mathematics

MDPI

Article

Dynamic Extraction of Initial Behavior for Evasive Malware Detection

Faitouri A. Aboaoja [1,*], Anazida Zainal [1], Abdullah Marish Ali [2], Fuad A. Ghaleb [1], Fawaz Jaber Alsolami [2] and Murad A. Rassam [3]

[1] Faculty of Computing, Universiti Teknologi Malaysia, Iskandar Puteri 81310, Malaysia
[2] Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[3] Department of Information Technology, College of Computer, Qassim University, Buraidah 51452, Saudi Arabia
* Correspondence: faitouri@graduate.utm.my

Abstract: Recently, malware has become more abundant and complex as the Internet has become more widely used in daily services. Achieving satisfactory accuracy in malware detection is a challenging task since malicious software exhibit non-relevant features when they change the performed behaviors as a result of their awareness of the analysis environments. However, the existing solutions extract features from the entire collected data offered by malware during the run time. Accordingly, the actual malicious behaviors are hidden during the training, leading to a model trained using unrepresentative features. To this end, this study presents a feature extraction scheme based on the proposed dynamic initial evasion behaviors determination (DIEBD) technique to improve the performance of evasive malware detection. To effectively represent evasion behaviors, the collected behaviors are tracked by examining the entropy distributions of APIs-gram features using the box-whisker plot algorithm. A feature set suggested by the DIEBD-based feature extraction scheme is used to train machine learning algorithms to evaluate the proposed scheme. Our experiments' outcomes on a dataset of benign and evasive malware samples show that the proposed scheme achieved an accuracy of 0.967, false positive rate of 0.040, and *F*1 of 0.975.

Keywords: malware analysis approaches; machine learning-based malware detection models; evasive malware; feature extraction methods; box-whisker plot algorithm

MSC: 68T10

check for updates

Citation: Aboaoja, F.A.; Zainal, A.; Ali, A.M.; Ghaleb, F.A.; Alsolami, F.J.; Rassam, M.A. Dynamic Extraction of Initial Behavior for Evasive Malware Detection. *Mathematics* **2023**, *11*, 416. https://doi.org/10.3390/math11020416

Academic Editors: Oliviu Matei and Rudolf Erdei

Received: 23 November 2022
Revised: 30 December 2022
Accepted: 4 January 2023
Published: 12 January 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

Malware, i.e., malicious software, is a common term for several computer attacks. Several malware types, e.g., Trojans, can make attacks, such as viruses, rootkits, worms, spyware, or others [1]. Whereas the malware classes have been developed continuously, the malware detection models sound incapable of preventing all the threats and harm caused by malicious software. As stated by Kaspersky Labs in their report [2], there may have been approximately 5,638,828 unique computers that have been compromised. Additionally, [3] in 2017 reported a 36% increase in attack occurrences every day. Therefore, Cybersecurity Ventures' Official Annual Cybercrime estimated in their report [4] that the cost attributed to these system attacks will be around 6 trillion US dollars in 2021.

Although, previous malware authors developed malicious software without complicating their malicious code. Recent malware have been engineered to remain undetected. Existing malware authors employ obfuscation and evasion tactics, resulting in more intellective and dynamic malware which causes low detection accuracy of malware detection models [5,6]. Investigating the deployment of approximately 45,375 malware samples, ref. [7] declared that the usage of evasion techniques has increased by 12% among malware

in the last ten years, and the evasion techniques have been used by 88% of malware at the end of 2019 utilizing new evasion techniques instead of the old ones. Subsequently, the majority of malicious software consists of at least one evasion technique.

Many researchers have investigated different analysis approaches, including static, dynamic, and hybrid, to extract representative features to improve the detection accuracy. Static analysis extracts features from the malware executive files, the portable excitable (PE) files, without running the malware samples. In contrast, dynamic analysis extracts features from malware behavioral activities in real-time by running them in an isolated environment [8]. Even though static analysis is safer than dynamic and can investigate multiple execution paths, it is an insufficient approach with encrypted malware. In contrast, dynamic analysis is less safe than static analysis due to the malware being examined in real-time. Numerous studies [9–11] have shown that dynamic analysis is more resistant than static analysis and provides a more trustworthy detection performance. However, dynamic analysis is ineffective in producing all the malicious activities because its limited time prevents execution for all the potential paths [12]. In addition, in most cases, dynamic analysis cannot find malicious behaviors related to evasive malware that are context-aware. A malware checks the authenticity of the execution environment before the attack [13]. Using static and dynamic data as hybrid data has been conducted by some researchers to benefit from the advantages of both analysis approaches. Still, besides the advantages, the hybrid analysis approach also gathers the disadvantages of both analysis approaches. Using static and dynamic data, malware can be detected utilizing signature-based, behavioral-based, and heuristic-based approaches. However, each one of those approaches has its drawbacks, e.g., it is extremely difficult to detect unknown malware using the signature-based approach [14,15], more time and resources are required in the behavioral-based approach [16], and the heuristic-based approach is restricted to detect only malicious behaviors that are represented in the generated rules [17].

Most of the recent research [8,18–21] developed behavioral malware detection models to render the proposed models more effective and capable of detecting unknown malware behaviors. However, modern malware is capable of detecting the analysis environment and thus executes alternative legitimate behaviors [22] to conceal malicious activities. That is to say, based on the initial exploring the environment, a malware may display legitimate behaviors and, stop execute the malicious code or continue their activities using fewer, yet unrepresentative functions [20,23–25]. Thus, collecting the entirety of features offered by the malware during the run time misleads the machine learning techniques to train less effective classifiers. Even selecting the features based on frequencies, weights, or appearance leads to creating a feature set that contains unrepresentative behaviors belonging to the evasive malware when they identify the nature of the execution environment and perform alternative behaviors.

In this perspective, our paper aimed at extracting the features in the initial part of the execution time (the pre-attack stage). More specifically, we aim to determine the border between representative and unrepresentative data in the entire data shown by the evasive malware samples during their execution to extract only the part that contains the representative data. To this end, this study suggests a feature extraction scheme based on the proposed dynamic initial evasion behaviors determination (DIEBD) technique. The API calls sequence in terms of time series data containing the API calls based on their occurrence during the execution. To extract a feature set by which the evasion behaviors are sufficiently represented, the entropy values of sliding windows through the API sequence are calculated to represent the change in the API distribution and thus detect the behavioral change point in the concerned API call sequence. The entropy value of each time window measures the data distribution of the API calls. Consequently, the box-whisker plot algorithm is utilized to keep track of the constructed sliding windows in each API sequence and evaluate their entropy values to determine the sliding window where behavior starts to change. To the best of our knowledge, the proposed (DIEBD)-based feature extraction scheme considers the changing behavior window as an indicator of the start of the unrepresentative data,

which may be noise, legitimate behaviors, or repeated behaviors. Finally, in the detection stage, the state-of-the-art machine learning algorithms are trained and evaluated using the features that are suggested by the proposed scheme. The experimental result shows that the proposed feature extraction scheme can reinforce the developed model to detect evasive malware effectively. The contributions of this paper are summarized as follows:

1. A feature extraction scheme based on a dynamic initial evasion behaviors determination (DIEBD) technique was proposed to extract the features by which the initial evasion behaviors were presented by effectively identifying the boundaries between the initial evasion behaviors and the unrepresentative behaviors in each instance.
2. A malware detection model was developed using state-of-the-art machine-learning techniques to validate and evaluate the proposed scheme, conducting an in-depth comparative analysis between the proposed DIEBD-based feature extraction scheme and the recent related feature extraction schemes in terms of the obtained classification accuracy and other detection metrics.
3. A comprehensive experimental evaluation was carried out demonstrate the improvement that the DIEBD-based feature extraction scheme had made.

The rest of this paper is organized as follows: the related work is highlighted in Section 2, while in Section 3 the evasion attack scenario is introduced. Section 4 shows the framework used in this paper. The experimental design is presented in Section 5, and the result analysis and discussion are addressed in Section 6. This paper is concluded in Section 7.

2. Related Work

A few studies, such as [23,24], have concentrated on evasive malware behaviors analysis to estimate the increase in malware behaviors complexity and sophistication. While P. Rodrigo et al. in [23] demonstrated that anti-VM evasion techniques are available in more than 81% of the database of 4 million malware samples, ref. [24] performed static analysis to investigate over 17,000 malware samples for the presence of evasion techniques. It was found that both generic and sophisticated malware are increasingly employing evasion techniques, such as anti-debugging and anti-VM. On the other hand, to examine the dynamism of malicious behaviors, ref. [26] applied a virtual environmental condition generator (VECG) to provide multiple environments and gather malicious behaviors under various conditions. The most common behaviors were selected to generate a unique signature for each family, with the result that their proposed model was shown to be 97% accurate. Another study by [22] designed the cheating engine SCARECROW, which simulated the common features of the analysis environments that have been observed by the evasive malware. SCARECROW is installed on the end host machine to convert it into similar analysis environment features in order to deceptively deactivate the evasive malware. Since it is a heavy duty to simulate all the fingerprinting of the analysis environment at the same time, the evaluation of SCARECROW provided an accuracy of 89%. Additionally, malware can easily detect the existence of SCARECROW when they investigate if their running environment reflects multiple virtual environment features from several sources like VMware and VirtualBox.

Studies in [7,13,27] introduced a particular way to detect evasive malware through designing and developing transparent analysis environments. Ref. [27] constructed an indistinguishable analysis environment. They presented BareCloud, with no in-guest monitoring components, as an evasive malware detection system by comparing the malware behaviors that were obtained on the BareCloud system with the behaviors that were captured on virtual-based systems. Further, MORRIGU is designed and developed by [13] in their work as a transparent analysis environment, which matches both automated and human-driven analysis based on five different common evasion techniques to classify evasive malware. Ref. [7] developed an evasion techniques analysis framework that can analyze the executable files in four levels of instruction, APIs, sys calls, and memory access and circumvent the malicious samples through hiding the control environments related

artifacts for 92 evasion techniques that were collected from the published studies. To evaluate the proposed analysis framework, a total of 45,375 malware samples and 516 benign samples are collected and analyzed using the proposed framework. On the other hand, several studies, such as [28–30], have been conducted to design and develop malware detection and classification models with no regard to evasive malware by removing malware samples capable of performing evasion techniques.

Most of the existing studies [8,18–21,31] applied the cuckoo sandbox as a malware analysis environment to recognize malware class characteristics using behavioral-based malware detection models. Ref. [18] built their proposed behavioral model based on a collection of API calls using malware and benign samples during the run time. Further, the most commonly used APIs were categorized into groups according to the data flow dependencies to construct sequence traces, which were then passed into the heuristic function to generate the corresponding actions. Malware and legitimate behaviors were represented based on binary vectors depending on the presence of each action in benign and malicious samples. Authors in [19] collected the API calls at the user level, and system calls at the kernel level using the cuckoo sandbox and kernel driver, respectively. Furthermore, feature ranking was used to select the optimal feature set, which was represented using the frequency histogram method. As a result of training and evaluating several machine learning classifiers, random forest (RF) has achieved the best performance based on the integrated user and kernel features.

