applied sciences

MDPI

Article

Contextual Identification of Windows Malware through Semantic Interpretation of API Call Sequence

Eslam Amer [1,2], Shaker El-Sappagh [3,4] and Jong Wan Hu [5,6,*]

1 Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, 17. Listopadu 2172/15, Ostrava-Poruba, 708 00 Ostrava, Czech Republic; eslam.amer@vsb.cz
2 Faculty of Computer Science, Misr International University, Cairo 11865, Egypt
3 Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain; Shaker.elsappagh@usc.es
4 Faculty of Computers and Artificial Intelligence, Benha University, Banha 13518, Egypt
5 Department of Civil and Environmental Engineering, Incheon National University, Incheon 22012, Korea
6 Incheon Disaster Prevention Research Center, Incheon National University, Incheon 22012, Korea
* Correspondence: jongp24@inu.ac.kr

Received: 27 September 2020; Accepted: 20 October 2020; Published: 30 October 2020

Abstract: The proper interpretation of the malware API call sequence plays a crucial role in identifying its malicious intent. Moreover, there is a necessity to characterize smart malware mimicry activities that resemble goodware programs. Those types of malware imply further challenges in recognizing their malicious activities. In this paper, we propose a standard and straightforward contextual behavioral models that characterize Windows malware and goodware. We relied on the word embedding to realize the contextual association that may occur between API functions in malware sequences. Our empirical results proved that there is a considerable distinction between malware and goodware call sequences. Based on that distinction, we propose a new method to detect malware that relies on the Markov chain. We also propose a heuristic method that identifies malware's mimicry activities by tracking the likelihood behavior of a given API call sequence. Experimental results showed that our proposed model outperforms other peer models that rely on API call sequences. Our model returns an average malware detection accuracy of 0.990, with a false positive rate of 0.010. Regarding malware mimicry, our model shows an average noteworthy accuracy of 0.993 in detecting false positives.

Keywords: malware detection; API call sequence; contextual behavior; malware mimicry

1. Introduction

With the rapid development in computers and Internet technology, malicious programs (malware) also have significantly developed in both categories and quantities. Researchers have centered their attention on inventing diversity malware detection methods to relieve the expeditiously growing malware rate. Generally, malware detection methods are categorized into either static or dynamic [1]. In static malware detection, researchers usually check and analyze portable executable (PE) files' contents without executing the malware samples.

Throughout the static analysis, analyzers investigated PE files by collecting and extracting specific features such as string patterns, operation code (op-code) sequences, and byte sequences. The features collected during static analysis are generally viewed as discriminating features that are used to decide whether a given sample is malicious or not [2]. Nevertheless, static malware detection methods have shown to be inappropriate to overcome the skillful techniques used by malware authors to bypass detection [3–5].

In contrast to static analysis, dynamic analysis tools are used to monitor the malware during execution. Through observing malware in real-time, we can extract valuable behavior features such as network behavior, system calls, registry change, and memory usage [6].

The Application Programming Interface (API) call sequences are viewed to be a distinguishable representative features in behavioral-based malware analysis [7]. The reason behind its prominence is because API call analysis can uncover and capture the malware behavior. Those types of real behaviors are not attainable in static analysis. Therefore, dynamic analysis research works relied on real-time features such as API call sequence as well as control flow that reveal malicious malware behavior [8]. However, dynamic analysis approaches are also insufficient. It was reported in [9] that brilliant malware can discover whether it runs on a virtual or real environment.

One of the most smart malware approaches to avoid exposure is through behaving as normal or benign executable files. This kind of mimicry behavior became a real challenge to malware detection tools. It is natural to think that the most common malware attacks (especially for Windows operating systems) are formed using executable files, however, security reports [10] showed that the wildest serious attacks are the ones that are carried out using mimicry infections. Those types of infections allow attackers to exploit the vulnerabilities of third-party applications to trigger executable payloads. Another quandary is regarded due to the vulnerabilities of third-party applications that are not promptly patched. Therefore, the late or absence of proper security updates increases much longer the lifespan of attacks committed by mimicry infections.

