


Article

Similarity-Based Hybrid Malware Detection Model Using API Calls

Asma A. Alhashmi ¹, Abdulbasit A. Darem ^{1,*} , Abdullah M. Alashjaee ², Sultan M. Alanazi ¹, Tareq M. Alkhalidi ³, Shouki A. Ebad ¹, Fuad A. Ghaleb ^{4,5}  and Aloyoun M. Almadani ¹

¹ Department of Computer Science, Northern Border University, Arar 9280, Saudi Arabia; asma.alhashmi@nbu.edu.sa (A.A.A.)

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

³ Department of Educational Technologies, Imam Abdulrahman Bin Faisal University, Dammam 34212, Saudi Arabia

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

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

* Correspondence: basit.darem@nbu.edu.sa or basit.darem@yahoo.com

Abstract: This study presents a novel Similarity-Based Hybrid API Malware Detection Model (HAPI-MDM) aiming to enhance the accuracy of malware detection by leveraging the combined strengths of static and dynamic analysis of API calls. Faced with the pervasive challenge of obfuscation techniques used by malware authors, the conventional detection models often struggle to maintain robust performance. Our proposed model addresses this issue by deploying a two-stage learning approach where the XGBoost algorithm acts as a feature extractor feeding into an Artificial Neural Network (ANN). The key innovation of HAPI-MDM is the similarity-based feature, which further enhances the detection accuracy of the dynamic analysis, ensuring reliable detection even in the presence of obfuscation. The model was evaluated using seven machine learning techniques with 10 K-fold cross-validation. Experimental results demonstrated HAPI-MDM's superior performance, achieving an overall accuracy of 97.91% and the lowest false-positive and false-negative rates compared to related works. The findings suggest that integrating dynamic and static API-based features and utilizing a similarity-based feature significantly improves malware detection performance, thereby offering an effective tool to fortify cybersecurity measures against escalating malware threats.

Keywords: malware detection; API calls correlation; feature similarity; static and dynamic analysis

MSC: 68M25



Citation: Alhashmi, A.A.; Darem, A.A.; Alashjaee, A.M.; Alanazi, S.M.; Alkhalidi, T.M.; Ebad, S.A.; Ghaleb, F.A.; Almadani, A.M. Similarity-Based Hybrid Malware Detection Model Using API Calls. *Mathematics* **2023**, *11*, 2944. <https://doi.org/10.3390/math11132944>

Academic Editors: Oliviu Matei and Rudolf Erdei

Received: 4 May 2023

Revised: 17 June 2023

Accepted: 26 June 2023

Published: 30 June 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Malicious software, commonly referred to as malware, is a constantly evolving threat, requiring effective detection solutions. Due to the prevalent use of obfuscation techniques by malware authors, new types of malware can elude detection, consequently posing substantial security threats. Malware propagation has reached significant figures due to the evolution of malware development tools that allow even novice individuals to create and spread massive threats, which seriously impact cyberspace [1]. Some hackers even provide malware-as-a-service, which can be used to create sophisticated malware that is challenging to detect using existing mechanisms.

The impact of malware can cause substantial damage to individuals and various sectors, such as petroleum, mining, banks, health, and financial sectors. Malware can be categorized based on its behaviors and characteristics into several types, such as viruses, worms, Trojan horses, and botnets [2–4]. Their behavior is represented by their functionality, impact on victim machines, and spreading mechanisms. For example, viruses need hosts

to spread and function, whereas worms are independent, duplicate themselves, and spread through the network autonomously. A Trojan disguises itself as benign software to gain unauthorized access to victim systems. Malware affects victim machines in different ways, including a loss of confidentiality, integrity, availability, authenticity, and/or accountability. For example, advanced persistent threats (APTs) may cause the leakage of sensitive or critical information. Botnets can be used to launch an attack against the availability of a system using victim machines. Ransomware is used to hinder access to the systems and information by either locking the target system or encrypting the victim's important files. The attacker then demands a ransom to unlock or decrypt the system. Some behaviors and characteristics of malware overlap, while other features are distinct to specific malware families. Such diverse types of features create sparse feature vectors that hinder the effective detection of novel malware.

The evolving landscape of cyber threats has seen the emergence of increasingly complex and destructive forms of malware. For instance, ransomware, known for its destructive and lucrative nature, typically encrypts a user's data and demands payment for the decryption key [5]. Another sophisticated malware type, Advanced Persistent Threats (APTs), are specifically designed to infiltrate systems stealthily and persist for extended periods, often targeting governmental and corporate systems [6]. In order to counter these threats, malware detection techniques are classified based on the type of analysis into static or dynamic analysis, or a hybrid of both. Static analysis, which offers a quick and resource-efficient way to examine a suspicious file without execution, provides insights into potentially malicious code segments and overall behavior [7–9]. This technique extracts malicious malware indicators from the Portable Executable (PE) file header and sections such as the binary sequence, the operation code, virtual and actual sections size, the strings, and the imported application program interface (API) functions from the dynamic link libraries (DLL), among many others [2,8,10–12]. Static analysis is particularly useful for the initial screening and filtering of benign files from potentially harmful ones and is effective against malware types such as trojans, adware, viruses, worms, etc. However, it can be thwarted by obfuscation and encryption techniques employed in more sophisticated malware such as APTs and ransomware. In contrast, dynamic analysis, which involves the actual execution of malware in a controlled environment, extracts features during runtime from the malware's interaction with the victim machine, such as API calls, system calls, system log files, windows registry modifications, and network traffic [2,8,10–12]. Dynamic analysis allows for an in-depth analysis of runtime behavior, capturing system interactions and behavioral patterns that sophisticated malware types, such as polymorphic malware, fileless malware, ransomware and APTs, find hard to entirely mask. This method, however, is resource-intensive and may lack scalability. Considering the strengths and weaknesses of both techniques, a hybrid approach utilizing both static and dynamic analysis offers a promising solution, providing a more robust and comprehensive defense against the ever-evolving malware threat landscape.

API calls and functions are common sources used by malware researchers to detect malware due to their effectiveness compared to other features. API-based features can be extracted from both static and dynamic analysis. The API functions extracted from the static analysis represent the expected behavior of the malware, while the API calls extracted from the dynamic analysis represent the actual behavior. These two sources of features are not mutually exclusive in malware analysis. Therefore, the presence of hybrid malware can be indicated by the absence of API calls during dynamic analysis, and its presence in the static analysis may indicate obfuscated malware. Moreover, the absence of the API functions during static analysis and their presence during dynamic analysis may indicate evasive malware behavior. The correlation between the API features that are extracted from the dynamic and static analysis can be used to improve the detection effectiveness of the existing malware solutions [13].

Although some hybrid malware detection models integrate features from both dynamic and static analysis as reported in the literature, to our knowledge, such correlations

have not been used to design an effective malware detection model. Therefore, a hybrid model that considers the correlation between the features extracted from the dynamic and static analysis is essential to improve the effectiveness of the existing hybrid models to overcome the challenge of evasive and obfuscated malware. In addition, in this study, the XGBoost and ANN classifiers were cascaded to significantly improve model performance by leveraging the complementary strengths of both models, performing effective feature engineering and ensemble learning, capturing non-linear relationships, and preventing overfitting. Furthermore, the complexity of malware behaviors and the continuous evolution of malware obfuscation techniques require advanced and intelligent detection systems that can adapt to new threats. The use of machine learning and data mining techniques has shown great potential in developing intelligent detection systems capable of identifying unknown malware with high accuracy.