On the other hand, based on temporal patterns, ref. [31] introduced a malware detection framework that relies on temporal abstraction and time interval techniques to discretize the API call occurrence values and create the interval patterns, respectively. Furthermore, a predefined threshold is used to specify the most frequent temporal patterns in each class. To evaluate the proposed temporal pattern features, temporal probabilistic profile (TPF) classifiers and machine learning classifiers, namely SVM, LR, and NB, are trained using non-time-based, order-based, and temporal pattern datasets. As a result, the developed models with temporal patterns provide the best performance.

Based on different behavioral features, API calls, network activities, and readable strings, ref. [8] established a behavioral-based malware detection model. The string features were extracted and selected using text mining and singular value decomposition (SVD) techniques, respectively. Furthermore, based on the integrated feature set, ensemble-based machine learning algorithms were trained and assessed to obtain an accuracy of 99.54%. In addition, ref. [20] implemented a malware detection model based on the *n*-gram technique, which was used to extract the dynamic API calls. Subsequently, *TF-IDF* is used to select only the irrelevant *n*-grams that were transformed into binary vectors used later to train the machine learning classifiers. As an outcome, the Logistic Regression (LR) classifier seemed to have the highest accuracy of 98.4%. Moreover, ref. [21] created a dataset by running over 20,000 samples covering eight malware families in the Cuckoo sandbox to acquire API call sequences. During the training phase, the behaviors of each malware family are constructed using the long-short-term memory (LSTM) algorithm. The proposed model, according to their findings, has an accuracy range of 83.5–95.5%. In [32], the authors examined natural languages processing techniques such as *TF-IDF*, paragraph vectors with (the PV-DM Distributed Memory Model, and (PV-DBOW) Distributed bag of words by using these techniques as vectorizers to transfer the API sequence into vectors that are fed into machine learning algorithms for training and testing purposes. The highest accuracy of 99% was obtained by the SVM support vector machine classifier using *TF-IDF* as the vectorizer technique.

However, detecting evasive malware by comparing malware behaviors obtained on various platforms to identify the deviation in such behaviors as evasion behaviors leads to capturing the inefficient behaviors when the malware recognizes the nature of all those environments, especially all the environments employing emulated or visualized components. Additionally, this approach suffers from missing the unknown evasion techniques that are not circumvented by those instrumented environments. Moreover,

constructing transparent analysis environments suffered from several weaknesses, such as malware may be capable of detecting the analysis environments in terms of user activity conditions. Furthermore, while the transparent analysis environments are thought to be indistinguishable, there is a chance of being recognized by malware when specific software and hardware are examined [27]. Additionally, the transparent analysis environments are designed and developed based on identified evasion techniques that have been known by researchers and used when those environments modify specific parameters to trigger the malicious behaviors to be appearing or hiding the predefined controlled environments-related artifacts that are often looked for by evasive malware. As a result, that analysis environment failed to trigger the extremely sophisticated malware that uses unknown evasion techniques to exhibit its malicious activities and thus the malicious behaviors related to the evasive malware looked unlike artifacts [13,33]. Moreover, temporal pattern-based models suffer from high misclassification rates when they deal with malware that performs similar legitimate behaviors. On the other hand, collecting the entire runtime malware behaviors without considering the effect of evasion tactics, which are extensively employed by malware to determine the nature of the execution environment, and hence hide, change, or stop their activities, leads to capturing unrepresentative data from which the developed model can be learned how to inaccurately detect that sophisticated malware. By observing the problem that was mentioned, this research aims to determine the border between representative and unrepresentative data for each evasive malware sample.

3. Evasive Malware Behaviors

To increase the potential of the weapon's effectiveness in malicious activities, the evasive malware benefits from achieving the evasion techniques to be capable of evading the detection and thus identifying that they are being executed in an isolated environment such as a cuckoo sandbox, which is distinguishable from the real machine [34]. The malicious software looks for specific indicators using evasion techniques to recognize that they are being analyzed. Practically, anti-analysis techniques are applied to discover hardware characteristics related to analysis environments, installed tools, processes and services, serial numbers or MAC addresses, and registry entries to determine whether malware is being operated in a sandbox or not [30].

Even though the evasion techniques are utilized by few legitimate software to protect intellectual property, N. Galloro et al. in [7] remarked that the evasion techniques that benign samples have never used are those often used by malware samples. Such kinds of malware recognize the nature of the execution environments by checking the system artifacts like the virtual environment processors, hardware ID, and human interactions [13,35–42]. Subsequently, the evasion techniques are utilized to detect registry data, downloaded applications or monitoring tools, processes, services, serial numbers or MAC addresses, and memory architecture, all of which indicate that the malware is processed in virtual environments [30].

Moreover, another strategy has been used by the evasive malware to avoid detection when this malware waits for a specific time period before they execute the malicious activities [13]. In addition, part of the evasive malware usually waits for a particular event number, whether keyboard or mouse, to be done through their network communication as a condition for starting their malicious behaviors; otherwise, they act benignly [13,26]. In addition, time-based evasion techniques have been exploited by evasive malware for measuring time-flow and CPU clock ticks to discover the existence of an analysis environment due to most analysis tools causing a slowdown during the execution of the processes [7]. Malware frequently uses this tactic by checking a threshold for how long a series of activities take to be performed (e.g., invoking library functions, or making system calls) [36].

On the other side, our claim to detect the evasive malware at an early stage is built based on the result of [43,44], whose work that showed that most of the behaviors that appeared by malicious samples in a sandbox are observed during the first two minutes of execution. To the best of our knowledge, evasive malware needs to apply its evasion

techniques during execution (logically at the beginning of their executions) to determine whether they are being executed in an analysis environment or not. As a result of the inspection stage, the evasive malware either performs malicious activities or alternative behaviors, such as executing as legitimate, stopping, or continuing their malicious activities with fewer functions [20,23–25]. Therefore, the constructed API call sequences contain consecutive different behaviors, which are the inspection behaviors at the beginning of the execution and either malicious or alternative behaviors after the malware identifies the execution environment execution.

Based on such an attack scenario, this study assumes that these consecutive behaviors offer API calls that are called in different distributions. Therefore, an entropy analysis is conducted to measure the distribution of each API call in the API call sequence. The main idea is that the change in the entropy value means an alteration in API call distributions, which indicates that different behavior is performed by the concerned instance. Accordingly, our hypothesis is established to identify when the attack behavior starts to change and considers the changing point as a border between the evasion behaviors and the next alternative behaviors. Therefore, we propose our DIEBD-based feature extraction scheme to collect the representative evasion behaviors from which the representative feature set is extracted based on the execution point that reflects the change in the behavior.

In our constructed dataset, as cases for evasion behaviors, the instances with the IDs 30, 145, and 7392 opened the Bois information registry key using the NtOpenKey() API call and read the system start-up information using the NtQueryValueKey() API call to determine whether they were being analyzed or not because bios information, such as the BOIS version for the virtual environments, is distinguishable. In addition, the malware instances numbered 608, 668, and 10,331 performed the IsDebuggerPresent() call to read the field "Being debugged" in the Process Environment Block (PEB) structure that exists with each process, and thus recognize if the execution environment is an isolated or real environment. Moreover, the NtDelayExecution() call was frequently achieved by malware samples with IDs of 4 and 8119 at the beginning of their execution to attempt to stall the execution for a long time period without showing malicious behaviors. Both instances under IDs 197 and 7650 intermittently used the GetCurseorPos() call to estimate the reality of mouse movement. To retrieve system information, such as the number of processor cores, which vary between the real and analysis environments, some malware samples, such as 86 and 152, performed the GetSystemInfo() call.

Figures 1 and 2 show, respectively, the entropy-based API call distribution for benign and malware samples. The figures illustrate how the API calls distribution can be visualized using the entropy values of each API call that are arranged in slide windows with a width of 100. The axis X represents the number of the API call, while the axis Y represents the entropy value of that API call. The labeled numbers in both figures represent the ID of the samples whose behaviors are displayed in the figures. Figure 1 demonstrates that the behaviors of benign samples are gradually changed between the initial and the end stages of execution. In contrast, almost all the malware samples in Figure 2 start the execution with behaviors that differ from their next stage execution behaviors. As a result of the sharp change in the malware behaviors that take place suddenly, the change behavior point (window) can be recognized. Therefore, the sharp behavior change points that are offered by malware samples can be used as indicators to determine the boundary of the evasive behaviors.

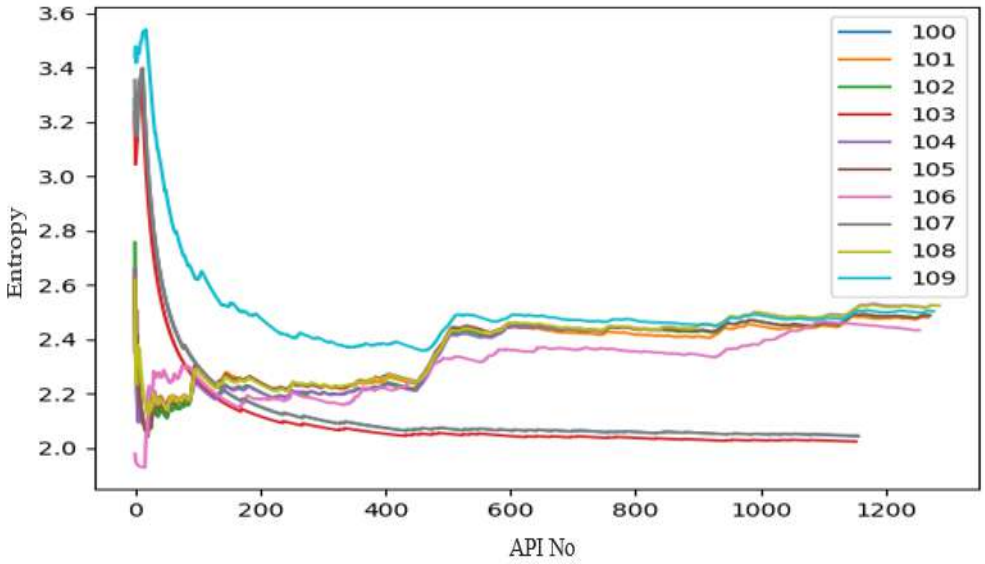

Figure 1. Entropy-based API call distribution of benign.

