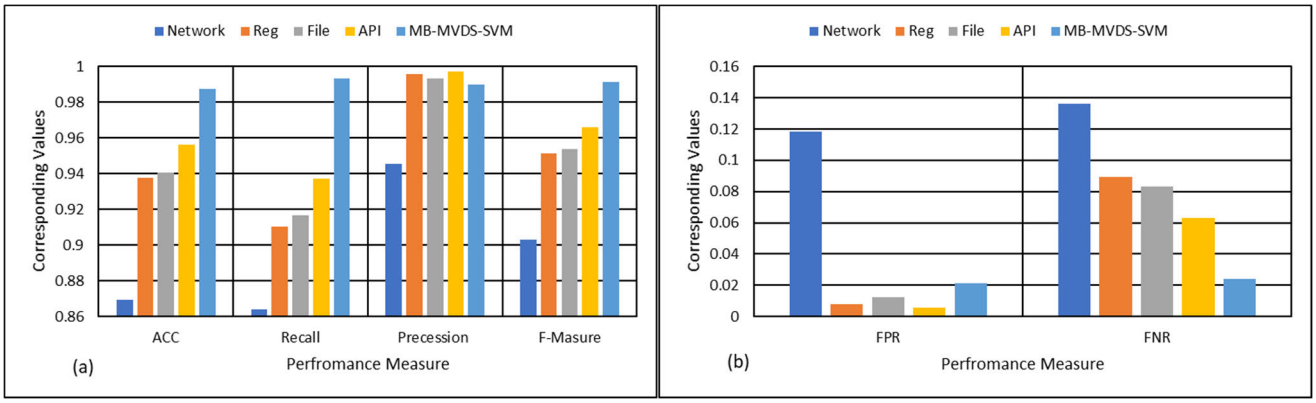

FIGURE 9. The performance measures using the support vector machine technique.

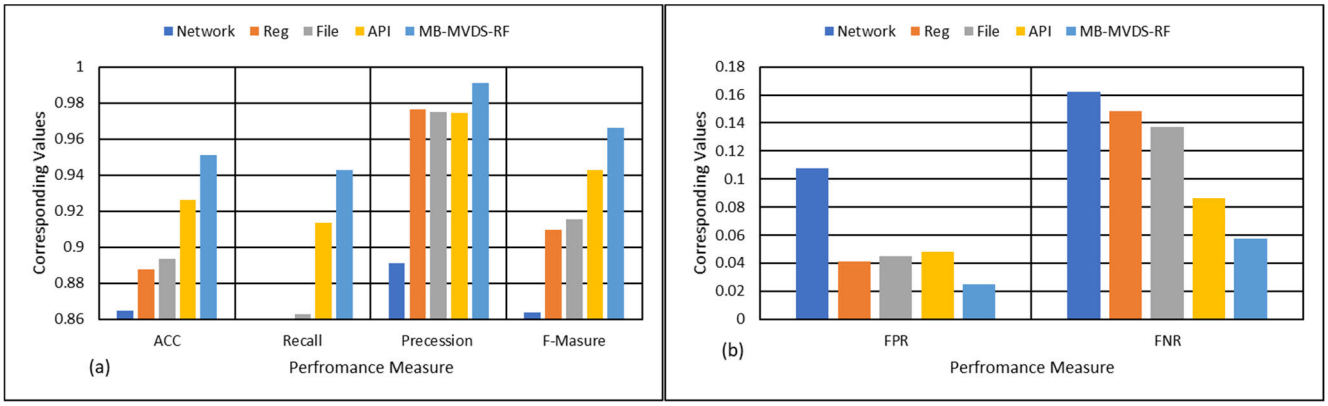

FIGURE 10. Performance measures using the random forest technique.

of precision and FPR. However, this model creates a relatively high false-positive rate FPR = 3%, which is the main drawback of this model.

Figures 10 (a) and (b) show the performance of the trained models using the Random Forest Algorithm. Figure 10 (a) shows the performance in terms of accuracy, recall, precision, and recall, while Figure 10 (b) displays the FPR and FNR results. It can be observed that the RF-based model trained using the combined features archives the best performance compared with the other studied models. However, this model generates a high false-positive rate FPR = 2% with FNR = 5.9%, which is the main problem of this model.