In recent years, numerous research efforts have focused on developing hybrid malware detection models that combine features extracted from both static and dynamic analysis. Static analysis-based features are used to characterize the malware's silent features, such as metadata and structure, while dynamic analysis-based features represent the malware's behavior during runtime. However, the correlation between API calls extracted from static and dynamic features has not been fully utilized to improve the detection performance of hybrid models. This study aims to address such a challenge by designing and developing a similarity-based hybrid malware detection model based on API-based features extracted from static and dynamic analyses. The main aims of this study can be summarized as follows:

1. To develop a similarity-based hybrid malware detection model named HAPI-MDM, integrating static and dynamic analysis techniques. The hypothesis is that the presence of API calls in one approach and its absence in the other approach are an indication of obfuscation and can be used to improve detection performance.
2. To leverage the correlation between the API calls and functions extracted from dynamic and static analyses, thus enabling our proposed model to better detect evasive and obfuscated malware.
3. To utilize the extreme gradient boosting (XGBoost) and artificial neural network (ANN) algorithms in developing two prediction models based on features extracted from both dynamic and static analyses. The output scores of these models, along with similarity features, are then used to build a classifier using ANN algorithms for decision-making.
4. To improve the performance of the proposed model through a two-layer ensemble classifier design using the XGBoost and ANN algorithms. By cascading XGBoost and ANN, we aim to take advantage of the complementary strengths of both models, perform effective feature engineering and ensemble learning, capture non-linear relationships, and prevent overfitting.

The rest of this paper is organized as follows. Related work is reviewed in Section 2. The proposed model is described in Section 3, and the experimental setup is explained in Section 4. In Section 5, the results are presented, discussed, and analyzed. Section 6 concludes this study.

2. Related Work

Malware detection is a hot research topic and has been the subject of intensive studies. Based on the type of analysis used to extract malware features and indicators, malware detection solutions can be classified into three approaches: static, dynamic, and hybrid. In static analysis, the malware features are extracted from the malware binary file (the PE file) [14–18]. Meanwhile, in dynamic analysis, the malware features are extracted during the runtime in an isolated analysis environment [14,19–33]. Some studies hybridize the features from dynamic and static analysis to improve detectability [5,9,12,26,34]. Many types of features have been investigated in both static and dynamic analysis. Examples of features extracted from the dynamic analysis include but are not limited to the operation

codes (Opcode) in the .text section of the PE file [12,13,15,35,36], the byte sequence, the API functions from the imports and exports in the PE header [37], the strings from the .data section [38], etc. Some other features were derived from the PE header, such as creation date [39] and section size [5].

Amer [21] proposed a multi-perspective malware detection model using API call sequences that were extracted from dynamic analysis. The relationships between the API functions in the API sequence were considered. Word-impeding techniques were used to represent the relation between API functions in a sequence. However, the model uses only features extracted from the static analysis that are subject to noise and obfuscation. A features-integrated malware detection model was proposed by Zhang, Qin [12]. Two types of static features were used, namely, the opcodes and the API functions. The convolutional neural network technique was used to train a predictor based on opcode features. Meanwhile, the backpropagation neural network was used to train a predictor based on the API functions. The outputs of the two predictors are fused using the softmax function for classification. Similarly, static analysis-based features were considered in the model.

Ficco [19] proposed a malware classification model based on the sequence of API calls and association rules. The API features were extracted from dynamic analysis, and the ascension rules were formed using the recurrent subsets of API calls. The transaction probability between two consecutive API calls was calculated using the Markov chain-based models. It is worth noting that the proposed model investigated API-based features extracted from dynamic analysis. However, such an approach can be compromised by evasive malware that is aware of the analysis environment. Accordingly, the association rules may become irrelevant to such behavior, which may lead to the wrong classification.

Tari [26] proposed a malware detection approach called subcurve HMM based on a partial analysis of the API sequence. The Hidden Markov Model (HMM) model was used to extract malicious patterns from API call sequences. The proposed model focused on a subset of API call sequences that have high probabilities of matching in training the model and improving the effectiveness of the HMM-based detection model. The representative features were extracted by visualizing the API sequence transactions as a curve, and the subcurve that has higher properties was selected for training. Although the proposed model is promising, it is reported by the authors that the API call sequence subsets are sensitive to the training, which leads to an increase in the false-positive rate. Xiaofeng [28] proposed a malware detection model by integrating the API sequences extracted from dynamic analysis with the statistical features extracted from the API sequence. Two prediction models were constructed using deep learning, specifically the bidirectional residual LSTM and the machine learning model based on API statistical features. The output of these models was used as input to the random forest algorithm for the final classification. The statistical relationship among the system calls was used to train a prediction algorithm based on simplifying the association call graph. Although such a method is promising, depending solely on dynamic analysis-based features simplifies the evasion task created by the malware author. A smart malware that has evasive behavior may hide the malicious payload in the analysis environment and execute it in the operational environment. The model in [40] utilized the API features extracted from dynamic analysis and was trained using RF, SVM, LR, and NB; LR was reported as the best-performing classifier. Authors in [41] constructed an ensemble model using SVM as a base classifier. The API features were represented using doc2vec. In the model proposed in [42], dynamic analysis was used to extract the API calls and then the Indicator of Compromise features were also extracted and represented using the TF/IDF technique. The LR algorithm was used to construct the detection model.

To sum up, as illustrated in Table 1, API calls and functions are the common sources that malware researchers have used to detect malware due to their effectiveness compared to other features. API-based features can be extracted from either dynamic or static analysis. The expected behavior can be extracted from API functions imported in the Import Address Table (IAT) of the PE samples in the static analysis. In contrast, the actual behavior, which

is represented by the API call sequence, can be extracted from dynamic analysis. These two sources of features are not mutually exclusive in malware analysis. However, there is a lack of investigation into the correlation between the API calls extracted from static analysis and the API calls extracted from dynamic analysis. The absence of API calls during dynamic analysis and its presence in the static analysis may indicate obfuscated malware. Accordingly, the correlation between the API features extracted from the dynamic and static analysis can be used to improve the detection effectiveness of the existing malware solutions. Moreover, the absence of the API functions during static analysis and their presence during dynamic analysis may indicate evasive malware behavior. Accordingly, this study investigates the design of a model based on incorporating the correlation between dynamic- and static-analysis-extracted API sequences. The following sections detail the description of the proposed model.

Table 1. Summarizes the related works that use API-related features.

Author	Classification Techniques	Features Representation	Analysis	Limitations
Amer, Zelinka [35]	Markov model	Word-impeding	Dynamic	Single source of features
Zhang, Qin [12]	CNN	TF-IDF	Static	Rely on IAT-based API features, which are subject to obfuscations
D'Angelo, Ficco [19]	Markov chain-based models	Association Rules	Dynamic	It can be compromised by evasive malware.
Suaboot, Tari [26]	HMM	HMM transitions	Dynamic	High false-positive rate due to overfitting to the training set
Xiaofeng [28]	LSTM/RF	LSTM	Dynamic	Vulnerable to evasive behavior
Finder et al. [40]	LR	TF-IDF	Dynamic	Vulnerable to fake APIs injection
Ali et al. [42]	Ensemble-based SVM	doc2vec	Dynamic	Used unrepresentative features
Tran and Sato [41]	LR	TF-IDF	Static	Poor accuracy due to the API noise obfuscation used by malware

3. The Proposed Model

As shown in Figure 1, the proposed hybrid APIs-based malware detection model consists of seven phases as follows. The first phase is the API extraction from both dynamic and static analysis. The second stage involves the extraction of the API features using n-gram techniques. The third phase is the feature representation, in which each textual API feature is represented by a numerical number based on the term frequency. The fourth phase is to select the important API features from both dynamic and static analysis. The fifth phase consists of building dynamic and static API-based classifiers. The sixth phase consists of computing the cosine similarity between the static and dynamic analysis feature vectors. Finally, the seventh stage is to construct the hybrid-based malware detection classifier. A detailed description of each phase is given in the following subsections.