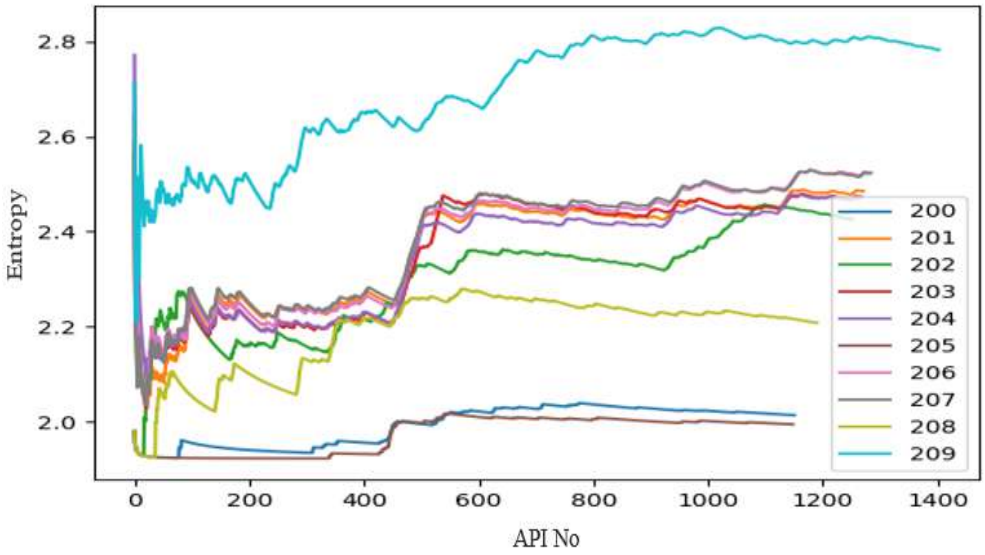

Figure 2. Entropy-based API call distribution of malware.

4. The Framework

Figure 3 shows an overview of the suggested framework. First, the evasive malware and benign samples are executed and monitored with the help of the cuckoo sandbox, and then JSON reports are obtained. Second, a Python module has been developed to collect the raw data, which consists of API calls that are invoked by each evasive malware and benign sample in the form of an APIs sequence for each sample. Third, the distribution of data is estimated by calculating the entropy value for each window in the obtained API sequences. Fourth, to identify the behavior-changing window as the starting point of the alternative behaviors of evasive malware, the box-whisker plots algorithm is implemented to produce a dynamic evasion behaviors dataset. Finally, machine learning classifiers are trained and tested based on the final dynamic evasion behaviors dataset to evaluate our proposed scheme.

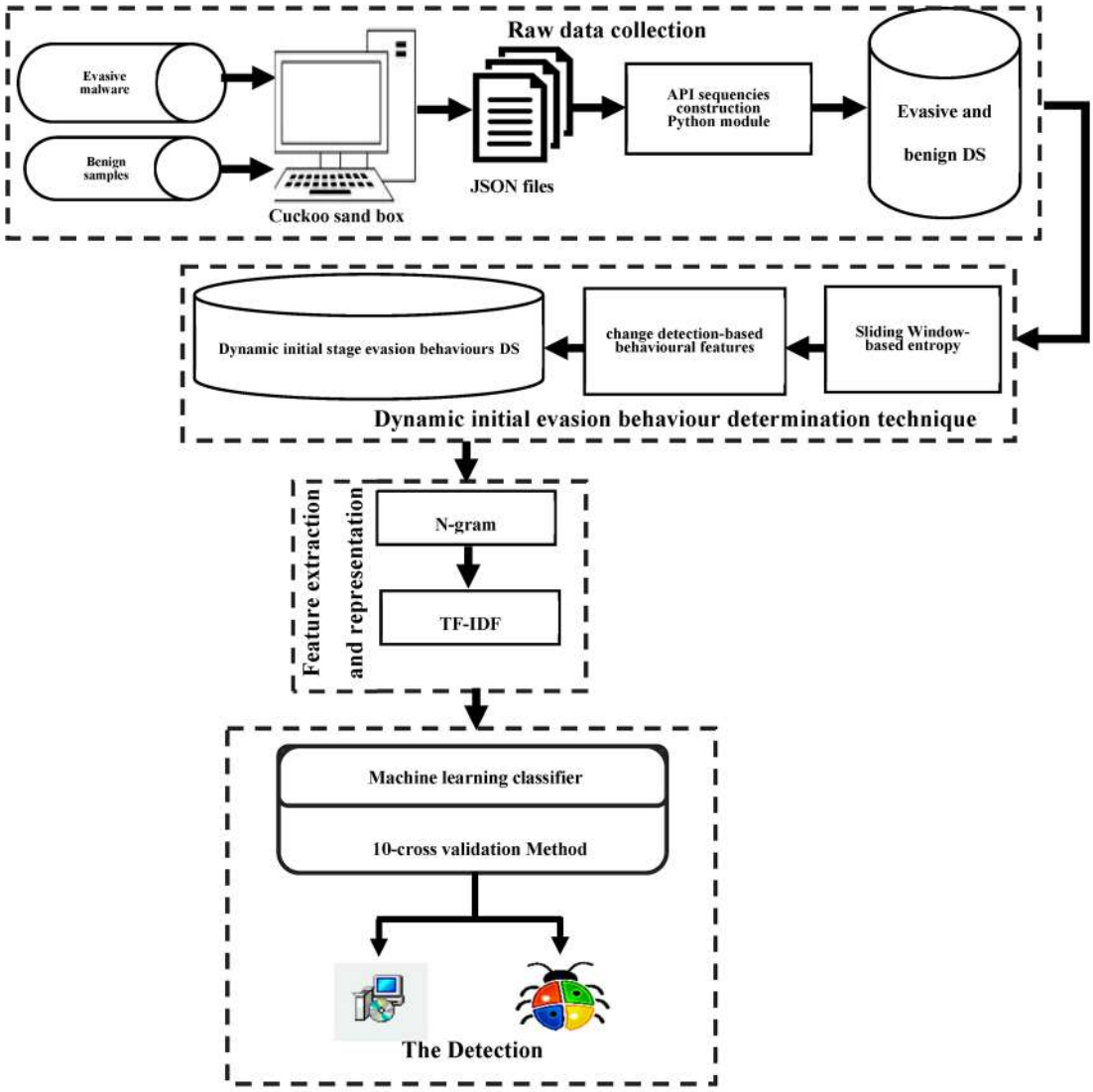

Figure 3. An overview of the framework.

4.1. Raw Data Collection

The API calls and their parameters are written and arranged according to their occur time in the form of JSON files by running the evasive malware and benign samples one by one in the cuckoo sandbox, which is a widely used simulator and analysis tool in the malware detection community [45–48]. Consequently, a python language module is developed and applied to generate an evasive malware dataset comprising the sample IDs and the corresponding API sequence that is organized depending on the time of occurrence of each API call during the run time of that executable file, resulting in Coma separated value (CSV) file as a database holding the sequences of API calls that are invoked by each evasive malware and benign sample. We picked API call sequences as malicious attributes because the order in which they are called reveals how malware interacts with an operating system, such as file IO, registry read/write, and so on [32]. Behavioral data generated from the dynamic analysis could have both representative data where the malware still examines the execution environment and unrepresentative data when the malware already recognizes the existence of the analysis environment.

4.2. Data Pre-Processing

The data pre-processing stage contains two steps: calculating the window-based entropy values and capturing the representative behaviors based on the changes in the

behaviors at each behavior sequence, which are considered as boundaries separating the evasion behaviors from the unrepresentative behaviors executed after the concerned instances recognize the nature of the execution environment. The collected data appear to contain both representative and non-representative data since evasive malware perform real behaviors as long as they do not recognize the characteristics of the execution environment, and alternative behaviors are performed when they determine that their execution environment is an analysis environment and then regenerate their behaviors to be similar to legitimate behaviors [8,12,18], or stop running by repeating some operations to consume the time, and provide noisy behaviors [49–51]. As a result, determining the boundary between the representative and the unrepresentative data directly is problematic.

4.2.1. Sliding Window-Based Entropy

We assume that there is a variation between the distribution of the API calls that appeared in the representative data part (evasion behaviors) and those that appeared in the unrepresentative data part (alternative behaviors). The entropy values have been calculated to represent the changes that potentially occur in the sequential data distributions [8,52,53]. Such studies inspire us to divide each API call sequence into API windows with a width of 100 to create a sequence of windows for each instance, where each window contains the same number of API calls n = 100 and the step that represents the distance between the current window and the next window equals 50, which produce overlapping windows sequence. We found in our experiments that the larger window sizes are insufficient for the short sequences. Furthermore, smaller window sizes may be more sensitive than larger window sizes to recognize the periods at which behavior changes. Consequently, using the entropy library from Python language, the entropy value of each window is calculated utilizing the following Equation (1):

$$H(x) = -\sum_{i=1}^{n} p(i).logp(i) \tag{1}$$

where $H(x)$ represents the measured entropy value, $p(i)$ indicates the proportion of ith API in the series of API windows that represent the API sequence of a specific sample, n is the total number of API call in each API sequence. The entropy distribution calculation process is shown in Figure 4.

4.2.2. Change Detection-Based Behavioral

Temporal analysis of the windows sequence is performed in the second step by tracking the deviation of the entropy values that are calculated for each window. The box-whisker plot algorithm is used to examine the variation in entropy values for the windows in the generated sequence to find the window that displays the behavior-changing point from which the evasive malware starts to perform another behavior different from the previous behaviors, so our assumption considers the gained detection point as a boundary between the evasion behaviors by which the evasive malware try to know the nature of the environment and the next behavior of that instance after the environment investigation step. The box-whisker plot is a non-parametric quantitative technique for presenting time series data elements without knowing their internal representation or distribution [54]. The box-whisker plot evaluates each window according to that window's generated temporal summary components. The created box-whisker-plot temporal summary for each window contains the four components, which are the median (μk), the entire quartile range (IQR), the upper limit (UL), and the lower limit (LL). As a result, Equation (2) can be used to describe the sequential summary of each window.

$$TSk(i) = \begin{cases} \mu k = (Q1 + Q3)/2 \\ IQR = (Q3 - Q1) \\ UL = Q3 + (K * IQR) \\ LL = Q3 - (K * IQR) \end{cases} \tag{2}$$

where $Q1$ and $Q3$ represent the first and third quartile, respectively, K refers to a constant value, and TS denotes the temporal score of the received ith windows sequence from the sample k. Therefore, by comparing the entropy value of each window to the upper and lower limit (UL, LL), the algorithm can identify if the received entropy is still in or out of the determined entropy range of the previously created windows sequence, and thus move to the next window in case the current entropy value still includes the entire entropy range of the previously obtained windows, or stop at this window to be the last window in the created API calls list in case the entropy value is out of the range of the entire entropy range, and then cut out the API call sequence from the point at which the changing entropy value is appearing in.