From Figures 7, 8, 9, and 10, we can conclude that the models designed using the combined features sets achieve better accuracy than those were designed using individual features. The better achievement is because the behavior of malware variants can be better represented by considering many types of behavior. When different behavioral features are considered, the model can accurately distinguish between malicious and non-malicious behavior. In most of the cases studied, API-based features can well represent malware variants. However, high false alarms are observed when a single type of behavioral feature is used. Combined features are outstanding in terms of reducing the FNR and FPR while achieving a high detection rate (recall). Moreover, a model designed with sequential deep learning achieved the best reduction of false-negative rates with high detection accuracy. XGBoost algorithm achieved a low false-negative rate while RF suffered from high FNR. Meanwhile, SVM achieves the best trade-off between precision and recall; however, both FPR and FNR are relatively higher than those of the proposed MDEB-MVDS-SDLXGB model.

To have insights into how the proposed MDEB-MVDS-SDLXGB performs with different malware categories, Table 4 illustrates the detection accuracy for each malware category in the dataset. As can be seen in Table 4, there are nine malware categories in the testing dataset namely, Virus, Worm, Backdoor, Trojan, Downloader, Bot, Dropper, Spyware, Keylogger, and Generic. The majority of the malware in the dataset are either Generic or Trojan. This malware are belonging to different malware families. In most cases, the proposed MDEB-MVDS-SDLXGB model archives higher than 99.2%. However, deep investigation needs to be conducted on balanced malware families. Such investigation has been lifted for future study.

TABLE 4. Detection accuracy based malware category.

Malware Category	Total	Detected	Detection Accuracy (%)
Virus	381	381	100
Worms	587	587	100
Backdoor	524	524	100
Trojan	2108	2102	99.72
Downloader	104	104	100
Bot	182	180	98.90
Dropper	34	34	100
Spyware	184	183	99.46
Keylogger	87	87	100
Generic	3428	3405	99.33
Total	**7619**	**7587**	**99.58**

TABLE 5. Performance in terms of detection evasive malware (%).

Acc	FPR	FNR	Recall	Precision	F-Measure
98.76	2.49	0.78	99.22	99.09	99.16

TABLE 6. Performance comparison with related work.

	Acc	FPR	FNR	Recall	Precision	F1
The MDEB-MVDS-SDLXGB proposed	99.23%	1.56%	0.048%	99.52%	99.44%	99.48%
API Call Sequence Darem et al. [25]	98.30%	2.40%	1.18%	98.82%	97.64%	98.22%
API Call Sequence [5]	97.58%	2.45%	2.28%	97.61%	97.45%	97.53%
IAT based API Calls Liu et al. [31]	96.33%	3.29%	3.95%	95.95%	96.60%	96.27%

TABLE 7. The improvement gained by the proposed model.

	Acc	FPR	FNR	Recall	Precision	F1
API Call Sequence Darem et al.[25]	0.93%	-0.84%	-1.13%	0.70%	1.80%	1.26%
API Call Sequence Kang et al. [5]	1.65%	-0.89%	-2.23%	1.91%	1.99%	1.95%
IAT based API Calls [31	2.90%	-1.73%	-3.90%	3.57%	2.84%	3.21%

To evaluate the performance of the proposed MDEB-MVDS-SDLXGB model in terms of detecting evasive malware behavior, to memic evasive behavior, evasive malware samples are created by injecting APIs sequences related to benign samples into the malware APIs sequence to represent the evasive behavior. Table 5 shows the performance of detecting such evasive malware behavior. As can be noticed in Table 5, the performance has been slightly degraded as compared to the results on the original dataset before injecting the evasive behavior (See Tables 2 and 3). However, the performance of the proposed model is still higher than the other tested and also with the related work as compared in the subsequent section. The use of ensemble deep learning classifiers with diverse features sets exposes such an evasive technique.