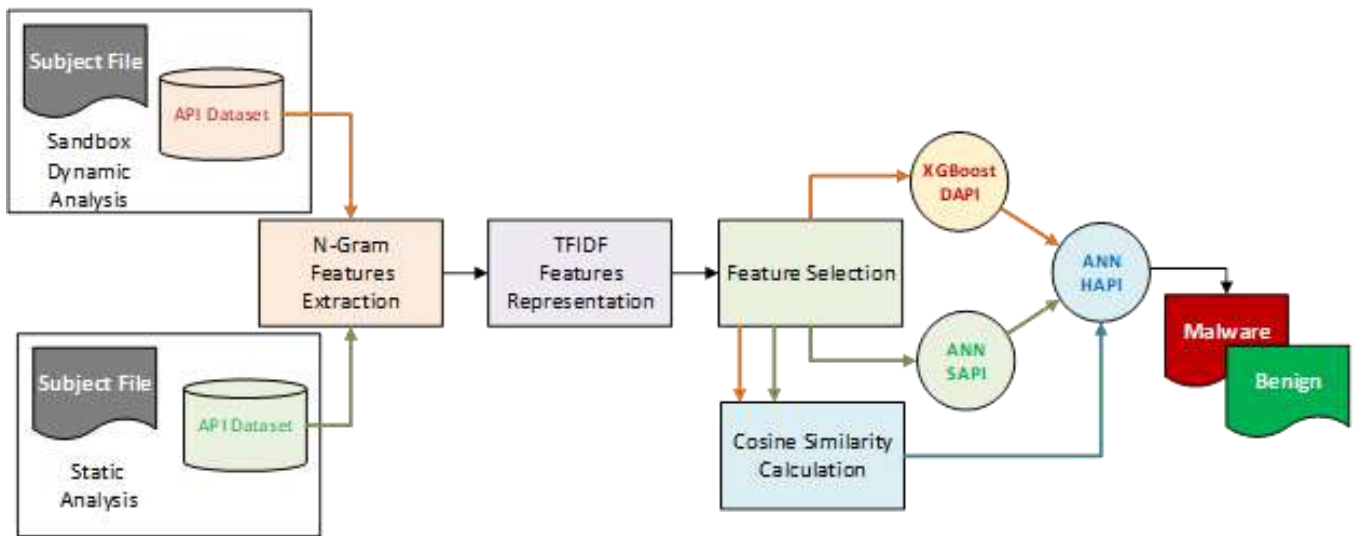


Figure 1. Overall stages.

3.1. Feature Acquisition Phase

In this phase, two types of features were extracted: static and dynamic features. The static features were extracted from the import table field in the header of the portable executable file. The import table of the portable executable file contains the API functions that the malware calls when the malware is executed. The second type of feature is the API calls features which are extracted from the dynamic analysis. In dynamic analysis, a malware file is executed in an isolated environment called a malware sandbox, such as a cuckoo sandbox. In the sandbox, a code is injected into the virtual space of the malware process to intercept API calls in a process called the hooking procedure. In the hooking procedure, the API calls are logged and thus used to represent the malware behavior. The extracted API functions can be categorized into five groups: process, memory, files, registry, and network. These functions can represent the subject's behavior from different angles. Consequently, API calls are commonly used features in dynamic analysis. Because process hooking degrades the system's performance, the most frequent APIs are usually logged. Because of this, static API calls will work as a complement to dynamic APIs.

3.2. Feature Extraction Phase

The API features logged in the previous phase are used as input for the feature extraction of this phase. The features were extracted from both dynamic and static analysis using the n-gram technique. In the n-gram technique [43], the feature sequences are extracted. Each feature sequence consists of one, two, or three API call functions. The hypothesis is that the co-occurrence of multiple API calls together in an executable file may indicate more granular details of its behavior.

3.3. Data Representation Phase

The API features that are extracted from the static and dynamic analysis are the names of the API functions, which are in textual format. The textual features need to be converted to numerical representation for data mining [2–4]. Term Frequency—the Inverse Document Frequency (TF-IDF) is commonly used to represent the textual data in its numerical forms using cross-entropy between malware samples and the population [2–5]. The TF-IDF is calculated based on two concepts: the term frequency (tf) and the document frequency (df). The term frequency (tf) is calculated based on the number of times an API function has occurred in the portable executable file in the static analysis, and it is the number of times that the API function is called in the dynamic analysis [1–4]. The document frequency (df) is the number of documents (samples) where the API function occurs. The inverse document frequency (idf) is used to penalize the weight of the common terms or the

commonly used functions, as their importance is low to distinguish between benign and malware behavior [4,5,8]. It also increases the weight of the rare API function, as it is an indication of specific behavior or characteristic of a malware sample. The $tf - idf$ can be calculated as follows:

$$tf - idf = tf \cdot \log \frac{N}{df} \quad (1)$$

where tf is the term frequency of the API function in a specific instance, df is the document frequency for the API function, and N is the number of samples in the dataset.

3.4. Feature Selection Phase

The features resulting from the feature extraction phase suffer from high dimensionality. Some of these features are too general, and others are too specific. The weight of the too-general feature approaches zero, forming a huge sparse matrix. The sparsity creates two main problems during the construction of the classifier, especially for machine learning techniques. Sparsity degrades both the classification accuracy and creates space and time complexity. In terms of classification accuracy, it is difficult to learn the behavioral or characteristic patterns from sparse matrices that well distinguish benign and malware samples. Meanwhile, the sparse vector increases the computational time to extract the useful pattern from the data as well as the memory space that the sparse vector can take. Therefore, feature selection is used to improve both the accuracy and efficiency of classification. In this study, important features based on information gain have been used to select API-based features. The entropy of the data is calculated and also for each class to select the set of features with high discrimination. For each feature in the dataset, the information gain between the feature and target class is calculated as follows:

$$S = - \sum p \log_2 p \quad (2)$$

$$IG = S - \sum_v \frac{|S_v|}{|S|} S_v \quad (3)$$

where S is the entropy of the dataset which is one for binary classification, p is the probability of the class in the database, and S_v is the relative entropy of the feature v .

3.5. Feature Scoring Phase

In this phase, each sample is scored based on its static characteristics and dynamic behavioral-based features using the selected features from the previous phase. Two machine learning techniques were used for the scoring: extreme gradient boosting (XGBoost) and artificial neural network (ANN) techniques. The XGBoost is used to train a model to score the APIs-based features that are extracted from dynamic analysis. XGBoost is more accurate than other machine learning classifiers for API-based features that are extracted from dynamic analysis, as reported by many researchers [10–12]. XGBoost is a kind of ensemble learning technique that uses the concept of a decision tree to classify parallel processing. It uses the gradient descent algorithm to minimize the error and boost the weak classifiers in sequential mode. It works based on the principle of boosting, which uses the residuals of weak classifiers to construct new stronger ones. To avoid the overfitting problem, a regularization based on tree pruning is used. The output of the XGBoost is two scores for each tested instance, the first score representing the maliciousness behavior and the second representing the being behavior of the sample.