Machine learning-based techniques have been used to detect malicious parts that are embedded in infected user applications such as PDF files. Research work demonstrated the effectiveness of learning-based systems at detecting obfuscated attacks that are capable of circumventing plain heuristics [11–13]; however, the problem still requires significant work to resolve.

Malware analysis tools should also pay attention to non-executable files that seem to behave benignly. Nevertheless, they conceal malicious code which makes their detection significantly harder. Although their imperfection, dynamic analysis is prospectively able to conquest some benchmark metrics. Those metrics are determined during malware interactions with the subsidiary operating system. Those metrics can be used to detect a possible attack [14].

In this work, we exploited the contextual embedding features in the API call sequence. Through modeling the transitions existing in the calling sequence, we generated behavioral models for malware and goodware. Although malicious and non-malicious applications are using the same API functions, we proved that there are variations in how both types utilized the API functions. We also propose a solution to detect Windows malware and malware mimicry or *fake* goodware programs.

We organized the rest of the paper as follows: Section 2 discusses the related work and other research backgrounds. In Section 3, we present our proposed malware detection model. The datasets, along with the empirical evaluations of our model, are presented in Section 4. Section 5 concludes this paper.

2. Related Work

Many studies aimed to analyze malware characteristics. The most leading way to analyze malware is through monitoring its behavior. One of the leading approaches to perceive the program behavior is through tracking its API calls [15,16]. API functions are standard by themselves; there are no groups called malicious or non-malicious functions. Malicious applications also utilize the regular API functions to perform its harmful activities. The calling mechanism to API functions does not characterize the difference between malicious and normal programs. Although, the flow order of API calls may lead to the contextual behavioral characteristic of the calling process [17]. However, due to the vast amount of API functions, it becomes laborious to describe running processes' behavioral attributes by monitoring and tracing all APIs simultaneously.

The API calling sequence that takes place among the processes and the operating system is considered influential. Hence, it is viewed as a fundamental distinction between the behavior of

malicious and normal processes [3]. Therefore, most research work in malware analysis tried to understand the process behavior through analyzing API calls [18]. The order of functions in the calling sequences could lead to meaningful expressions that provide reliable malware recognition. The API calls encode sufficient information regarding the possible malware functionalities that happen throughout malware execution.

Popular machine learning algorithms such as Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Decision Tree (DT), and Naive Bayes (NB) are widely used in malware detection [19–22]. Conventional machine learning algorithms are potentially able to learn behavioral features from malware samples. However, the performance of any machine learning algorithm is determined by the accuracy of the extracted features. In addition, it is also troublesome to extract significant behavioral features to improve detection performance. Therefore, common machine learning algorithms seem discouraging for malware detection [23,24].

Lu et al. [25] and Wu et al. [26] converted API calls into regular expression (RE) rules to identify and extract malicious sequence patterns. They recognized any malicious sequence as malware when any match exists between the observed API call sequence and predefined RE rules. Taejin et al. [27] transformed API calls into some code arrangements and grouped the APIs using n-gram. Tran et al. [28] used natural language processing to analyze the API call sequence. They divided the long sequence calls into small chunks using approaches like n-gram. The resultant n-grams were assigned values by using the term frequency-inverse document frequency (TF-IDF).

The main objective of using TF-IDF is to transform the textual n-grams into numerical features to enable the application of machine learning algorithms. However, statistical approaches like TF-IDF do not conserve any contextual association that exists among words [29,30]. Consequently, in our work, we employed the word embedding on the API calling sequences to infer the contextual association among the API calls.

Despite the accuracy of machine learning-based models for malware detection, researchers getting more suspicions about the reliability of learning algorithms against malware mimicry attacks [31–34]. These types of attacks became quite popular, as shown in [31], which are subsequently discussed in [35–38]. They showed that mimicry attacks lead to deceiving malware detection models, which resulted in misleading classification.

Throughout this paper, we proposed a malware detection mechanism relying on the contextual perception among APIs within the calling sequence. We also addressed the mimicry behaviors that malware can have. The proposed work provided a reliable technique that detects with high accuracy, both malware and mimicry malware (fake goodware) calling sequences.