VI. COMPARISON WITH THE RELATED WORK

The proposed model is compared with state-of-the-art related solutions. As mentioned earlier, most of the related work used API call sequences to construct the malware detection model [2], [4], [5], [25], [26], [31], [32]. Accordingly, the proposed model in this study was compared with the models that utilized the API calls features extracted from either dynamic or static analysis. The comparison with the model in [25] was made without providing the labels during the testing (assuming no human intervention), which is the main limitation of the model in [25]. As mentioned in Section 2 (the related work section), the models designed in [26], [31], [32] extracted the API calls from the import address table (IAT) of the PE files using static analysis. Meanwhile, the solutions in [2], [4], [5], [25] used API sequences extracted from dynamic analysis. Accordingly, two models were implemented for the comparisons, each consisting of two classifiers. The first model utilizes the API Calls Sequences that were extracted from the dynamic analysis, and the second model uses the IAT-based API calls to construct the first classifier. The second classifier is constructed using features extracted from the binary sets of the ***.text*** section of the PE samples. These features are commonly used in the literature in congestion with API-based features, as mentioned in [4], [5]. Both models were trained using the XGBoots algorithm due to its effectiveness for APIs classification compared to other machine learning techniques (as discussed in the previous section, see Figure 6 and reported in [5]). Figure 11 and Table 5 present the detailed performance comparison of the proposed model with the corresponding state of the art in terms of accuracy, recall, precision, F Score, and the false positive rate and false-negative rate.

Table 6 lists the performance comparison between the proposed model (MDEB-MVDS-SDLXGB) with three related works in which API features are extracted using dynamic analysis with adaptive deep learning classifier as in Darem *et al.* [25], API Call Sequences extracted from dynamic analysis as proposed in [5], and API Calls extracted from static analysis namely from the Import Addressable Table (IAT) as in [31]. As shown in Table 7, the proposed MDEB-MVDS-SDLXGB model outperforms the related work concerning all tested performance measures. The improvement gained by the proposed model is listed in Table 7.

As can be seen in Table 6 and 7, the overall performance in terms of F-measure of the proposed model is 99.48% which is 1.26% higher than Darem *et al.* [25], 1.95% higher than API Call Sequences extracted from dynamic analysis as proposed by Kang *et al.* [5], and 3.21% higher than the performance using the API Calls extracted from static analysis (IAT) as in Liu *et al.* [31].

To sum up, the results of the proposed MDEB-MVDS-SDLXGB model support the hypothesis of integrating different behavioral features to extract the hidden patterns that can effectively discriminate between benign and malicious programs. It is clear from the results of the deep learning-based classifier with combined features MB-MVDS-SDLC (see Tables 2 and 3) as compared to the performance

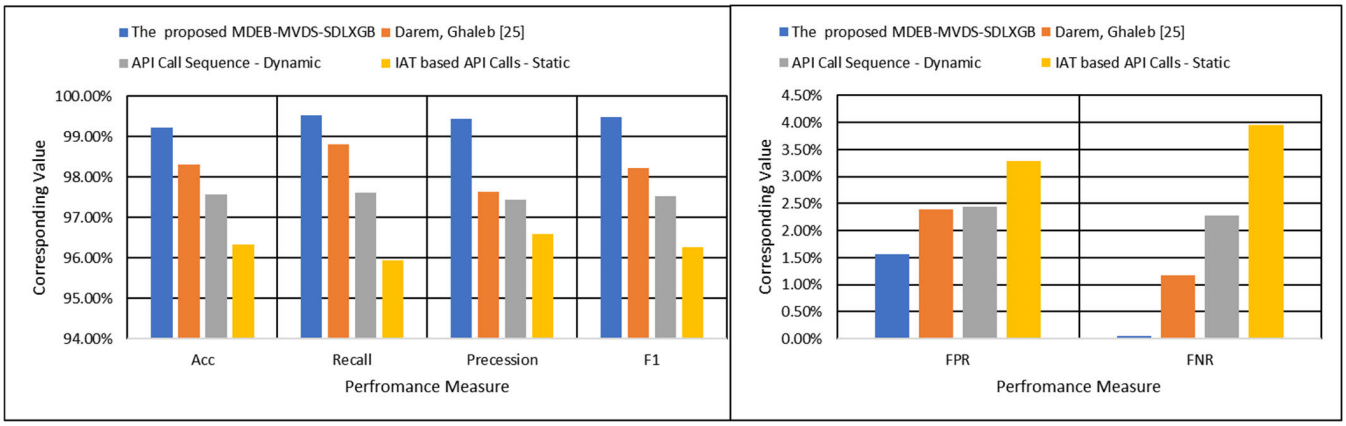

FIGURE 11. The performance comparison with the related work.

of the single feature set used in MB-MVDS-SDL, MB-MVDS-XGB, MB-MVDS-SVM, and MB-MVDS-RF (see Figures 7,8,9, and 10). The correlation between features was also considered. The ensemble-based learning contributed to considering different sets of patterns that malware can represent a wide range of behaviors, and this interprets the effectiveness of the ensemble classifiers MDEB-MVDS-SDLMV (see Table 2 and Figures 6 and 7) compared to the nonensemble-based model MB-MVDS-SDL (see Table 2 and Figure 8).