The second scoring model is trained using ANN, namely the forward with backpropagation algorithm. ANN was originally used due to malware authors' use of obfuscation techniques to bypass detection. Moreover, ANN is a commonly used classifier in the domain of malware detection using static API-based features [12–19]. ANN can understand the complex relationship between the API functions called and the target class by automatically converging to a representation that maps the input to the output well. Similar

to the XGBoost, the output of the ANN is two scores for each tested instance, one for maliciousness characteristics and the other for the being characteristics of the sample.

XGBoost and ANN have different strengths and weaknesses and they complement each other. XGBoost is particularly effective at handling structured data with a large number of features, while ANN is better suited for handling unstructured data, such as images or text. In this study as shown in Figure 1, XGBoost and ANN are cascaded to gain the advantage of the complementary strengths of both models and achieve better performance than using either model alone. XGBoost is particularly effective at feature engineering, where the model creates new features that capture the relationships between existing features. These new features can then be used as input for the ANN, improving its performance. By cascading XGBoost and ANN, we can create an ensemble of two models that can learn from each other and improve each other's predictions. This can result in a more accurate and robust model. In addition, XGBoost is particularly effective at feature engineering, where the model creates new features that capture the relationships between existing features. On the other hand, ANN is particularly effective at capturing non-linear relationships between features. By cascading XGBoost and ANN, we can leverage the power of ANN to capture complex relationships that XGBoost may not be able to capture on its own. XGBoost uses regularization techniques, such as L1 and L2 regularization, to prevent overfitting. By cascading XGBoost and ANN, we can use these regularization techniques to prevent overfitting in both models, resulting in a more generalizable model.

3.6. Similarity-Based Feature Extraction Phase

In this phase, the similarity between the features extracted from the dynamic and static analysis is calculated. The API functions that are presented in the static analysis-based features and do not occur during the runtime are an indication of obfuscation and maliciousness. Similarly, the occurrence of API calls during the dynamic analysis and the absence of that feature in the dynamic analysis may suggest obfuscation and maliciousness. Thus, the similarity between the feature vectors representing the malware sample's characteristics and behavior is calculated using cosine similarity. Cosine similarity is an important measure for determining the correlation between two vectors because it provides a way to measure the similarity between the directions of two vectors, regardless of their magnitude. It is a crucial measure for determining the correlation between two vectors because it provides a simple and effective way to measure the similarity between their directions, which is often a key factor in improving ANN learning tasks. Cosine similarity is used to measure the distance between two vectors by calculating the angle between them when they originate from a particular point [18]. It can be calculated as follows:

$$Similarity = \frac{V \cdot W}{\|V\| \times \|W\|} \quad (4)$$

where V is the features vector from the static analysis, W is the features vector from the dynamic analysis, the dot (\cdot) product is calculated by the sum of the products of the corresponding elements in each vector, and the cross (\times) product of V and W represents the product of the magnitudes of V and W , symbolized by $\|V\|$ and $\|W\|$, respectively.

3.7. Decision-Making Phase

In this phase, the final decision is made about the maliciousness or goodness (benignness) of the sample. The static and dynamic scores calculated by the ANN and XGBoost classifiers, along with similarity-based features that were constructed in the previous phases, were used as input features to train an Artificial Neural Network-based classifier for decision-making, as shown in Figure 2. Algorithm 1 shows the online operation of the proposed model, HAPI-MDM.

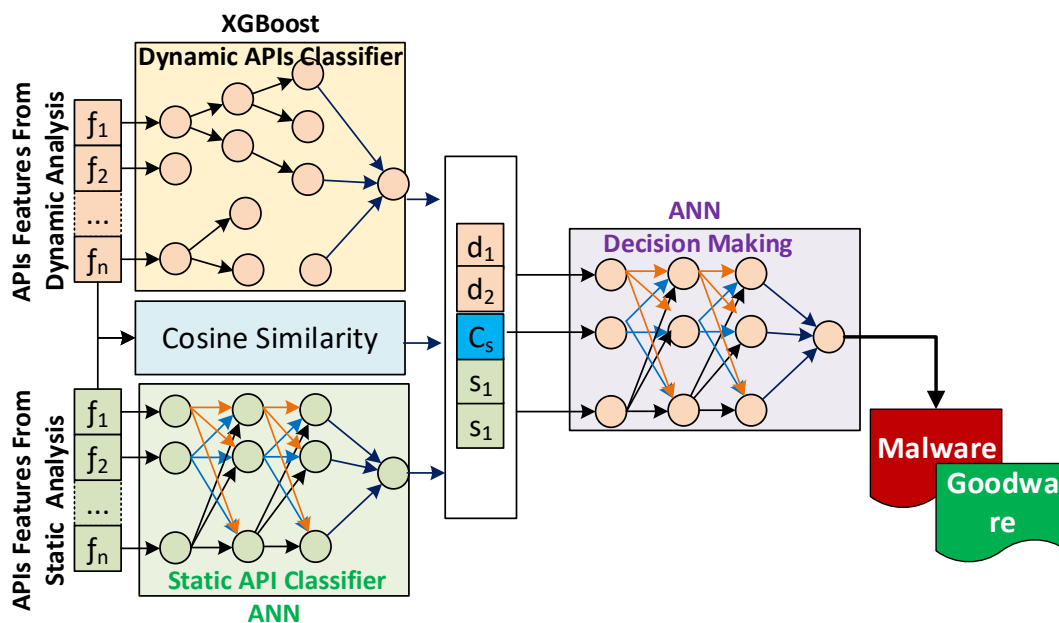


Figure 2. The proposed malware detection model.

As can be seen from Figure 2 and Algorithm 1, the APIs-based feature vector of the dynamic behavioral analysis is inputted to the trained XGBoost prediction model, while the selected features that are extracted from static analysis to characterize the samples are inputted to the neural network prediction model. The output of these two models is two values that represent the scores of how likely the malware sample is similar to the learned malware patterns and how likely it is similar to the known benign patterns, respectively. These two features have been used along with the cosine similarity-based features as input to the second ANN classifier as a decision-maker. The output of the decision-making classifier is the class of the sample, either malware or benign.

Algorithm 1: HAPI–MDM Online Operation

- 1: Extract static API features f_s
 $f_s \text{ extract} \leftarrow \text{import_table}(\text{sample})$
- 2: Extract the dynamic API features
 $f_d \text{ extract} \leftarrow \text{sand_box}(\text{sample})$
- 3: Features Extraction using n-gram in range of 1 and 3
 $f_s \text{ extract} \leftarrow n\text{-gram}(f_s, 1-3)$
 $f_d \text{ extract} \leftarrow n\text{-gram}(f_d, 1-3)$
- 4: Features Representations TF-IDF
 $f_s \text{ transform} \leftarrow \text{tf-idf}(f_s)$
 $f_d \text{ transform} \leftarrow \text{tf-idf}(f_d)$
- 5: Features Selection using information gain
 $\hat{f}_s \text{ select} \leftarrow \text{IG}(f_s)$
 $\hat{f}_d \text{ select} \leftarrow \text{IG}(f_d)$
- 6: First Stage Probabilistic Outputs of the trained XGBoost Classifier
 $f_{s-XGBoost} \text{ first stage probability} \leftarrow \text{XGBoost}(\hat{f}_s)$
 $f_{d-XGBoost} \text{ first stage probability} \leftarrow \text{XGBoost}(\hat{f}_d)$
- 7: Calculate Cosine Similarity

$$S = \frac{f_{s-XGBoost} \cdot f_{d-XGBoost}}{\|V_{f_{s-XGBoost}}\| \times \|f_{d-XGBoost}\|}$$
- 8: Merge the features
 $f \text{ merge} \leftarrow f_{s-XGBoost} \|S\| f_{d-XGBoost}$
- 9: Use the trained ANN model for final decision
 $d \text{ second stage: classify} \leftarrow \text{ANN}(f)$