3. Proposed Model

As mentioned above, former research studies were mainly concerned with finding and extracting behavioral features' patterns in the API calling sequences. Behavioral patterns are used as features for identifying and detecting malware. However, previous studies did not attempt to investigate the association that may exist among the different API functions in the entire calling sequence.

In our proposed model, we aimed to discover any relation(s) that occur in benign or malicious calls. As shown in Figure 1, our model consists of three phases, namely, initialization, learning, and testing phase. We will briefly discuss each phase in the following sections.

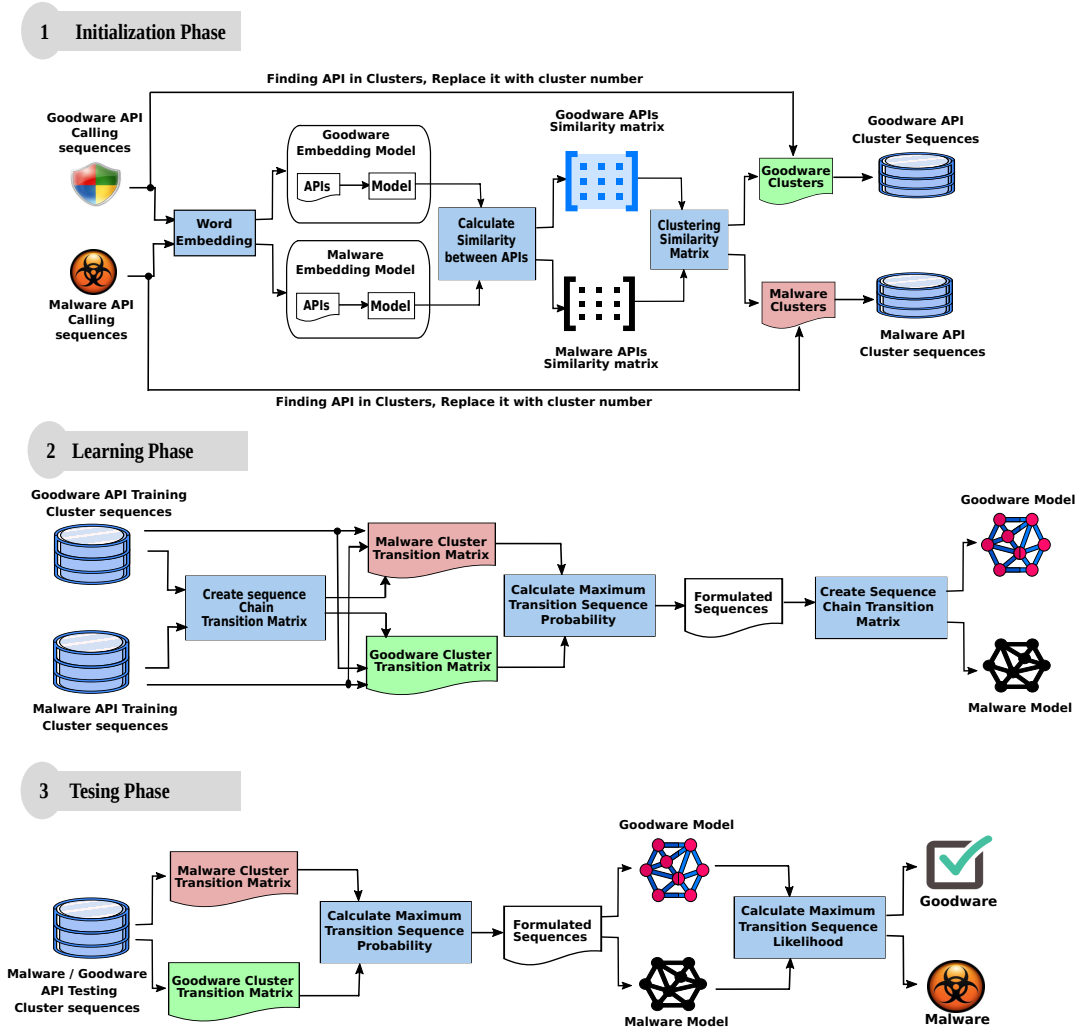


Figure 1. Proposed malware detection model.