Although the proposed model attends the highest accuracy even with the tested evasive malware behavior as compared by the related works, a deep analysis of obfuscated and evasive malware behavior is needed. Because the main focus of this paper is on variant malware detection, the in-depth investigation of obfuscated and evasive malware behavior is lifted for future work. However, as shown in Table 5, the use of combined features with ensemble deep learning makes it possible for detecting evasive behavior especially when malware uses benign APIs sequence to evade the detection.

VII. CONCLUSION AND FUTURE WORK

This study proposes a multifaceted and Deep Ensemble Behavioral-based Malware Variant Detection Scheme using sequential deep learning and the Extreme Gradient Boosting algorithm. The proposed model combines different sets of behavioral features to detect the malware variants. The hypothesis is that each type of behavioral feature can tell a part of the maliciousness or goodness of the investigated executable file. A deep multifaceted hidden features vector is extracted automatically from the last hidden layer of a trained deep sequential learning model. Four deep learning models were constructed, each trained based on different sets of behavioral features such as API calls sequence, file access behavior, registry access, and network traffic. The hidden representative features are extracted from the hidden layer of each trained deep learning model and combined into one feature vector. These features are used as input to the XGBoost technique to train a set of ensemble classifiers. Ensemble-based learning creates multiple different patterns that represent different behavioral perspectives. An obfuscated malware variant can be detected and naturalized due to its difficulty in hiding its malicious behavior. The results show that the proposed model improves the detection accuracy while reducing the false-negative rate compared to the related evaluated models.

One challenge that may face the proposed detection model is evasive malware that does not show its malicious behavior during the feature extraction phase. A stealthy malicious program that behaves like a benign one can go undetected until specific conditions have occurred. One can think of including features from the static analysis to extract such statistical features. However, static features are subject to obfuscation by malware authors; thus, they can remain hidden. One should consider continuous monitoring of behavioral activities as a critical, challenging, and open research problem. Future research will extract features from the runtime environment to continuously monitor malware behavior and detect malicious patterns.

REFERENCES

[1] (2020). AV-TEST. *Malware Statistics and Trends Report*. (Dec. 27, 2020). [Online]. Available: https://www.av-test.org/en/statistics/malware/#: :text=Every%20day%2C%20the%20AV%2DTEST,potentially%20unwanted%20applications%20(PUA).

[2] M. Sun, X. Li, J. C. Lui, R. T. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for android," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1103–1112, May 2017.

[3] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: Detecting Android malware by ORGB analysis," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 55–69, Mar. 2020.

[4] 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," *Comput. Secur.*, vol. 84, pp. 376–392, Jul. 2019.

[5] J. Kang and Y. Won, "A study on variant malware detection techniques using static and dynamic features," *J. Inf. Process. Syst.*, vol. 16, no. 4, pp. 882–895, 2020.

[6] *Threat Report, in Webroot Smarter Cybersecurity*, H. Lonas, Ed., Webroot, Broomfield, CO, USA, 2018.

[7] X. Liu, X. Du, Q. Lei, and K. Liu, "Multifamily classification of Android malware with a fuzzy strategy to resist polymorphic familial variants," *IEEE Access*, vol. 8, pp. 156900–156914, 2020.

[8] B. A. S. Al-rimy, M. A. Maarof, M. Alazab, S. Z. M. Shaid, F. A. Ghaleb, A. Almalawi, A. M. Ali, and T. Al-Hadhrami, "Redundancy coefficient gradual up-weighting-based mutual information feature selection technique for crypto-ransomware early detection," *Future Gener. Comput. Syst.*, vol. 115, pp. 641–658, Feb. 2021.

[9] Y. A. Ahmed, B. Koçer, S. Huda, B. A. Saleh Al-rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *J. Netw. Comput. Appl.*, vol. 167, Oct. 2020, Art. no. 102753.