In our methodology, the formation of feature vectors plays a pivotal role. These vectors are composed of individual features extracted from both static and dynamic analysis of malware samples. The static features are obtained from the Portable Executable (PE) file header and sections, including binary sequences, operation codes, virtual and actual section sizes, strings, and the imported application program interface (API) functions from the dynamic link libraries (DLL), among many others. Dynamic features, on the other hand, are extracted during runtime from the malware's interaction with the victim's host machine, such as API calls, system calls, system log files, windows registry modifications, and network traffic. These feature vectors serve two main purposes in our analysis. First, they provide a quantifiable representation of each malware sample's static and dynamic characteristics. This representation allows us to observe patterns and similarities between different malware samples that can be utilized for classification. Second, they serve as the input to our classification model, forming the basis of our malware detection approach. The feature vectors are input into the classification model following a normalization process to ensure each feature contributes equally to the model's outcome. Post normalization, these vectors are input into the classifier, which has been trained to distinguish between benign and malicious samples based on these features. The advantage of this approach is that it allows us to capture and utilize a wide array of information about each malware sample, combining insights from both static and dynamic analysis for more effective detection.

4. Experimental Design

This section presents the data set used, the experimental procedures, and the performance evaluation.

4.1. Dataset Description

The dataset used in this study consists of two types of samples: malware and benign. The portable executable samples are designed to run on the Windows operating system. The malware files were obtained from VxHeaven, a public repository containing more than 70 million malware files since 2008. This repository is commonly used for malware research [7,10–15,18–23,44]. The benign files were collected from a freshly installed Windows 7 operating system. The dataset contains 5000 samples, with 3000 malware and 2000 benign samples. Dynamic API features were extracted by executing the malware and benign files in a sandbox environment with Windows 7 as the guest operating system. Static features were also extracted from the import tables in the Portable Executable samples.

4.2. Experimental Procedures

The dataset was randomly divided into two sets: 70% for training and validation, and 30% for testing. The training dataset was used to train the XGBoost and ANN prediction models, and the prediction scores of these models were used to train the ANN-based decision-making classifier. The XGBoost model used 100 estimators, and the ANN prediction model used the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm as the optimization algorithm. BFGS is one of the variants of the gradient descent algorithm and has been shown to have better accuracy than the plain gradient descent algorithm. The learning rate was set close to zero, 10⁻⁵ for better generalizability. The neural network prediction model consists of three layers: the input, hidden, and output layers. The input layer has 396 neurons, the hidden layer has 15 neurons, and the output layer has 2 neurons. The second ANN classifier has three layers with five input neurons, twelve hidden neurons, and one output neuron.

4.3. Performance Evaluation

To assess the effectiveness of the proposed model, we utilized five commonly used performance measures in the literature, including overall accuracy, detection rate (recall), precision, F1 score, false-positive rate, and false-negative rate. Moreover, to evaluate the proposed model, we used machine learning techniques that were commonly used to

evaluate related works. We developed two models to evaluate the dynamic API-based features and the static API-based features: DAPI-MDM and SAPI-MDM, respectively. Both models underwent a two-stage training process involving XGBoost for feature extraction and pre-decision and ANN for the final decision on the sample’s class. Additionally, we compared the performance of our proposed model to that of related works [13,17], which used dynamic API features with LR and SVM models for training, and the model in [44], which used static-type features and was trained using the LR model. A detailed explanation of the results can be found in the following section.

5. Results and Discussion

Figure 3 and Table 2 show the results obtained from the proposed HAPI-MDM along with the comparison with the related work, namely, DAPI-MDM and SAPI-MDM. As seen in Figure 3, the proposed HAPI-MDM outperforms the other models concerning all studied performance measures. In terms of overall accuracy, the proposed HAPI-MDM achieves the best accuracy compared to DAPI-MDM and SAPI-MDM, while SAPI-MDM achieves the least accuracy.

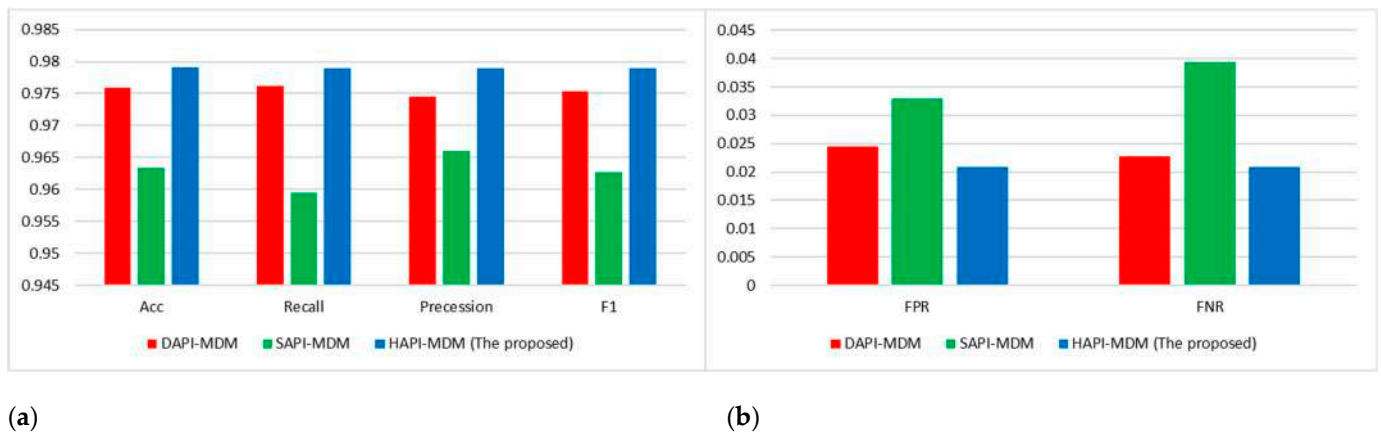


Figure 3. Detection performance in terms of (a) accuracy, recall, precision, and F1 score, and (b) FPR and FNR.

Table 2. Comparison with the related work.

	Acc	FPR	FNR	Recall	Precession	F1	AUCROC
SAPI-MDM	96.33%	3.29%	3.95%	95.95%	96.60%	96.27%	96.66%
FINDER 2022 [17]	86.45%	11.3%	16.47%	94.12%	90.67%	92.36%	89.69%
DAPI-MDM	97.58%	2.45%	2.28%	97.61%	97.45%	97.53%	97.50%
TRAN 2017 [13]	91.18%	9.52%	7.25%	91.45%	94.23%	92.82%	92.36%
ALI 2020 [41]	87.45%	17.12%	10.12%	90.14%	92.04%	91.08%	87.46%
HAPI-MDM (proposed)	97.91%	2.08%	2.08%	97.90%	97.90%	97.90%	97.91%

The dynamic analysis-based MDM, DAPI-MDM, performs better than the static-based MDM, SAPI-MDM. Due to malware authors’ use of obfuscation techniques, the APIs extracted from the dynamic analysis are misleading and do not represent the samples well. Accordingly, there is no clear discrimination between malware and non-malware. Meanwhile, in dynamic analysis, the accuracy performance is better than that of static analysis due to the difficulty in hiding the malware behavior. Thus, the mismatch between dynamically extracted APIs and statistically extracted APIs indicates obfuscation and further improves the proposed similarity-based hybrid MDM (HAPI-MDM) accuracy. As shown in Figure 4, HAPI-MDM also achieved the lowest false-positive and false-negative rate compared with the other models due to the good representation of the similarity-based feature used to boost the performance of the dynamic-based analysis.