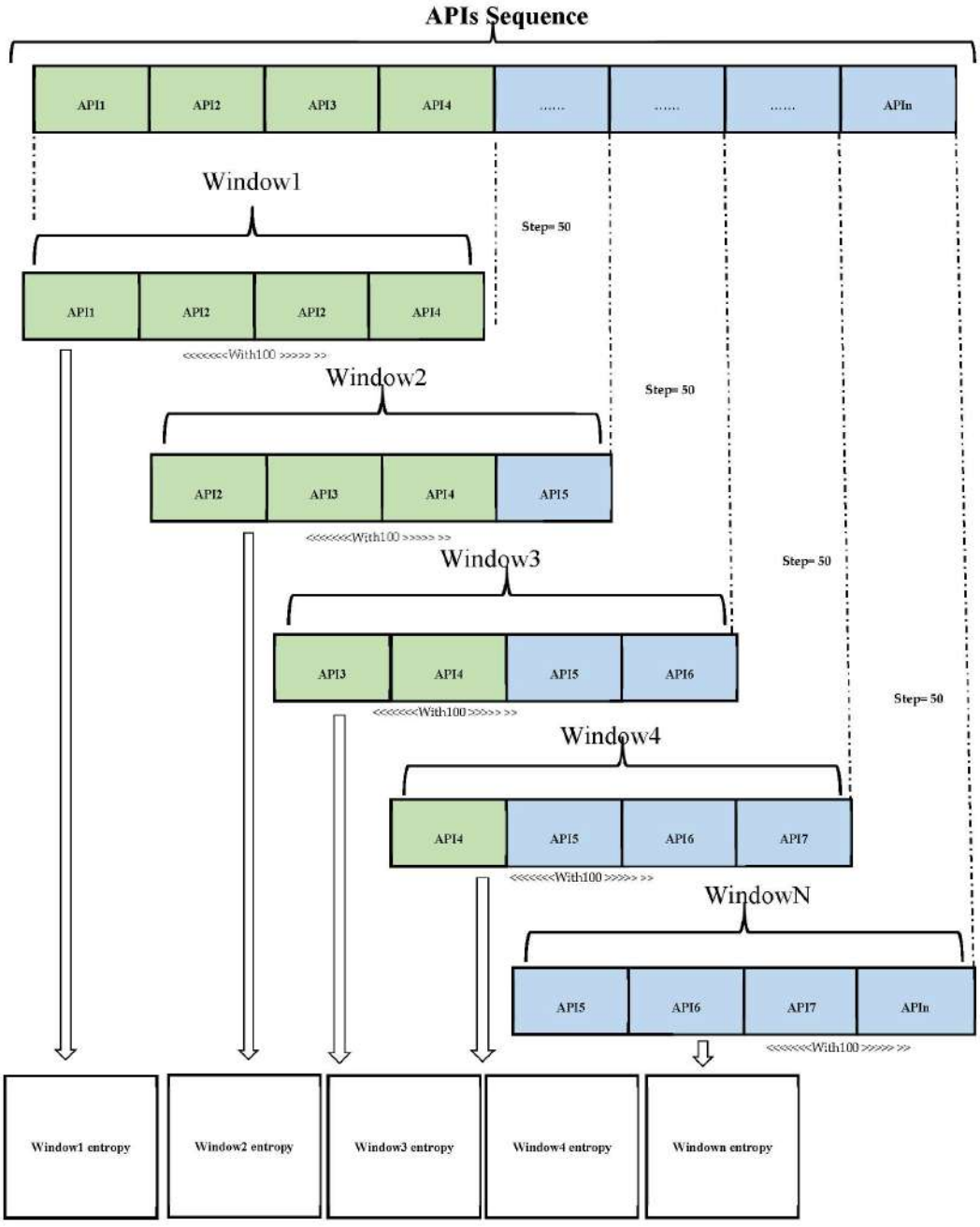

Figure 4. Entropy distribution representation.

4.3. Feature Extraction and Representation

The *n*-gram technique has been proposed by several studies [55–58] for extracting sub-text features with length N from the original string. In our study, we extract 2-g from the API sequences of each evasive malware and benign sample in our database. We picked the *n*-gram-based API sequence feature because malware must call multiple API calls rather than a single API call to execute their damage [47]. Therefore, the corresponding *n*-gram-based API sequence features must be valuable in understanding and modelling malicious behavior.

In addition to converting the received text data into numerical data, which is a suitable form to be fed into machine learning techniques, the term frequency-inverse document frequency (*TF-IDF*) technique has been implemented in our feature extraction and representation stage [21]. *TF-IDF* works based on the basic concept that a text that occurs in almost all the executable files is not a useful determiner, so it must acquire less weight than one that occurs in a few files [59]. Weight-based vector with length of 1150, which provides the best performance, is generated to represent the extracted behaviors of each sample. This study determined the size of 2 for the *n*-gram technique because the total number of the generated features is sharply increased when the *n*-gram size is increased [47,53] and thus, more processing time is required. Therefore, a high dimensionality of the feature set is the main cause behind developing an overfitting model [60,61]. This paper used 2-g rather than 1- as both are small sizes of *n*-gram because there are dependencies when neighboring API calls [62]. These dependencies can create a semantic relationship between the consecutive API calls when taking the sub-sequences of API calls as features instead of considering each feature as an independent feature.

We compute the *TF-IDF* for each developed text feature (*n*-gram) utilizing the following mathematical equation:

$$TF - IDF = TF(i, j) \times DF(i) \tag{3}$$

where $TF - IDF$ represents the computed weight of each 2-g-based feature, $TF(i, j)$ and $IDF(i)$ are calculated using Equations (4) and (5), respectively.

$$TF(i, j) = \frac{n(i, j)}{\sum_k (k, j)} \tag{4}$$

where $TF(i,\ j)$ is the recurrence of each 2-g-based feature i in 2-g sequence j, $n(i, j)$ indicates how many times the 2-g-based feature i appears in 2-g sequence j, and $\sum\ k\ (k,\ j)$ represents the number of the 2-g-based features in 2-g sequence j. Consequently, *IDF* is calculated using Equation (5).

$$IDF(i) = log2\ \frac{|D|}{|\{j : i\ \epsilon\ j | j\ \epsilon\ D\}|} \tag{5}$$

where $IDF(i)$ describes how much the 2-g-based feature i is uncommon in all the 2-g sequences, and D represents the number of all the 2-g sequences. $|\{j\text{: } i \in j\,|\, j \in D\}|$ means the number of 2-g sequences that consist of the 2-g-based feature i. As a result, an initial evasion behaviors dataset has been generated to contain the vectors by which the API call sequence-based 2-g features are represented using the corresponding weight values in each evasive malware and benign sample.

4.4. The Detection

As we are interested in developing a malware detection model with a high detection rate, this study introduced a machine learning-based malware detection model instead of a deep learning-based model because deep learning requires a huge amount of training data to train an effective model and provide a satisfactory detection rate, while machine learning needs a smaller amount of training data to train a sufficient model by which an acceptable detection rate can be achieved [56,63,64]. This paper used machine learning algorithms

across the classifier types that have been widely applied in the literature, such as K-nearest neighbor (KNN), regression trees (CART), naive bayes (NB), support vector machine (SVM), artificial neural network (ANN), random forest (RF), logistic regression (LR), and extreme gradient boosting (XGBoost). To evaluate the proposed DIEBD-based feature extraction scheme, machine learning classifiers are evaluated using the K fold cross-validation method with K = 10 utilizing 40% of the dataset which is randomly reserved for testing purposes as unseen samples, while the remaining 60% of the created dataset has been used to train the selected machine learning models.

5. Experimental Design

The procedure for evaluating the suggested model is discussed in this section. We review how the experimental environment is set up, describing the dataset that is used and the performance metrics that were employed.

5.1. Experiment Environment Setup

The Cuckoo sandbox 2.0.7 is used to build up a dynamic analysis environment on the Ubuntu 18.04 host machine to record malicious and benign executable file behaviors in real-time. Sandboxes are solutions that malware analysts and researchers typically employ to perform dynamic analysis [65–67]. The virtual box was used in conjunction with Cuckoo sandbox tools to create an isolated, supervised, and simulated environment for malware analysis. The sandbox infrastructure is set up according to the instructions in [68]. To enable command and control (C & C) communication and prohibit the propagation of malicious behavior during the analysis, a virtual box with a host-only adapter has been employed. The experiments were performed on the guest machine, which is Windows 7 Professional SP1. Several software and applications, including Microsoft Office, Google Chrome, Adobe Acrobat Reader, and Mozilla Firefox, were loaded on the guest machine to establish a simulated testing environment. In the internal storage of the guest machine, user-like directories were also created in various places. Malware and benign samples are executed on the virtual machine one by one for a predetermined time period. Consequently, when the malware or benign sample finishes its execution, or the experiment reaches the end of its time, the JSON behavioral report corresponding to that instance is generated and saved on the host machine, whereas the virtual machine is restored to the cleaned state. As a result, to execute all the candidate malware and benign executable files, two folders are created to contain the collected JSON reports of malware and benign files separately. The experiments of this study are carried out on ubuntu 18.04 with Intel(R) Core(TM) i7-4790 CPU @3.60 GHz and RAM of 16.0 GB.

5.2. The Dataset

Obtaining a representative dataset was a difficult task because most researchers never grant access to their datasets and our concentration on evasive malware with the unavailability of the public dataset of evasive malware [13]. Our dataset consists of API call sequences representing the behaviors of 7208 evasive malware collected from [7] and [69] as well as 3848 benign samples collected from [70] and the freshly installed Windows 7 operating system. This dataset is created by executing all the evasive malware samples in the constructed analysis environment described in Section 5.1. Table 1 shows the evasive malware types and benign samples.

Table 1. Evasive malware types and benign files.