3.1. Initialization Phase

The main purpose of the initialization phase is to restyle the API call sequence form to the cluster sequence one. A major obstacle we faced during malware analysis is the considerable amount of different API functions that make the analysis process extremely hard. However, the analysis process becomes possible if there is a way to customize that massive number of APIs.

Nevertheless, we think that the API functions' arrangements in the malware calling sequences do not exist at random. It conceals some remarkable contextual patterns which carry out their malignant activities. The contextual malicious patterns are relatively similar in some way among various malware sequences. Through extracting the contextual patterns from enormous malware API call sequences, we enhance our capability of characterizing the contextual relations which exist within malicious API call sequences.

Therefore, in our model, we relied on word embedding [39] to find the contextually related API functions. Analogous to word embedding, according to the API call sequences, the distribution of API function vectors in the space depends solely on the contextual similarity among APIs in the input corpus. Therefore, when two API functions are contextually similar, they will be positioned close to each other in the neighborhood space. Similarly, when two API functions are contextually dissimilar, they will be placed remotely from each other. During our experimentation, we set the embedding dimension feature vectors size to 300, window size to 8, and workers to 6. As shown in Figure 1, word embedding produces two outputs for each training sequences namely, *APIs* and *embedding model*.

In our model, we used the embedding model that resulted in each training category to calculate the similarity between its API function. The similarity computation produces two outputs, namely, goodware and malware API similarity matrix. The similarity matrix describes the similarity among individual API functions in its categorical sequence.

Through clustering the goodware/malware similarity matrix, we grouped different API functions that are contextually similar traits into a finite number of clusters. In our model, we used the k-means algorithm [40] to cluster the similarity matrix. We relied on the elbow method [41] to acquire the ideal number of k clusters to provide it to the k-means algorithm. In our experiments we obtained $k = 10$ as the optimal number of clusters for malware and goodware API calls.

As shown in Figure 1, for any API call sequence, we searched the resulted clusters for every API function in the sequence. When found, the function in the given sequence was be replaced by the cluster number that contains it. For example, according to the dataset introduced in [17], the following sample is an API sub-sequence of the malware sequence $Worm.Win32.Vobfus.agac$: *lstrcpyw, getthreadlocale, lstrcmpiw, globalalloc, globallock, globalunlock, globalrealloc, registerclipboardformatw, registerclipboardformata, getsystemdirectorya, isdbcsleadbyte, getversion, virtualallocex, getcommandlinea, getstartupinfoa.*

According to our model, each API in the previous API sequence will be searched against the clusters. The following representation denotes the cluster sequence that replaced the above API sequence:

1,1,1,1,4,4,1,6,4,7,8,1,1,1,1

The conversion of the original calling sequence into cluster sequence is considered the most pivotal step. Within a limited number of clusters, we got a perfect chance to restrict the sequence combination possibilities that malware can have. Therefore, malware analysis becomes possible.

3.2. Learning Phase

We can view the clusters generated in the initialization phase (Section 3.1) as a limited collection of states S where $S = \{S_1, S_2, S_3, S_4, \ldots, S_n\}$. According to our new representation for the calling sequence, the process, whether it is a malware or goodware, is expressed using a limited number of states called Markov states. A process normally begins at any state S_i, successively it may transit to a different state S_j as a subsequent action. According to the input sequence, the process can also change its state or wait in the same state. Therefore, the process is described through generated series of states $S_{i,1}, S_{i,2}, S_{i,3}, S_{i,4}, \ldots, S_{i,k}$. The movement series across various states are described as transitions between the different states. Our model relied on a first-order Markov chain to model transition sequences, where a state is completely counting on its former one. Therefore, a Markov model that has n states will ultimately have n^2 transition probabilities. These transition probabilities can be depicted as $n \times n$ matrix.

In our model, we used the maximum likelihood estimation (MLE) [42] to generate the transition likelihood probabilities, which describes the order of state transitions. Tables 1 and 2 are examples of an actual cluster transitions' matrices that were resulted from our experimentation on the dataset in [17]. Table 1 describes the transition probabilities that had arisen among the malware clusters' states, whereas Table 2 presents the transition' probabilities that were emerged among goodware clusters' states. Both malware and goodware clusters' transition matrices are regarded as the core of our model.