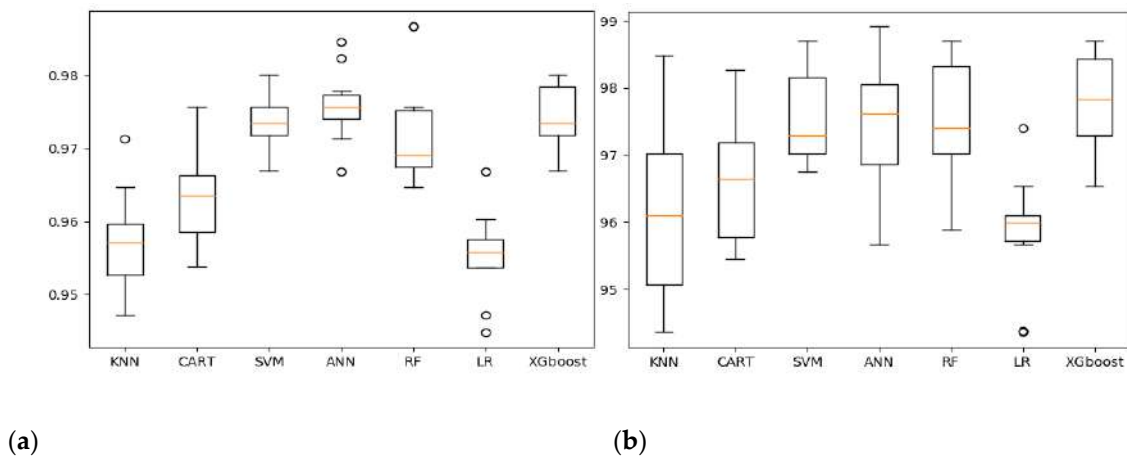


Figure 4. Statistical summary of the compared techniques: (a) static APIs; (b) dynamic APIs.

Figure 4 shows the performance comparison between different machine learning algorithms with API calls-based features extracted from dynamic and static analysis. The statistical summary in terms of the boxplot of the accuracy performance is obtained using a ten k-fold evaluation of seven machine learning techniques. These techniques were selected because they are commonly suggested for malware classification in related work. Overall, the statistical summaries of the classifiers evaluated based on the static API features are smaller than that of the dynamic APIs-based classifiers. This suggests that the features extracted from the static analysis agree with each other, which implies that the malware may use similar obfuscation techniques. Meanwhile, in dynamic analysis, the boxes are comparatively tall, suggesting that the malware samples have different behavioral activities. This is a valid interpretation because the datasets contain different types of malware that definitely have different behavioral activities.

Figure 5 illustrates the detailed accuracy results of the 10 K-fold. It shows that the ANN achieves the best performance compared to the other techniques studied, while the LR achieves the worst accuracy. Similarly, both ANN and XGBoost outperform the other studied techniques in dynamic analysis.

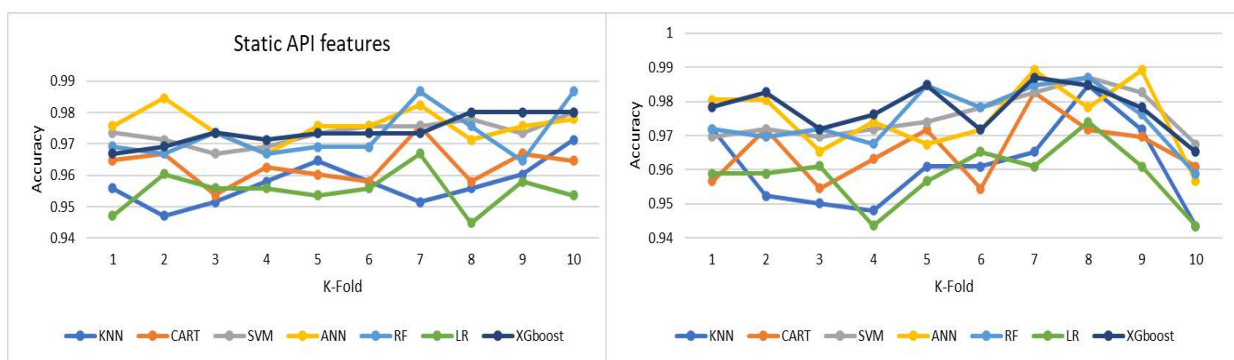


Figure 5. Detailed comparison of different machine learning techniques in terms of (a) static APIs and (b) dynamic APIs.

Table 2 shows the performance comparison between the proposed model and the related work. The proposed model outperforms the related work in all measures. The models in [39,40], which were constructed based on the dynamic API sequence, have achieved better performance in terms of the overall performance F-measure and the area under the curve (AUC) than the model proposed in [41], which was constructed based on

features extracted using the static analysis. However, both DAPI-MDM and SAPI-MDM, which are built using single features, have performed better than the related work based on single feature sets [13,17,41]. The reason for improvement gained by DAPI-MDM and SAPI-MDM is the architecture of the proposed model, which is based on two stages of learning where the XGBoost works as feature extraction for the ANN network in the second stage. The similarity measure in the HAPI-MDM model has improved detection performance as compared to the single features. As can be seen in Table 2, the improvement gained by the model is significant.

Figure 6a,b demonstrate the key characteristics that have been identified from both static and dynamic analyses. In Figure 6a, it can be observed that the ReadProcessMemory feature is the most frequently used by malware. The ReadProcessMemory function is a Windows API function that is utilized by malware to extract sensitive data from other processes running on a system, such as passwords or personal information. Another feature is the GetAsyncKeyState, which is another Windows API function that can be used by malware for keylogging purposes, and it retrieves the state of a particular key on a keyboard, including whether it is currently pressed or not. In Figure 6b, several API functions are highlighted in the word cloud, with the most notable being the NtQuerySystemInformation function. This function is a Windows API that retrieves different system information such as operating system version, computer name, processor architecture, and more. Malware can use this function to obtain information about other systems on the network, such as IP addresses, host names, and open ports, which can be used to identify potential targets for further attacks. This feature was discovered in the dynamic analysis and was not highlighted in the static analysis. Conversely, ReadProcessMemory appears to be notable in the static analysis-based API but is not highlighted as much in the dynamic analysis-based features. Therefore, both analyses’ APIs can complement each other to improve detection performance, which is interpreted as the performance improvement achieved by the proposed model compared to related work. Malware can inject its malicious code into a resource file, and then the LoadResource API function can be used to load the code into memory. The malware can then execute the code from memory, bypassing traditional file-based security solutions. Consequently, this API function is hidden from static analysis.