Class	Evasion Techniques	Number		Source
malware	Hardware id-based evasion	82		[69]
	Bios-based evasion	6		[69]
	Processor feature-based evasion	134		[69]
	Exception-based evasion	197		[69]
	Timing-based evasion	689		[69]
	Wait for keyboard	3		[69]
	Not available	6096		[7]
	Malware Total Number		7208	
Benign		2000		Win7
Benign		1848		[70]
	Benign Total Number		3848	

5.3. Performance Measures

The performance of the proposed DIEBD-based feature extraction scheme has been evaluated using the common metrics used by the researchers, which include False positive rate (*FP*), False negative rate (*FN*), detection accuracy (*ACC*), detection rate (*DR*), precision (*P*), and *F*-measures (*F*1). Furthermore, *FP* represents the ratio of benign labelled as malware, while *FN* indicates the percentage of malware classified as benign [71]. While the accuracy (*ACC*) measures the proportion of successfully detected samples using Equation (6), the detection rate (*DR*) computes the fraction of malicious samples that are correctly classified using Equation (7) [72]. In addition, the Precision (*P*) measures the ratio of malware samples that are predicted as malware among all the malware and benign samples that are predicted as malware using Equation (8). *F*-measure (*F*1) totalizes the mean value of precision and detection rate using Equation (9) [72].

$$ACC = \frac{TP + TN}{Tp + TN + FP + FN} \tag{6}$$

$$DR = \frac{TP}{TP + FN} \tag{7}$$

$$P = \frac{TP}{TP + FP} \tag{8}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{9}$$

6. Results Analysis and Discussion

Two stages were carried out to assess the efficacy of the proposed DIEBD-based feature extraction scheme. Because the *K* values of the whisker-box plot can impact the detection performance, several *K* values including (0.5, 1.0, 1.5, 2.0) were investigated in order to emphasize the impact of the *K* value on the proposed scheme performances. Accordingly, first, based on the *K* values, four subsets of the datasets, each with a different length of features vector, were extracted. The best *K* value was identified by using the datasets which are created utilizing different *K* values to train and test the classifiers. Each machine learning classifiers, including KNN, CART, NB, SVM, ANN, RF, LR, and XGBoost, were trained using the four subsets of datasets separately utilizing 60% of each dataset, while the remaining 40% of each created dataset is used to evaluate the selected classifiers through 10-cross validation method. Second, for the comparison with the related work models which use the entire data without cutting the API call sequences when the behaviors are changed to train the developed classifiers, the dataset which contains API call sequences with all the invoked API calls are used to train and evaluate the same machine learning classifiers. Figure 5 shows the 10-cross validation-based training and test phases.

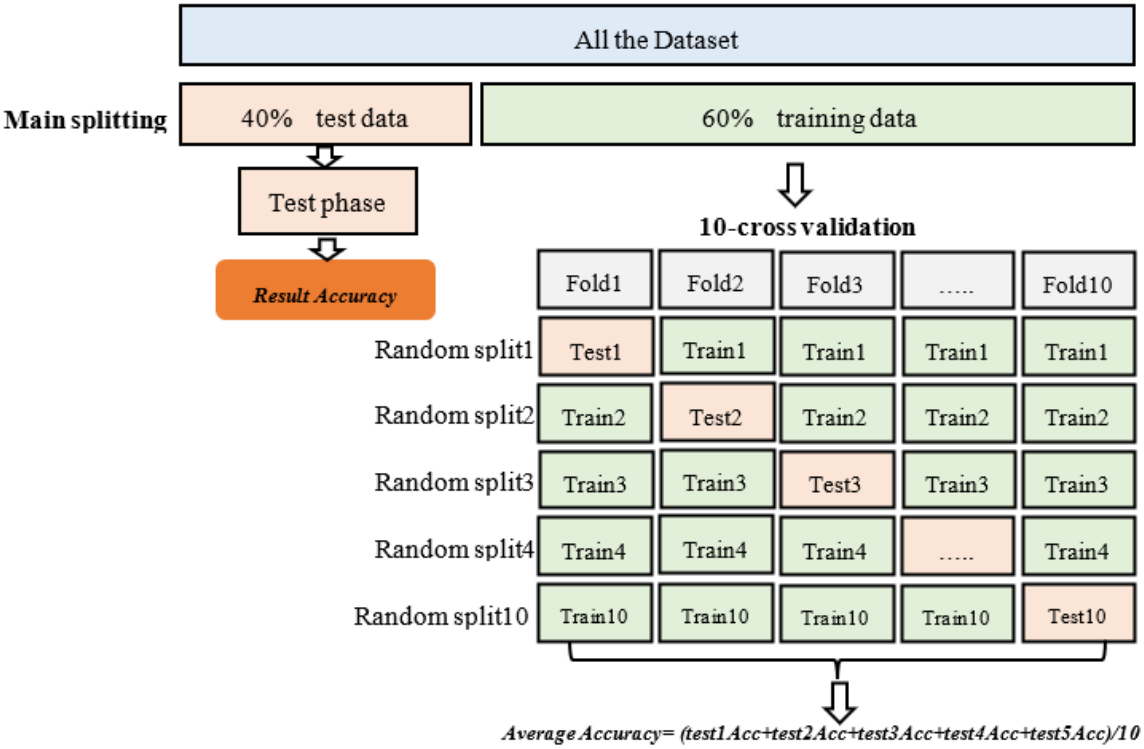

Figure 5. 10-Fold cross validation.

6.1. Selecting the K Value

The experimental outcomes of the suggested DIEBD-based feature extraction scheme with multiple K values of 0.5, 1.0, 1.5, and 2.0 are summarized in Tables 2–5, utilizing the measurement metrics that are explained in Section 5.3. By taking the highest results that are carried out by the XGboost classifier as a guideline, these tables show that the lowest detection performances are performed by the XGBoost classifier when K = 1.5 and 2.0, while the developed XGBoost has achieved the highest performance when k = 1. In particular, the tables reveal that the outcomes with k = 1.5 and k = 2.0 provided the poorest accuracy, $F1$, and detection rate, while k = 1 performs the greatest detection accuracy, $F1$, and detection rate. Similarly, the total value of FPR and FNR cases is increased to reach the highest values, 0.072 and 0.030, respectively, when K = 1.5, and decreased to 0.040 and 0.029 with k = 1. More specifically, the accuracy value started high at around 0.96 when K = 0.5 and K = 1.0, while the accuracy rate decreased when K = 1.5 and K = 2.0. However, the highest accuracy value is provided when k = 1.0. Likewise, the $F1$ value reached around 0.97 with K = 0.5 and K = 1.0 but decreased to about 0.96 when K = 1.5 and K = 2.0. Furthermore, the results of the experiments show that the proposed scheme carried out a similar detection rate with around 0.97 when K = 0.5, 1.0, and 1.5. Nevertheless, when K = 2.0, the detection rate is reduced to around 0.96. In connection with the precision, the highest precision of 0.978 is achieved when K = 1.0, whereas the minimum precision of 0.961 is provided with K = 1.5.

Table 2. Accuracy (*ACC*), Detection Ratel (*DR*), Precision (*P*), *F*-measure (*F*1), False positive rate (FPR), and False negative rate (FNR) for the proposed models with $k = 0.5$.

Classifier	Accuracy	Detection Rate	Precision	*F*1	FPR	FNR
KNN	0.915	0.924	0.945	0.934	0.101	0.076
CART	0.936	0.948	0.954	0.951	0.086	0.052
NB	0.759	0.671	0.945	0.785	0.074	0.329
SVM	0.952	0.964	0.962	0.963	0.071	0.036
ANN	0.962	0.970	0.972	0.971	0.053	0.030
RF	0.952	0.972	0.955	0.964	0.086	0.028
LR	0.929	0.954	0.939	0.946	0.118	0.046
XGBoost	0.962	0.974	0.967	0.971	0.062	0.026

Table 3. Accuracy (*ACC*), Detection rate (*DR*), Precision (*P*), *F*-measure (*F*1), False positive rate (FPR), and False negative rate (FNR) for the proposed models with $k = 1.0$.

Classifier	Accuracy	Detection Rate	Precision	*F*1	FPR	FNR
KNN	0.890	0.888	0.941	0.914	0.105	0.112
CART	0.932	0.937	0.958	0.947	0.078	0.063
NB	0.792	0.728	0.939	0.821	0.089	0.272
SVM	0.943	0.948	0.964	0.956	0.066	0.052
ANN	0.957	0.963	0.970	0.967	0.055	0.037
RF	0.947	0.962	0.957	0.959	0.082	0.038
LR	0.922	0.944	0.937	0.940	0.120	0.056
XGBoost	0.967	0.971	0.978	0.975	0.040	0.029

Table 4. Accuracy (*ACC*), Detection Rate (*DR*), Precision (*P*), *F*-measure (*F*1), False positive rate (FPR), and False negative rate (FNR) for the proposed models with $k = 1.5$.

Classifier	Accuracy	Detection Rate	Precision	*F*1	FPR	FNR
KNN	0.877	0.884	0.922	0.903	0.134	0.116
CART	0.924	0.944	0.938	0.941	0.112	0.056
NB	0.766	0.707	0.910	0.795	0.126	0.293
SVM	0.924	0.937	0.945	0.941	0.098	0.063
ANN	0.952	0.962	0.963	0.962	0.067	0.038
RF	0.932	0.961	0.935	0.948	0.120	0.039
LR	0.912	0.944	0.921	0.932	0.146	0.056
XGBoost	0.955	0.970	0.961	0.965	0.072	0.030

Table 5. Accuracy (*ACC*), Detection Rate (*DR*), Precision (*P*), *F*-measure (*F*1), False positive rate (FPR), and False negative rate (FNR) for the proposed models with $k = 2.0$.

Classifier	Accuracy	Detection Rate	Precision	*F*1	FPR	FNR
KNN	0.876	0.866	0.941	0.902	0.104	0.134
CART	0.916	0.928	0.944	0.936	0.106	0.072
NB	0.777	0.723	0.922	0.810	0.118	0.277
SVM	0.920	0.919	0.958	0.938	0.077	0.081
ANN	0.948	0.958	0.962	0.960	0.072	0.042
RF	0.929	0.949	0.944	0.946	0.109	0.051
LR	0.908	0.923	0.937	0.930	0.120	0.077
XGBoost	0.958	0.967	0.969	0.968	0.060	0.033

Figure 6 shows the performance comparison between KNN, CART, NB, SVM, ANN, RF, LR, and XGboost classifiers in terms of accuracy with various *K* values (0.5, 1.0, 1.5, and

2.0). It shows that the KNN, CART, SVM, ANN, RF, and LR classifiers presented higher accuracy with $K = 0.5$ than when $K = 1.0$. However, the highest accuracy of 0.967 is achieved by the XGboost classifier when $K = 1.0$. In contrast, the accuracy rates of the proposed classifiers are decreased when the K value is increased to 1.5 or 2.0. Figure 6 concludes that the most suitable K value to improve the detection accuracy is 1.0. This is because the assigned K value impacts the entropy values range between the lower and upper limits by modifying the lower and upper limit values. Furthermore, as long as the K value is bigger, the lower limit is decreased, and the upper limit is increased. Therefore, the chance of adding more features to the features set is increased due to the entropy values that are related to these new features being inside the range between the lower and upper limits.