The transition sequence for any process becomes more reasonable when transforming the ambiguous cluster sequences into a meaningful form. The main motivation behind the reformulation is to unveil the behavioral transition of a given process. In another meaning, we require an explicit form to monitor and describe the malicious and the non-malicious likelihood behaviors for given malware and goodware sequences, respectively.

Table 1. Malware cluster transition matrix.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0.0756	0.4634	0.0363	0.0815	0.1668	0.1161	0.0232	0.0170	0.0019	0.0182
1	0.0638	0.6714	0.0538	0.0401	0.1028	0.0305	0.0209	0.0040	0.0081	0.0047
2	0.0566	0.5672	0.2203	0.0288	0.0432	0.0285	0.0468	0.0010	0.0073	0.0002
3	0.0349	0.5032	0.0007	0.3779	0.0662	0.0004	0.0164	0.0001	0.0000	0.0001
4	0.0770	0.3164	0.0118	0.0233	0.4936	0.0379	0.0180	0.0101	0.0108	0.0009
5	0.1846	0.2373	0.0058	0.0002	0.1060	0.4087	0.0498	0.0021	0.0001	0.0054
6	0.2632	0.0908	0.0007	0.0003	0.0138	0.2772	0.3121	0.0410	0.0000	0.0009
7	0.0026	0.7060	0.0035	0.0039	0.1022	0.0000	0.0001	0.0000	0.1816	0.0001
8	0.0383	0.8785	0.0054	0.0185	0.0534	0.0039	0.0000	0.0001	0.0000	0.0020
9	0.1271	0.1506	0.0027	0.0005	0.2247	0.0028	0.0001	0.0007	0.0007	0.4900

Table 2. Goodware cluster transition matrix.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0.6651	0.0025	0.0905	0.0047	0.0475	0.1657	0.0046	0.0035	0.0151	0.0008
1	0.0279	0.2257	0.0181	0.0000	0.0005	0.7268	0.0001	0.0004	0.0005	0.0000
2	0.2423	0.0006	0.3210	0.0054	0.0506	0.3484	0.0198	0.0004	0.0048	0.0068
3	0.2934	0.0025	0.1053	0.3580	0.0000	0.1190	0.0278	0.0000	0.0000	0.0941
4	0.3423	0.0009	0.1476	0.0000	0.4013	0.1073	0.0003	0.0000	0.0004	0.0000
5	0.0453	0.0439	0.0419	0.0011	0.0043	0.8591	0.0024	0.0000	0.0004	0.0015
6	0.1632	0.0039	0.2360	0.0943	0.0011	0.1840	0.2444	0.0021	0.0011	0.0697
7	0.3512	0.0000	0.0057	0.0312	0.0000	0.0099	0.0000	0.3494	0.2503	0.0025
8	0.3634	0.0060	0.0068	0.0491	0.0005	0.0215	0.0000	0.1691	0.3836	0.0002
9	0.0759	0.0000	0.1458	0.0363	0.0000	0.1587	0.5273	0.0004	0.0000	0.0557

Throughout our model, we relied on Equation (1) to achieve the required reformulations. According to Equation (1), the transition probability for a sequence (i, j) will have a value of *one* if its corresponding malicious probability is greater than its non-malicious one in cluster transition matrices. Otherwise, it will receive a value of *zero*.

$$sequence(i,j) = \begin{cases} 1 & p(Malware|(i,j)) > p(Goodware|(i,j)) \\ 0 & Otherwise \end{cases} . \quad (1)$$

where *(i,j)* is referring to the shifting of the sequencing process from state *i* to state *j*, $p(Malware|(i,j))$ and $p(Goodware|(i,j))$ are referring to the sequence transition in malware and goodware cluster transition probabilities, respectively.