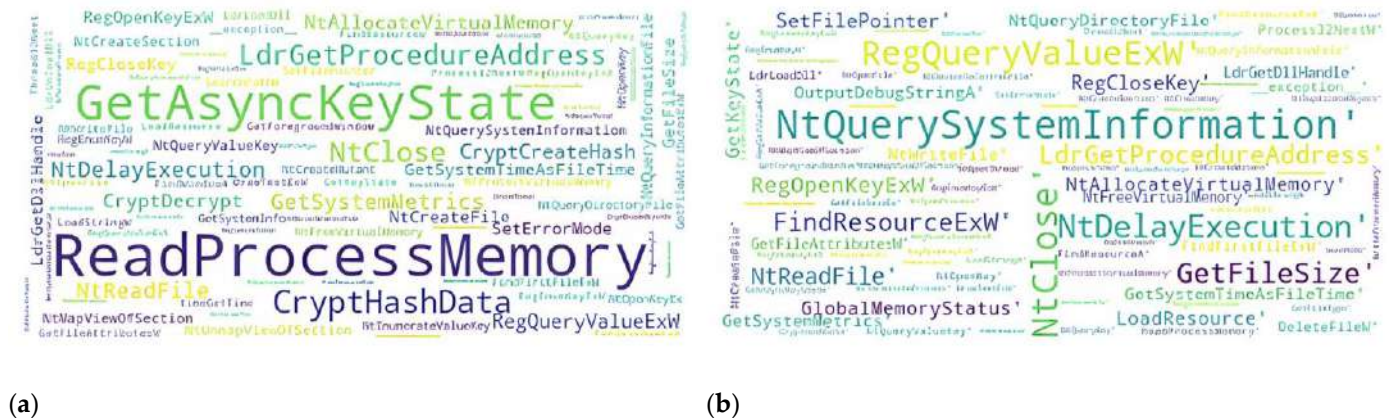


Figure 6. API features extracted from (a) static analysis and (b) dynamic analysis.

Our proposed model presents several advantages that contribute to its effectiveness in malware detection. It incorporates features from both static and dynamic malware analysis, enabling it to capture a wide array of information, thus making it effective in detecting different types of malware, including those that employ obfuscation techniques. A key facet of our model is the inclusion of similarity-based features, which have proven instrumental in detecting instances of obfuscated malware, often capable of eluding traditional detection models. Furthermore, we apply ensemble learning, using a two-layer ensemble classifier that combines XGBoost and ANN. This unique combination allows us to leverage the complementary strengths of these models, enabling the model to capture

non-linear relationships and prevent overfitting, thereby improving performance. However, we acknowledge a few limitations of our model. While our hybrid model performs commendably, the computational cost associated with dynamic analysis might affect scalability when dealing with a large volume of samples. Moreover, our model's performance heavily relies on the quality and relevance of the extracted features. There is a potential that if malware employs advanced obfuscation, significantly altering these features, our model's detection performance might be impacted. Furthermore, due to the dynamic nature of malware threats, our model may require frequent updates or retraining to maintain its effectiveness over time as new types of malware and obfuscation techniques emerge.

6. Conclusions

The conclusions drawn from the results of this study suggest that the proposed similarity-based hybrid malware detection model, HAPI-MDM, has demonstrated superior performance compared to the related works (97.91%). The comparison of the proposed model with other models that used dynamic and static APIs-based features showed that HAPI-MDM achieved the best overall accuracy, lowest false-positive rate (2.08%), and lowest false-negative rate (2.08%) among all models. The proposed model outperformed the related work in terms of all the performance measures.

The analysis of the extracted features revealed that combining both dynamic and static API features resulted in better detection performance compared to using a single feature set. The proposed similarity-based feature improved the detection performance of the dynamic-based analysis. In addition, the comparison of the machine learning techniques used to evaluate the model showed that ANN outperformed the other techniques in terms of accuracy, while XGBoost and ANN outperformed the other techniques when using dynamic API features.

Overall, the results suggest that combining dynamic and static API-based features and using a similarity-based feature can improve malware detection performance. The proposed hybrid model can potentially be used as a valuable tool to enhance the security of computer systems against malware attacks. Further research is needed to evaluate the model's performance with more diverse and larger datasets and to investigate the model's ability to detect new, unknown malware.

Author Contributions: A.A.A. contributed to conceptualization, dataset collection, writing—original draft preparation, and supervision; A.A.D. contributed to writing—original draft preparation, project administration, and funding acquisition; A.M.A. (Abdullah M. Alashjaee) contributed to investigation, resources, and draft preparation; S.M.A. contributed to formal analysis and visualization; T.M.A. contributed to methodology, writing, related work, and editing; S.A.E. contributed to methodology and related work; F.A.G. contributed to software, validation, methodology, and experimentation; A.M.A. (Aloyoun M. Almadani) contributed to the review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deanship of Scientific Research at Northern Border University, Arar, K.S.A. grant no. (RG-NBU-2022-1724).

Data Availability Statement: Data available on request from the corresponding author.

Acknowledgments: The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University for funding work through research group No. (RG-NBU-2022-1724).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Perwej, Y.; Abbas, S.Q.; Dixit, J.P.; Akhtar, N.; Jaiswal, A.K. A systematic literature review on the cyber security. *Int. J. Sci. Res. Manag.* **2021**, *9*, 669–710. [[CrossRef](#)]
2. Kim, J.-Y.; Cho, S.-B. Obfuscated Malware Detection Using Deep Generative Model based on Global/Local Features. *Comput. Secur.* **2022**, *112*, 102501. [[CrossRef](#)]
3. Kakisim, A.G.; Nar, M.; Sogukpinar, I. Metamorphic malware identification using engine-specific patterns based on co-opcode graphs. *Comput. Stand. Interfaces* **2020**, *71*, 103443. [[CrossRef](#)]