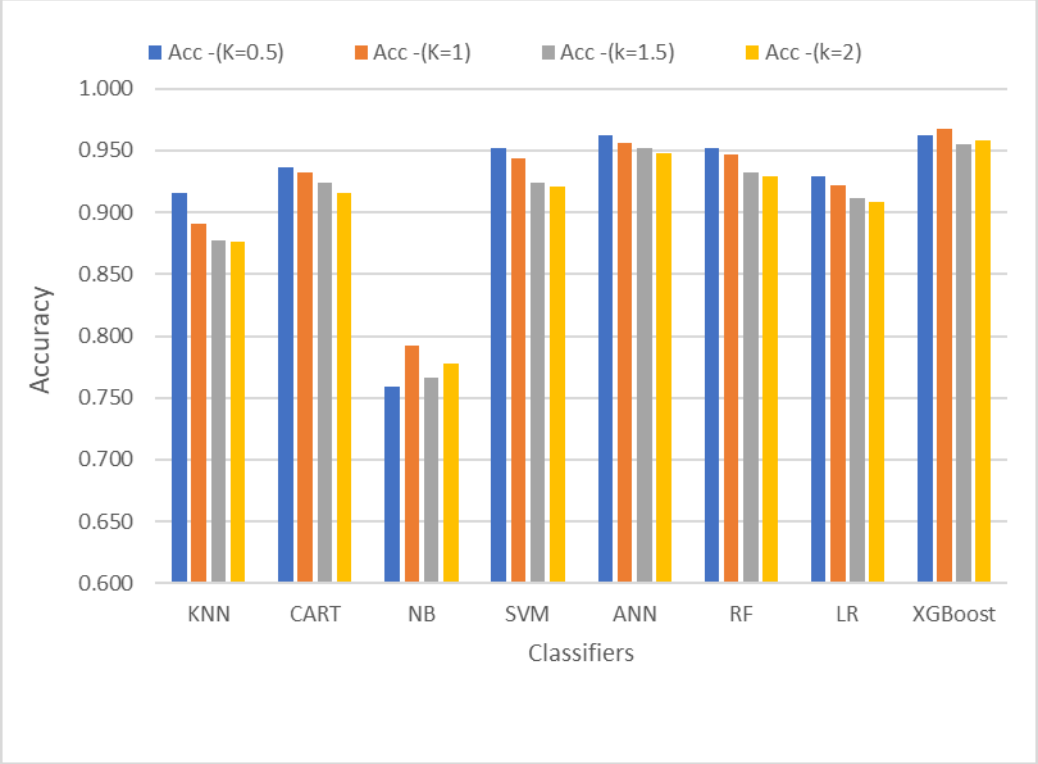

Figure 6. The comparison between the used classifiers in terms of the accuracy with various K values (0.5, 1.0, 1.5, and 2.0).

To the best of our knowledge, the proposed DIEBD-based feature extraction scheme transacts with the data offered by the evasive malware. Thus, this comparison derives that increasing the entropy values range between the lower and upper limits by raising the K value until $K = 1.0$ has improved the detection accuracy rate because the classifiers have been trained based on features by which the evasive behaviors are accurately represented since the dynamic break decision is made in the most appropriate change behavior point (timely break decision).

However, when $K = 1.5$ or 2.0, the entropy values of extra data have been included since the entropy values range between lower and upper limits accepts them. As a result, the entropy values range is more than enough; these extra data may not represent evasive behaviors. Otherwise, it can represent the alternative behaviors that appeared after the evasive malware already knew the nature of the environment. With $K = 1.5$ or 2.0, the correct change behavior point cannot be located accurately, leading to more non-discriminatory data. Therefore, unrepresentative features are extracted when a late decision is made (late-break decision).

6.2. The Comparison with Existing Work

We perform comparison experiments between our proposed scheme and state-of-the-art feature extraction methods using the several assessment metrics specified in Section 5.3. We employ machine learning techniques with API call-based *n*-gram features as our baseline methods. In [20,31,32], the features have been extracted from the whole collected data, which is exhibited by malware samples during the runtime without considering the impact of evasion techniques and the unrepresentative data that belong to the evasive malware when they recognize the analysis environment.

The authors in [31] collected the API calls that were invoked by malware and benign samples during the run time. Therefore, the *n*-gram technique was employed to extract the features from the API sequences. To represent the extracted features, the *TF-IDF* technique was utilized. Finally, machine learning classifiers were trained and evaluated using *TF-IDF*-based features. Ref. [20] used an occurrence times threshold of 500 to select the most frequent API call-based *n*-gram features. Ref. [31] extracted the frequent temporal patterns that were used to train and evaluate machine learning classifiers. Using our created dataset and our candidate machine learning algorithms, we implemented their methods to compare the performance of our proposed DIEBD-based feature extraction scheme with their feature extraction methods.

Figures 7–10 show the performance of machine learning techniques that are trained and evaluated based on the features extracted using the proposed scheme and the related work feature extraction schemes in terms of accuracy, *F*-measure, FPR, and FNR, respectively. Figures 7 and 8 show that most of the developed models, e.g., KNN, CART, SVM, ANN, RF, LF, and XGBoost, that are trained based on the features set extracted from the proposed scheme outperformed the accuracies and *F*-measures of related work feature extraction schemes. Even though the NB model that learned how to distinguish the evasive behaviors utilizing the [32] approach achieved higher accuracy and *F*-measure, the training phase consumed a long period of time due to the high dimensionality of the dataset.

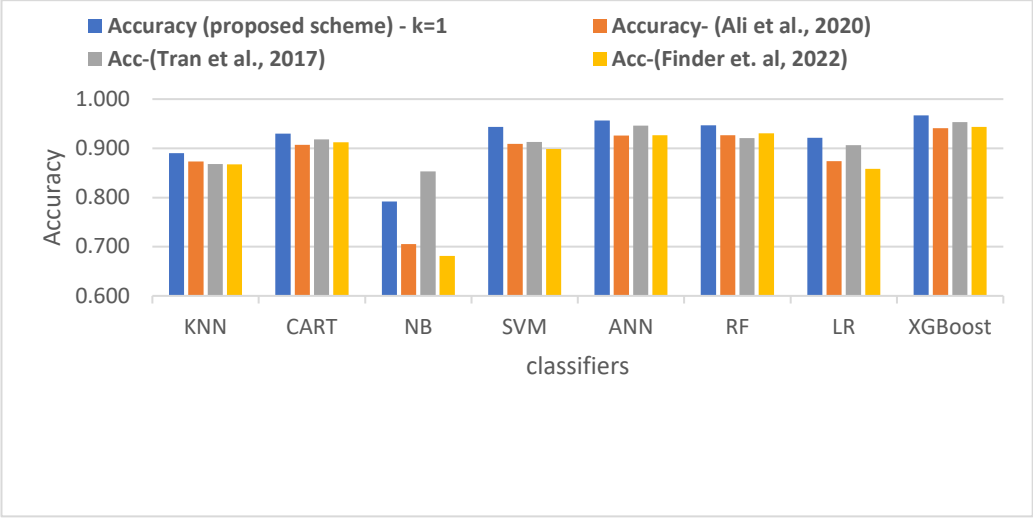

Figure 7. The comparison of the Detection Accuracy between the proposed DIEBD-based scheme and the related work-based schemes.

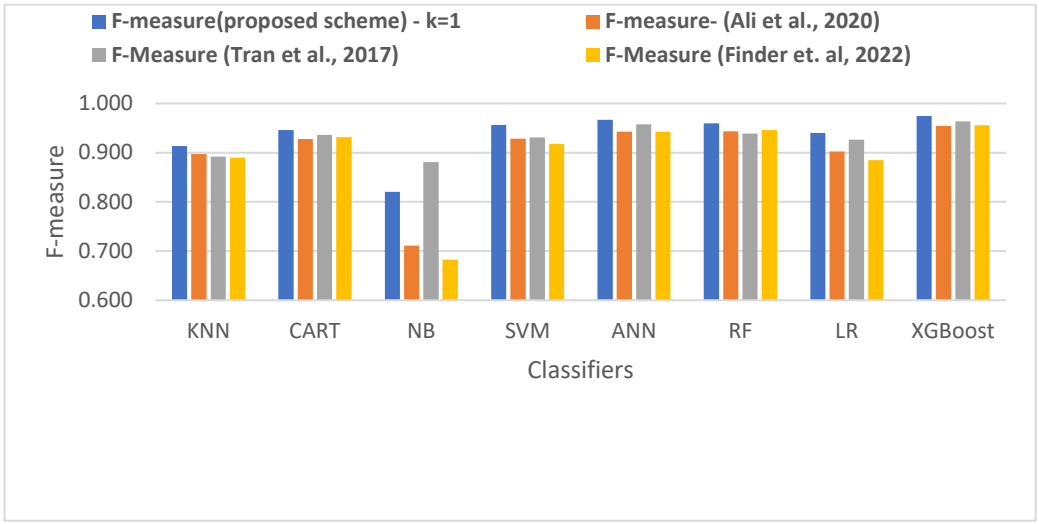

Figure 8. The comparison of the *F*-measure between the proposed DIEBD-based scheme and the related work-based schemes.

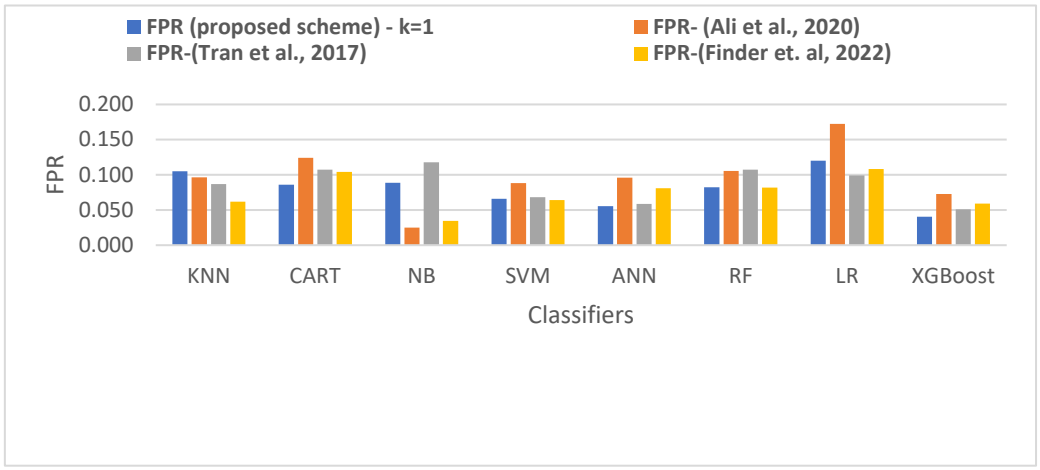

Figure 9. The comparison of the FPRs between the proposed DIEBD-based scheme and the related work-based schemes.