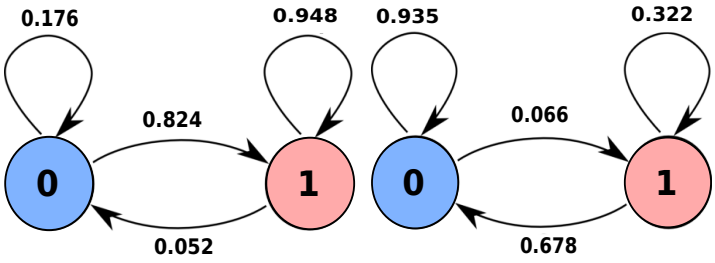
According to our model, the final classification for a transition, whether it is malicious or not, is depending on the maximum transition probabilities for the transition in malware and goodware cluster transition matrices. Accordingly, the transition is changed to *one* when it is malicious and *zero* otherwise. Hence, the whole calling sequence will be transformed into a new series of ones and zeros. For example, recall the generated cluster sequence which appeared in Section 3.1. Let us examine how our model determines whether it is malicious or not. Our model needs to determine the following transition probabilities that characterize the sequence:

p(1,1), p(1,1), p(1,1), p(1,4), p(4,4), p(4,1), p(1,6), p(6,4), p(4,7), p(7,8), p(8,1), p(1,1), p(1,1), p(1,1)

The probability of each transition will be fetched from malware and goodware cluster transition matrices in Tables 1 and 2, respectively. Table 3 showed the transition probabilities' tracing for the preceding sequences. Consequently, the formulation outcome of both malware and goodware transitions will be: 1 1 1 1 1 1 1 1 1 0 1 1 1 1.

Our proposed model used the newly formulated sequences to generate generic behavioral models that characterize malicious and non-malicious sequences. Once more, we used the maximum likelihood

estimation to generate transition models for malware and goodware. The learning phase finishes its work by producing two behavioral models: the malware and goodware models (Figure 2a,b, respectively).



(**a**) Malware Model (**b**) Goodware Model

Figure 2. Behavioral malware and goodware models.

Table 3. Formulating the transitions' probabilities for sequence 1,1,1,1,4,4,1,6,4,7,8,1,1,1,1.

Sequence Transition	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,4)**	**p(4,4)**	**p(4,1)**	**p(1,6)**	**p(6,4)**	**p(4,7)**	**p(7,8)**	**p(8,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**
p(sequence, Malware)	0.6714	0.6714	0.6714	0.1028	0.4936	0.3164	0.0209	0.0138	0.0101	0.1816	0.8785	0.6714	0.6714	0.6714
p(sequence, Goodware)	0.2257	0.2257	0.2257	0.0005	0.4013	0.0009	0.0001	0.0011	0.0000	0.2503	0.0060	0.2257	0.2257	0.2257
Formulation	1	1	1	1	1	1	1	1	1	0	1	1	1	1

3.3. Testing Phase

Generally, the intended purpose of the testing phase is to investigate the performance of our model in distinguishing newly sequences. Therefore, we provided our model with an unseen test set of malware and goodware sequences. As shown in Figure 1, the testing phase initially reformulates the input sequences as in the demonstration shown in Table 3.

We examined each sequence against both malware and goodware transition matrices. The model stores the transition probability when the sequence progresses from one state into another one. Our model relies on *maximum cumulative likelihood* of transition probabilities to determine whether a sequence is malicious or not. Accordingly, the formulated sequence that was generated for the example in Section 3.2 will be tested against malware and goodware models as shown in Table 4.

As depicted in Table 4, since the sequence accumulated likelihood for in the malware model is greater than its goodware, the sequence is regarded as *malicious*. Figure 3 shows a graphical illustration that describes how our model decides whether a sequence is malicious or not. We used some parts from the example in Section 3.2 for more clarification.

Table 4. Parsing of sequence (1 1 1 1 1 1 1 1 1 0 1 1 1 1) against malware and goodware models.