4. Kumar, N.; Mukhopadhyay, S.; Gupta, M.; Handa, A.; Shukla, S.K. Malware Classification using Early-Stage Behavioral Analysis. In Proceedings of the 2019 14th Asia Joint Conference on Information Security (AsiaJCIS), Kobe, Japan, 1–2 August 2019; pp. 16–23.
5. Moreira, C.C.; de Sales, C.D., Jr.; Moreira, D.C. Understanding Ransomware Actions through Behavioral Feature Analysis. *J. Commun. Inf. Syst.* **2022**, *37*, 61–76. [[CrossRef](#)]
6. Ouassini, A.; Hunter, M. Advanced Persistent Threats (APTs). In *The Handbook of Homeland Security*; CRC Press: Boca Raton, FL, USA, 2023; pp. 163–165.
7. Gandotra, E.; Bansal, D.; Sofat, S. Malware analysis and Classification: A survey. *J. Inf. Secur.* **2014**, *5*, 44440. [[CrossRef](#)]
8. Zhang, J.; Qin, Z.; Yin, H.; Ou, L.; Zhang, K. A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. *Comput. Secur.* **2019**, *84*, 376–392. [[CrossRef](#)]
9. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and Classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [[CrossRef](#)]
10. Al-rimy, B.A.S.; Maarof, M.A.; Shaid, S.Z.M. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Comput. Secur.* **2018**, *74*, 144–166. [[CrossRef](#)]
11. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [[CrossRef](#)]
12. Sibi Chakkaravarthy, S.; Sangeetha, D.; Vaidehi, V. A Survey on malware analysis and mitigation techniques. *Comput. Sci. Rev.* **2019**, *32*, 1–23. [[CrossRef](#)]
13. Galib, A.H.; Mainul Hossain, B.M. A Systematic Review on Hybrid Analysis using Machine Learning for Android Malware Detection. In Proceedings of the 2019 2nd International Conference on Innovation in Engineering and Technology (ICIET), Dhaka, Bangladesh, 23–24 December 2019; pp. 1–6. [[CrossRef](#)]
14. Baek, S.; Jeon, J.; Jeong, B.; Jeong, Y.S. Two-stage hybrid malware detection using deep learning. *Hum. -Cent. Comput. Inf. Sci.* **2021**, *11*, 10–22967.
15. Al-Hashmi, A.A.; Ghaleb, F.A.; Al-Marghilani, A.; Yahya, A.E.; Ebad, S.A.; Saqib, M.; Darem, A.A. Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model. *IEEE Access* **2022**, *10*, 42762–42777. [[CrossRef](#)]
16. Xiao, G.; Li, J.; Chen, Y.; Li, K. MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *J. Parallel Distrib. Comput.* **2020**, *141*, 49–58. [[CrossRef](#)]
17. Urooj, U.; Al-rimy, B.A.S.; Zainal, A.; Ghaleb, F.A.; Rassam, M.A. Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions. *Appl. Sci.* **2022**, *12*, 172. [[CrossRef](#)]
18. Cavnar, W.B.; Trenkle, J.M. N-gram-based text categorization. In Proceedings of the SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, USA, 11–13 April 1994.
19. D'Angelo, G.; Ficco, M.; Palmieri, F. Association rule-based malware classification using common subsequences of API calls. *Appl. Soft Comput.* **2021**, *105*, 107234. [[CrossRef](#)]
20. Ahmed, Y.A.; Koçer, B.; Huda, S.; Al-rimy, B.A.S.; Hassan, M.M. A system call refinement-based enhanced Minimum Redundancy Maximum Relevance method for ransomware early detection. *J. Netw. Comput. Appl.* **2020**, *167*, 102753. [[CrossRef](#)]
21. Amer, E.; Zelinka, I.; El-Sappagh, S. A Multi-Perspective malware detection approach through behavioral fusion of API call sequence. *Comput. Secur.* **2021**, *110*, 102449. [[CrossRef](#)]
22. Bylykbashi, K.; Qafzezi, E.; Ikeda, M.; Matsuo, K.; Barolli, L. Fuzzy-based Driver Monitoring System (FDMS): Implementation of two intelligent FDMSs and a testbed for safe driving in VANETs. *Future Gener. Comput. Syst.* **2020**, *105*, 665–674. [[CrossRef](#)]
23. Liu, X.; Lin, Y.; Li, H.; Zhang, J. A novel method for malware detection on ML-based visualization technique. *Comput. Secur.* **2020**, *89*, 101682. [[CrossRef](#)]
24. Al-Rimy, B.A.S.; Maarof, M.A.; Alazab, M.; Alsolami, F.; Shaid, S.Z.M.; Ghaleb, F.A.; Al-Hadhrami, T.; Ali, A.M. A Pseudo Feedback-Based Annotated TF-IDF Technique for Dynamic Crypto-Ransomware Pre-Encryption Boundary Delineation and Features Extraction. *IEEE Access* **2020**, *8*, 140586–140598. [[CrossRef](#)]
25. Yang, S.; Li, S.; Chen, W.; Liu, Y. A Real-Time and Adaptive-Learning Malware Detection Method Based on API-Pair Graph. *IEEE Access* **2020**, *8*, 208120–208135. [[CrossRef](#)]
26. Suaboot, J.; Tari, Z.; Mahmood, A.; Zomaya, A.Y.; Li, W. Sub-curve HMM: A malware detection approach based on partial analysis of API call sequences. *Comput. Secur.* **2020**, *92*, 101773. [[CrossRef](#)]
27. Dhanya, L.; Chitra, R.; Anusha Bamini, A.M. Performance evaluation of various ensemble classifiers for malware detection. *Mater. Today Proc.* **2022**, *62*, 4973–4979. [[CrossRef](#)]
28. Xiaofeng, L.; Fangshuo, J.; Xiao, Z.; Shengwei, Y.; Jing, S.; Lio, P. ASSCA: API sequence and statistics features combined architecture for malware detection. *Comput. Netw.* **2019**, *157*, 99–111. [[CrossRef](#)]
29. Grover, J.; Prajapati, N.K.; Laxmi, V.; Gaur, M.S. Machine Learning Approach for Multiple Misbehavior Detection in VANET. *Adv. Comput. Commun.* **2011**, *192 Pt III*, 644–653.
30. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016.
31. Palša, J.; Ádám, N.; Hurtuk, J.; Chovancová, E.; Madoš, B.; Chovanec, M.; Kocan, S. MLMD—A Malware-Detecting Antivirus Tool Based on the XGBoost Machine Learning Algorithm. *Appl. Sci.* **2022**, *12*, 6672. [[CrossRef](#)]

32. Elnaggar, R.; Servadei, L.; Mathur, S.; Wille, R.; Ecker, W.; Chakrabarty, K. Accurate and Robust Malware Detection: Running XGBoost on Runtime Data from Performance Counters. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 2066–2079. [[CrossRef](#)]
33. Saadat, S.; Joseph Raymond, V. Malware classification using cnn-xgboost model. In *Artificial Intelligence Techniques for Advanced Computing Applications*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 191–202.
34. Al-rimy, B.A.S.; Maarof, M.A.; Shaid, S.Z.M. Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection. *Future Gener. Comput. Syst.* **2019**, *101*, 476–491. [[CrossRef](#)]
35. Bai, J.; Shi, Q.; Mu, S. A Malware and Variant Detection Method Using Function Call Graph Isomorphism. *Secur. Commun. Netw.* **2019**, *2019*, 1043794. [[CrossRef](#)]
36. Zhang, H.; Xiao, X.; Mercaldo, F.; Ni, S.; Martinelli, F.; Sangaiah, A.K. Classification of ransomware families with machine learning based onN-gram of opcodes. *Future Gener. Comput. Syst.* **2019**, *90*, 211–221. [[CrossRef](#)]
37. Wang, W.; Gao, Z.; Zhao, M.; Li, Y.; Liu, J.; Zhang, X. DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features. *IEEE Access* **2018**, *6*, 31798–31807. [[CrossRef](#)]
38. Kang, J.; Won, Y. A study on variant malware detection techniques using static and dynamic features. *J. Inf. Process. Syst.* **2020**, *16*, 882–895.
39. Ramchoun, H.; Ghanou, Y.; Ettaouil, M.; Janati Idrissi, M.A. Multilayer perceptron: Architecture optimization and training. *IJIMAI* **2016**, *4*, 26–30. [[CrossRef](#)]
40. Finder, I.; Sheetrit, E.; Nissim, N. Time-interval temporal patterns can beat and explain the malware. *Knowl.-Based Syst.* **2022**, *241*, 108266. [[CrossRef](#)]
41. Tran, T.K.; Sato, H. NLP-based approaches for malware classification from API sequences. In Proceedings of the 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), Hanoi, Vietnam, 15–17 November 2017; pp. 101–105.
42. Ali, M.; Shiaeles, S.; Bendiab, G.; Ghita, B. MALGRA: Machine Learning and N-Gram Malware Feature Extraction and Detection System. *Electronics* **2020**, *9*, 1777. [[CrossRef](#)]
43. Darem, A.A. A novel framework for windows malware detection using a deep learning approach. *Comput. Mater. Contin.* **2022**, *72*, 461–479. [[CrossRef](#)]
44. Ebad, S.A.; Darem, A.A.; Abawajy, J.H. Measuring software obfuscation quality—A systematic literature review. *IEEE Access* **2021**, *9*, 99024–99038. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.