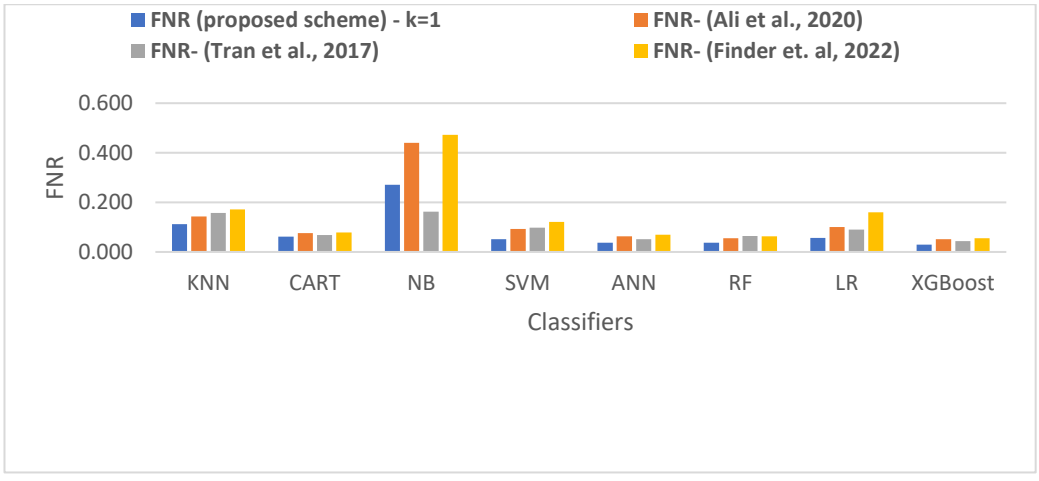

Figure 10. The comparison of the FNRs between the proposed DIEBD-based scheme and the related work-based schemes.

The results in Figures 7–10 imply that the proposed DIEBD technique could collect the evasive behaviors more carefully and avoid collecting the unrepresentative behaviors offered by such malware when they recognize that they are being executed in an analysis environment. This is because the proposed DIEBD measures the degree of the behavior change by comparing the entropy value of the current slide window in the windows sequence with the entropy value range of the previous slide windows to determine the behavior-changing window and avoid moving to the next slide window based on the fact that told, such malware change their behaviors when they realize that they are being analyzed.

Figures 9 and 10 illustrate that most of the proposed scheme-based models provide FPRs and FNRs lower than those provided by the models developed based on related work feature extraction schemes. XGBoost, ANN, and SVM achieved the lowest FPRs that ranged between 0.040 and 0.066. The comparison between FNRs of the proposed scheme-based models and the related work schemes-based models can be described as follows: The FNR of the proposed scheme-based models ranged between 0.029 for the XGBoost model and 0.272 for NB model, while the related work schemes-based models provided FNR ranged between 0.044 and 0.473. This suggests that the proposed DIEBD-based feature extraction scheme avoids extracting the alternative behaviors exhibited by such malware to mimic the legitimate behaviors when they know that they are executed in an analysis environment. Consequently, the proposed scheme-based models are prevented from learning those behaviors as malicious behaviors during the training phase, and thus the proposed scheme-based models discriminate them more accurately. However, the ability of the proposed scheme to distinguish between evasion behaviors achieved by malware for malicious purposes and evasion behaviors performed by benign for legitimate purposes (FPR) was worse than the ability of the proposed scheme to avoid learning the mimic legitimate behaviors performed by malware as malicious behaviors (FNR).

To the best of our knowledge, assigning the K value of 1.0 provides more ability for the proposed scheme to represent the evasion behavior characteristics of the evasive malware than the other K values. This is because the classifiers have been trained based on features by which the evasion behaviors are accurately represented. These behaviors are learned after separating them from the unrepresentative behaviors, which can be mimics of benign behaviors, noisy behaviors, or repeated behaviors that can be included when increasing the value of K. In contrast, gathering the entire set of behaviors provided by malicious software during execution time results in the development of ineffective models that learn to mimic legitimate behaviors as malicious activities since they are expressed by malware, and thus suffer from high FPR and low accuracy.

7. Conclusions

The advanced development in malware analysis has made the malware authors more serious about equipping the malware with forward defense mechanisms, such as evasion techniques, to avoid being analyzed and detected. In this paper, a feature extraction scheme based on dynamic initial evasion behaviors was proposed. Particularly, this paper concerned the feature extraction phase using the approach often neglected by the existing feature extraction schemes. In contrast to the previous solutions, the suggested feature extraction scheme identified the boundary of the representative evasion behaviors using the proposed dynamic initial evasion behavior determination (DIEBD) technique. Our feature extraction scheme used this technique to ensure that only the representative evasion behaviors were targeted to be collected by computing the entropy value of each window in the API sequence of each sample. Furthermore, the box-whisker plot algorithm was utilized to measure the entropy value variations to determine the change behavior point which was used as an indicator to avoid collecting unrepresentative data. The comparison between the proposed scheme and the existing work showed that the proposed scheme achieved higher classification performance than the existing work's feature extraction schemes. The results gained by the proposed scheme using the Xgboost classifier were 0.967, 0.040, 0.971, 0.978,

and 0.975 which outperformed the obtained results using the feature extraction scheme of the related work with Xgboost, i.e., 0.953, 0.051, 0.956, 0.972, and 0.964 for accuracy, false positive rate, detection rate, precision, and *F*1, respectively. As future work, this study needs to be extended to involve malicious software that is able to perform the evasion behaviors without calling API calls. In addition, the proposed scheme has a limitation of observing only partial evasion behaviors, and it is incapable to obtain the evasion behaviors when malware achieves them later. Another future work direction can be formed by further investigation to distinguish between evasion behaviors that are applied for legitimate and malicious purposes.

Author Contributions: Conceptualization, F.A.A. and A.Z.; methodology, F.A.G. and M.A.R.; validation, F.A.G.; analysis, F.A.A.; investigation, F.A.A. and A.Z.; software, F.A.A. and F.A.G.; resources, F.J.A. and A.M.A.; writing—original draft preparation, F.A.A.; writing—review and editing, A.Z., F.A.G. and A.M.A.; visualization, F.A.A.; supervision, A.Z., F.A.G. and M.A.R.; data curation, F.J.A. and A.M.A.; project administration, F.J.A. and A.M.A.; funding acquisition, A.M.A. and F.A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research work was funded by Institutional Fund Projects under grant no. (IFPIP-550-611-1442) from the Ministry of Education and King Abdulaziz University, DSR, Jeddah, Saudi Arabia.

Data Availability Statement: Not applicable.

Acknowledgments: This research work was funded by Institutional Fund Projects under grant no. (IFPIP-550-611-1442). Therefore, the authors gratefully acknowledge technical and financial support from the Ministry of Education and King Abdulaziz University, DSR, Jeddah, Saudi Arabia.

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

References

1. Singh, A.; Handa, A.; Kumar, N.; Shukla, S.K. Malware classification using image representation. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11527, pp. 75–92. [CrossRef]
2. Kaspersky Security. Kaspersky Security Bulletin. Available online: https://go.kaspersky.com/rs/802-IJN-240/images/KSB_statistics_2018_eng_final.pdf (accessed on 25 August 2022).
3. H. Sciences. Internet security Threat Report 2017. 2017. Available online: https://docs.broadcom.com/doc/istr-22-2017-en (accessed on 12 August 2022).
4. Morgan, S. Cybercrime Damages $6 Trillion By 2021. 2017. Available online: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/ (accessed on 18 August 2022).
5. Sahay, S.K.; Sharma, A.; Rathore, H. Evolution of Malware and Its Detection Techniques. In *Advances in Intelligent Systems and Computing*; Springer: Singapore, 2020; Volume 933, pp. 139–150. [CrossRef]
6. Jang, S.; Li, S.; Sung, Y. FastText-Based Local Feature Visualization Algorithm for Merged Image-Based Malware Classification Framework for Cyber Security and Cyber Defense. *Mathematics* **2020**, *8*, 460. [CrossRef]
7. Galloro, N.; Polino, M.; Carminati, M.; Continella, A.; Zanero, S. A Systematical and longitudinal study of evasive behaviors in windows malware. *Comput. Secur.* **2021**, *113*, 102550. [CrossRef]
8. Singh, J.; Singh, J. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Inf. Softw. Technol.* **2020**, *121*, 106273. [CrossRef]
9. Yoo, S.; Kim, S.; Kim, S.; Kang, B.B. AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification. *Inf. Sci.* **2020**, *546*, 420–435. [CrossRef]
10. Shijo, P.V.; Salim, A. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]
11. Darshan, S.L.S.; Jaidhar, C.D. Windows malware detection system based on LSVC recommended hybrid features. *J. Comput. Virol. Hacking Tech.* **2018**, *15*, 127–146. [CrossRef]
12. Sihwail, R.; Omar, K.; Ariffin, K.A.Z.; Al Afghani, S. Malware Detection Approach Based on Artifacts in Memory Image and Dynamic Analysis. *Appl. Sci.* **2019**, *9*, 3680. [CrossRef]
13. Mills, A.; Legg, P. Investigating Anti-Evasion Malware Triggers Using Automated Sandbox Reconfiguration Techniques. *J. Cybersecur. Priv.* **2020**, *1*, 19–39. [CrossRef]
14. Jha, S.; Prashar, D.; Long, H.V.; Taniar, D. Recurrent neural network for detecting malware. *Comput. Secur.* **2020**, *99*, 102037. [CrossRef]
15. Lin, W.-C.; Yeh, Y.-R. Efficient Malware Classification by Binary Sequences with One-Dimensional Convolutional Neural Networks. *Mathematics* **2022**, *10*, 608. [CrossRef]
16. Noor, M.; Abbas, H.; Bin Shahid, W. Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. *J. Netw. Comput. Appl.* **2018**, *103*, 249–261. [CrossRef]