Input Sequence	**1 1 1 1 1 1 1 1 1 0 1 1 1 1**														**Likelihood Accumulation**
Sequence Transitions	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	**p(1,0)**	**p(0,1)**	**p(1,1)**	**p(1,1)**	**p(1,1)**	
Likelihood (Malware | Sequence)	0.948	0.948	0.948	0.948	0.948	0.948	0.948	0.948	0.948	0.052	0.824	0.948	0.948	0.948	12.252
Likelihood (Goodware | Sequence)	0.322	0.322	0.322	0.322	0.322	0.322	0.322	0.322	0.322	0.678	0.066	0.322	0.322	0.322	4.608

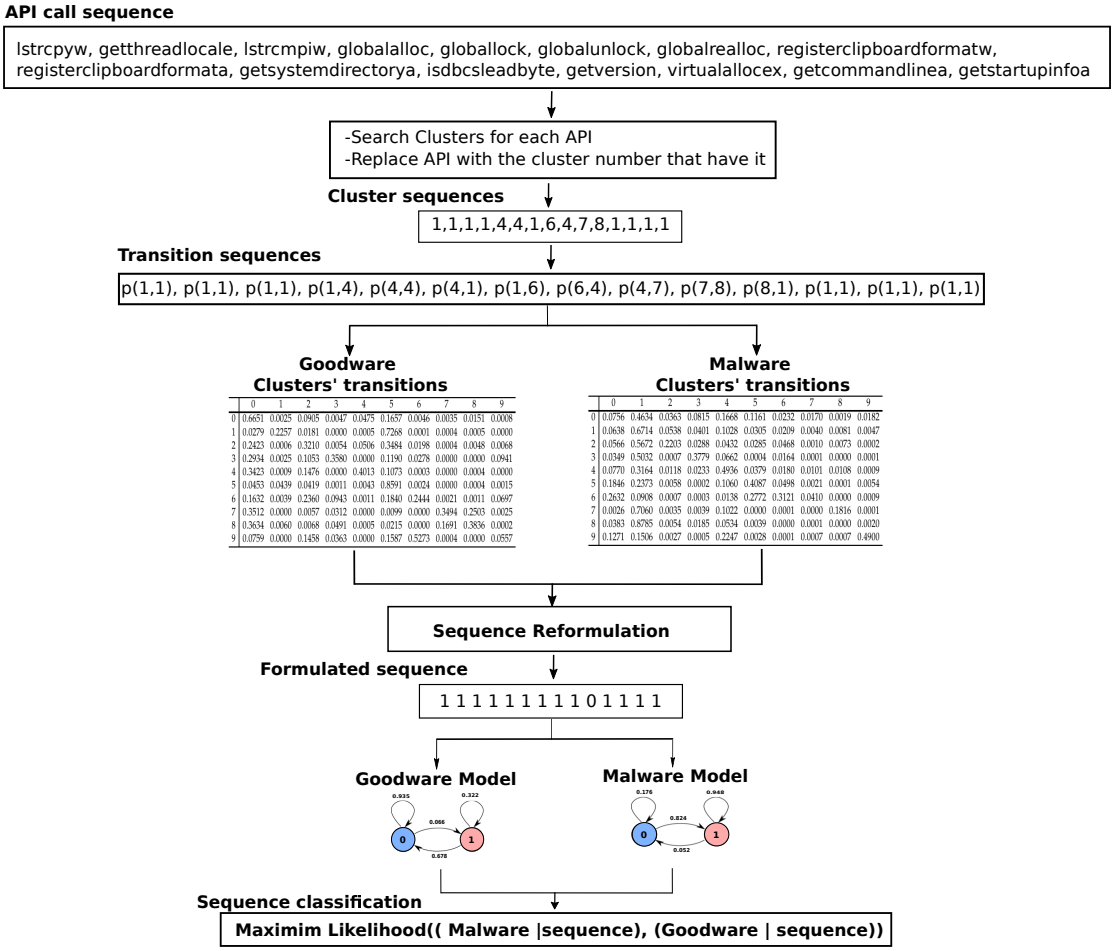


Figure 3. Graphical description for tracing a sequence.

4. Results and Discussion

Throughout this section, we evaluate our model through various datasets using standard evaluation metrics. We show that our model could efficiently recognize whether a sequence of API calls leads to malicious activities or not.