[10] A. G. Kakisim, M. Nar, and I. Sogukpinar, "Metamorphic malware identification using engine-specific patterns based on co-opcode graphs," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103443.

[11] 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," *J. Parallel Distrib. Comput.*, vol. 141, pp. 49–58, Jul. 2020.

[12] A. Khalilian, A. Nourazar, M. Vahidi-Asl, and H. Haghighi, "G3MD: Mining frequent opcode sub-graphs for metamorphic malware detection of existing families," *Expert Syst. Appl.*, vol. 112, pp. 15–33, Dec. 2018.

[13] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "Droidensemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.

[14] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ML-based visualization technique," *Comput. Secur.*, vol. 89, Feb. 2020, Art. no. 101682.

[15] S.-C. Hsiao, D.-Y. Kao, Z.-Y. Liu, and R. Tso, "Malware image classification using one-shot learning with siamese networks," *Proc. Comput. Sci.*, vol. 159, pp. 1863–1871, Jan. 2019.

[16] Z. Cui, X. Fei, X. Cai, C. Yang, G. G. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3187–3196, Jul. 2018.

[17] Q. Qian and M. Tang, "Dynamic API call sequence visualisation for malware classification," *IET Inf. Secur.*, vol. 13, no. 4, pp. 367–377, Jul. 2019.

[18] S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi, "A survey on malware analysis and mitigation techniques," *Comput. Sci. Rev.*, vol. 32, pp. 1–23, May 2019.

[19] Z. Salehi, A. Sami, and M. Ghiasi, "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values," *Eng. Appl. Artif. Intell.*, vol. 59, pp. 93–102, Mar. 2017.

[20] C. K. Patanaik, F. A. Barbhuiya, and S. Nandi, "Obfuscated malware detection using API call dependency," in *Proc. 1st Int. Conf. Secur. Internet Things (SecurIT)*, 2012, pp. 185–193.

[21] A. Sami, B. Yadegari, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 1020–1025.

[22] N. Kumar, S. Mukhopadhyay, M. Gupta, A. Handa, and S. K. Shukla, "Malware classification using early stage behavioral analysis," in *Proc. 14th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2019, pp. 16–23.

[23] R. Sihwail, K. Omar, and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 8, nos. 4–2, p. 1662, Sep. 2018.

[24] D. K. Mahawer and A. Nagaraju, "Metamorphic malware detection using base malware identification approach," *Secur. Commun. Netw.*, vol. 7, no. 11, pp. 1719–1733, Nov. 2014.

[25] A. A. Darem, F. A. Ghaleb, A. A. Al-Hashmi, J. H. Abawajy, S. M. Alanazi, and A. Y. Al-Rezami, "An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning," *IEEE Access*, vol. 9, pp. 97180–97196, 2021.

[26] J. Bai, Q. Shi, and S. Mu, "A malware and variant detection method using function call graph isomorphism," *Secur. Commun. Netw.*, vol. 2019, pp. 1–12, Sep. 2019.

[27] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "AsDroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 1036–1046.

[28] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *Comput. Secur.*, vol. 83, pp. 208–233, Jun. 2019.

[29] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. V. Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 65–79.

[30] S. Hameed, F. I. Khan, and B. Hameed, "Understanding security requirements and challenges in Internet of Things (IoT): A review," *J. Comput. Netw. Commun.*, vol. 2019, pp. 1–14, Jan. 2019.

[31] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 9, pp. 1336–1347, Sep. 2017.

[32] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[33] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct. 2004.

[34] B. A. S. Al-Rimy, M. A. Maarof, M. Alazab, F. Alsolami, S. Z. M. Shaid, F. A. Ghaleb, T. Al-Hadhrami, and A. M. Ali, "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.

[35] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–40, May 2018.

[36] M. Sikorski and A. Honig, *Practical Malware Analysis the Hands-On Guide to Dissecting Malicious Software*. 2012.

[37] S. A. Ebad, A. A. Darem, and J. H. Abawajy, "Measuring software obfuscation quality—A systematic literature review," *IEEE Access*, vol. 9, pp. 9903–99024, 2021.

[38] Zhang, H., Classification of ransomware families with machine learning based onN-gram of opcodes. Future Generation Computer Systems, 2019. vol. 90, pp. 211–221.

[39] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526.