17. Caviglione, L.; Choras, M.; Corona, I.; Janicki, A.; Mazurczyk, W.; Pawlicki, M.; Wasielewska, K. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access* **2020**, *9*, 5371–5396. [CrossRef]
18. Galal, H.S.; Mahdy, Y.B.; Atiea, M.A. Behavior-based features model for malware detection. *J. Comput. Virol. Hacking Tech.* **2015**, *12*, 59–67. [CrossRef]
19. Nunes, M.; Burnap, P.; Rana, O.; Reinecke, P.; Lloyd, K. Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis. *J. Inf. Secur. Appl.* **2019**, *48*, 102365. [CrossRef]
20. 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]
21. Catak, F.O.; Yazı, A.F.; Elezaj, O.; Ahmed, J. Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ Comput. Sci.* **2020**, *6*, e285. [CrossRef]
22. Zhang, J.; Gu, Z.; Jang, J.; Kirat, D.; Stoecklin, M.; Shu, X.; Huang, H. Scarecrow: Deactivating Evasive Malware via Its Own Evasive Logic. In Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Valencia, Spain, 29 June–2 July 2020; pp. 76–87. [CrossRef]
23. Branco, R.; Barbosa, G.N.; Drimel, P. Scientific But Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies. *J. Chem. Inf. Model.* **2012**, *53*, 1689–1699. [CrossRef]
24. Chen, P.; Huygens, C.; Desmet, L.; Joosen, W. Advanced or Not? A Comparative Study of the Use of Anti-Debugging and Anti-VM Techniques in Generic and Targeted Malware. In *IFIP Advances in Information and Communication Technology*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 471, pp. 323–336.
25. Ali, M.; Shiaeles, S.; Papadaki, M.; Ghita, B.V. Agent-based Vs Agent-less Sandbox for Dynamic Behavioral Analysis. In Proceedings of the 2018 Global Information Infrastructure and Networking Symposium, GIIS 2018, Thessaloniki, Greece, 23–25 October 2018. [CrossRef]
26. Alaeiyan, M.; Parsa, S.; Conti, M. Analysis and classification of context-based malware behavior. *Comput. Commun.* **2019**, *136*, 76–90. [CrossRef]
27. Kirat, D.; Vigna, G.; Kruegel, C.; Vigna, G.; Kruegel, C. BareCloud: Bare-Metal Analysis-Based Evasive Malware Detection. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014. Available online: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-kirat.pdf (accessed on 2 September 2022).
28. Banin, S.; Dyrkolbotn, G.O. Multinomial malware classification via low-level features. *Digit. Investig.* **2018**, *26*, S107–S117. [CrossRef]
29. Banin, S.; Shalaginov, A.; Franke, K. Memory access patterns for malware detection. *Nor. Nor. Inf.* **2016**, 96–107. Available online: http://hdl.handle.net/11250/2455297 (accessed on 5 September 2022).
30. Denzer, T.; Shalaginov, A.; Dyrkolbotn, G.O. Intelligent Windows Malware Type Detection based on Multiple Sources of Dynamic Characteristics. *Nis. J.* **2019**, *12*, 1–16.
31. Finder, I.; Sheetrit, E.; Nissim, N. Time-interval temporal patterns can beat and explain the malware. *Knowl.-Based Syst.* **2022**, *241*, 108266. [CrossRef]
32. 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. [CrossRef]
33. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-Rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware Detection Issues, Challenges, and Future Directions: A Survey. *Appl. Sci.* **2022**, *12*, 8482. [CrossRef]
34. Veerappan, C.S.; Keong, P.L.K.; Tang, Z.; Tan, F.; Veerappan, C.S.; Keong, P.L.K.; Tang, Z.; Tan, F. Taxonomy on malware evasion countermeasures techniques. In Proceedings of the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), Singapore, 5–8 February 2018; Volume 2018, pp. 558–563. [CrossRef]
35. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Comput. Surv.* **2019**, *52*, 1–48. [CrossRef]
36. Bulazel, A.; Yener, B. A survey on automated dynamic malware analysis evasion and counter-evasion: PC, Mobile, and Web. In *ACM International Conference Proceeding Series*; ACM: New York, NY, USA, 2017. [CrossRef]
37. Afianian, A.; Niksefat, S.; Sadeghiyan, B. Malware Dynamic Analysis Evasion Techniques: A Survey. *ACM Comput. Surv.* **2019**, *52*, 1–28. [CrossRef]
38. Lau, B.; Svajcer, V. Measuring virtual machine detection in malware using DSD tracer. *J. Comput. Virol.* **2008**, *6*, 181–195. [CrossRef]
39. Miramirkhani, N.; Appini, M.P.; Nikiforakis, N.; Polychronakis, M. Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts. In Proceedings of the IEEE Symposium on Security and Privacy, San Jose, CA, USA, 22–24 May 2017; pp. 1009–1024. [CrossRef]
40. O'Ree, A.; Obaidat, M.S. A dynamic malware analyzer against virtual machine aware malicious software. *Secur. Commun. Netw.* **2012**, *5*, 422–437.
41. Singh, J.; Singh, J. Challenges of Malware Analysis: Obfuscation Techniques. *Int. J. Inf. Secur. Sci.* **2018**, *7*, 100–110. Available online: http://www.ijiss.org (accessed on 16 September 2022).
42. Ehteshamifar, S.; Barresi, A.; Gross, T.R.; Pradel, M. Easy to Fool? Testing the Anti-Evasion Capabilities of PDF Malware Scanners. Available online: http://arxiv.org/abs/1901.05674 (accessed on 3 August 2022).
43. Küchler, A.; Mantovani, A.; Han, Y.; Bilge, L.; Balzarotti, D. Does Every Second Count? Time-Based Evolution of Malware Behavior in Sandboxes. In Proceedings of the 2021 Network and Distributed System Security Symposium, virtually, 21–25 February 2021. Available online: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_4C-5_24475_paper.pdf (accessed on 8 August 2022).

44. Kim, M.; Cho, H.; Hyun, J.; Member, Y.I. Large-Scale Analysis on Anti-Analysis Techniques in Real-World Malware. *IEEE Access* **2022**, *10*, 75802–75815. [CrossRef]
45. Zhou, J.; Hirose, M.; Kakizaki, Y.; Inomata, A. Evaluation to Classify Ransomware Variants Based on Correlations between APIs. In Proceedings of the 6th International Conference on Information Systems Security and Privacy, Valletta, Malta, 25–27 February 2020; pp. 465–472. [CrossRef]
46. 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. *Futur. Gener. Comput. Syst.* **2019**, *101*, 476–491. [CrossRef]
47. Pektaş, A.; Acarman, T. Classification of malware families based on runtime behaviors. *J. Inf. Secur. Appl.* **2017**, *37*, 91–100. [CrossRef]
48. Hwang, J.; Kim, J.; Lee, S.; Kim, K. Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques. *Wirel. Pers. Commun.* **2020**, *112*, 2597–2609. [CrossRef]
49. Du, D.; Sun, Y.; Ma, Y.; Xiao, F. A Novel Approach to Detect Malware Variants Based on Classified Behaviors. *IEEE Access* **2019**, *7*, 81770–81782. [CrossRef]
50. Oyama, Y. Trends of anti-analysis operations of malwares observed in API call logs. *J. Comput. Virol. Hacking Tech.* **2017**, *14*, 69–85. [CrossRef]
51. Oyama, Y. Investigation of the Diverse Sleep Behavior of Malware. *J. Inf. Process.* **2018**, *26*, 461–476. [CrossRef]
52. Ling, Y.T.; Sani, N.F.M.; Abdullah, M.T.; Hamid, N.A.W.A. Nonnegative matrix factorization and metamorphic malware detection. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 195–208. [CrossRef]
53. Pektaş, A.; Acarman, T. Malware classification based on API calls and behaviour analysis. *IET Inf. Secur.* **2018**, *12*, 107–117. [CrossRef]
54. Ghaleb, F.A.; Maarof, M.A.; Zainal, A.; Rassam, M.A.; Saeed, F.; Alsaedi, M. Context-aware data-centric misbehaviour detection scheme for vehicular ad hoc networks using sequential analysis of the temporal and spatial correlation of the consistency between the cooperative awareness messages. *Veh. Commun.* **2019**, *20*, 100186. [CrossRef]
55. Li, X.; Qiu, K.; Qian, C.; Zhao, G. An Adversarial Machine Learning Method Based on OpCode N-grams Feature in Malware Detection. In Proceedings of the 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC), Hong Kong, China, 27–30 July 2020; pp. 380–387. [CrossRef]
56. 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. *Futur. Gener. Comput. Syst.* **2018**, *90*, 211–221. [CrossRef]
57. Fuyong, Z.; Tiezhu, Z. Malware Detection and Classification Based on N-Grams Attribute Similarity. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 21–24 July 2017; Volume 1, pp. 793–796. [CrossRef]
58. Yang, L.; Liu, J. TuningMalconv: Malware Detection with Not Just Raw Bytes. *IEEE Access* **2020**, *8*, 140915–140922. [CrossRef]
59. Zhang, W.; Yoshida, T.; Tang, X. A comparative study of TF*IDF, LSI and multi-words for text classification. *Expert Syst. Appl.* **2011**, *38*, 2758–2765. [CrossRef]
60. Al-Hashmi, A.A.; Ghaleb, F.A.; Al-Marghilani, A.; Yahya, A.E.; Ebad, S.A.; Muhammad Saqib, M.S.; Darem, A.A. Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model. *IEEE Access* **2022**, *10*, 42762–42777. [CrossRef]
61. Chen, P.-H.; Zafar, H.; Galperin-Aizenberg, M.; Cook, T. Integrating Natural Language Processing and Machine Learning Algorithms to Categorize Oncologic Response in Radiology Reports. *J. Digit. Imaging* **2017**, *31*, 178–184. [CrossRef] [PubMed]
62. Zhang, J. Clement: Machine learning methods for malware recognition based on semantic behaviours. In Proceedings of the 2020 International Conference on Computer Information and Big Data Applications, CIBDA 2020, Guiyang, China, 17–19 April 2020; pp. 233–236. [CrossRef]
63. Kumar, H.; Chawla, N.; Mukhopadhyay, S. Towards Improving the Trustworthiness of Hardware based Malware Detector using Online Uncertainty Estimation. *arXiv* **2021**, arXiv:2103.11519. Available online: http://arxiv.org/abs/2103.11519 (accessed on 13 August 2022).
64. Chauhan, N.K.; Singh, K. A review on conventional machine learning vs deep learning. In Proceedings of the 2018 International Conference on Computing, Power and Communication Technologies, GUCON 2018, Greater Noida, India, 28–29 September 2018; pp. 347–352. [CrossRef]
65. Sun, Y.; Bashir, A.K.; Tariq, U.; Xiao, F. Effective malware detection scheme based on classified behavior graph in IIoT. *Ad. Hoc. Netw.* **2021**, *120*, 102558. [CrossRef]
66. Usman, N.; Usman, S.; Khan, F.; Jan, M.A.; Sajid, A.; Alazab, M.; Watters, P. Intelligent Dynamic Malware Detection using Machine Learning in IP Reputation for Forensics Data Analytics. *Futur. Gener. Comput. Syst.* **2021**, *118*, 124–141. [CrossRef]
67. García, D.E.; DeCastro-García, N. Optimal feature configuration for dynamic malware detection. *Comput. Secur.* **2021**, *105*, 102250. [CrossRef]
68. Revision, C.F. «Docs» Installation, Cuckoo Foundation Revision a665d2a6. 2022. Available online: https://cuckoo.readthedocs.io/en/latest/installati (accessed on 16 August 2022).
69. Kirat, D.; Vigna, G. MalGene: Automatic extraction of malware analysis evasion signature. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 769–780. [CrossRef]
70. Wei, C.; Li, Q.; Guo, D.; Meng, X. Toward Identifying APT Malware through API System Calls. *Secur. Commun. Netw.* **2021**, *2021*, 1–14. [CrossRef]

71. Darshan, S.L.S.; Jaidhar, C.D. An empirical study to estimate the stability of random forest classifier on the hybrid features recommended by filter based feature selection technique. *Int. J. Mach. Learn. Cybern.* **2019**, *11*, 339–358. [CrossRef]
72. Rostamy, A.A.A.; Khosroanjom, D.; Niknafs, A.; Rostamy, A.A. Fuzzy AHP models for the evaluation of IT capability, data quality, knowledge management systems implementation and data security dimensions. *Int. J. Oper. Res.* **2015**, *22*, 194. [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.