4.1. Datasets

To verify our model, we gathered varieties of API call sequences from [17,43,44]. We carried out our experiments with various datasets of different sizes to observe the efficiency of our model against the size of data.

4.2. Evaluation Metrics

Our model evaluation used well-known evaluation metrics such as precision, recall, F-measure, and accuracy. We also used other evaluation metrics inspired by the confusion matrix, such as false-positive rate (FPR) and false-negative rate (FNR). These measures assess the performance quality of the classification methods.

$$Precision = \frac{TruePositives(TP)}{TP + FalsePositives(FP)} \tag{2}$$

$$Recall = \frac{TP}{TP + FalseNegatives(FN)} \tag{3}$$

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{4}$$

$$Accuracy = \frac{TP + TrueNegatives(TN)}{TP + TN + FP + FN} \quad (5)$$

$$FalsePositiveRate = \frac{FP}{FP + TN} \quad (6)$$

$$FalseNegativeRate = \frac{FN}{FN + TP} \quad (7)$$

4.3. Malware Detection Evaluation

In our experimentation, we split our data into 50% for training and 50% for testing. Throughout the training process, we implemented a modified version of the k-fold strategy called the random subsamples (with replacement). The implemented model is slightly different from the k-fold in that, during each iteration, the selection of the training and testing samples are performed at random. The superiority of random subsamples (with replacement) comes from its elastically to determine the number of iterations and the size of training and testing samples. Our training samples were populated at random while maintaining a condition of eliminating any duplication for samples that may exist in the training or testing samples.

Our model avoided the training bias through performing our experiments 10 times for each dataset. We calculated the average returned results for all experiments per each dataset to be its final evaluation measure. Experimental results demonstrated high proficiency in detecting and discriminating unseen samples.

Our model has a high accuracy detection rate with tiny false positives. Table 5 shows that our method provides an average precision, recall, F-measure, and accuracy of 0.990, an average false-positive rate of 0.010, and an average false-negative rate of 0.010.

We experimented with our model against new unseen test samples to prove its validity and efficiency. The new samples contain 701 malware samples from https://github.com/duj12/cnn-lstm-based-malware-document-classification and 300 goodware samples from https://github.com/leocsato/detector_mw. According to the accuracy measures, as described in Table 6, our proposed model showed a considerable detection accuracy of 0.983, along with a false-positive rate of 0.034.

Table 5. Model detection evaluation.

Datasets	Accuracy Measures					
	Precision	**Recall**	**F-Measure**	**Accuracy**	**FPR**	**FNR**
Ki et al., 2015 [17]	0.999	0.998	0.999	0.999	0.001	0.001
Kim et al., 2018 [44]	0.994	0.986	0.990	0.990	0.006	0.014
CSDMC [43]	0.987	0.982	0.985	0.985	0.012	0.018
Catak et al., 2020 [45]	0.980	0.994	0.987	0.987	0.020	0.007
Average	**0.990**	**0.990**	**0.990**	**0.990**	**0.010**	**0.010**

Table 6. Model performance against unseen samples.

Datasets	Accuracy Measures					
	Precision	**Recall**	**F-Measure**	**Accuracy**	**FPR**	**FNR**
Testing Dataset	0.965	1.000	0.983	0.983	0.034	0.000

According to the malware detection accuracy measures, Table 7 showed that our proposed work outperformed other peer dynamic analysis approaches that used the API call sequence. We compared our results with different approaches to prove its competency. Our model showed an average accuracy of 0.999, which is considered the most trustworthy one compared to other approaches.

Table 7. Comparison with other works.

Study	# of Malware	F-Measure	Accuracy	Used Feature
Ahmed et al., 2009 [46]	416	-	0.980	API call sequence
Rieck et al., 2011 [47]	3133	0.950	-	API call sequence
Qiao et al., 2014 [6]	3131	0.909	-	API call sequence
Qiao et al., 2013 [48]	3131	0.947	-	API call sequence
Ki et al., 2015 [17]	23,080	0.999	0.998	API call sequence
Catak et al., 2020 [45]	7101	-	0.950	API call sequence
Proposed Work	23,080	0.999	0.999	API call sequence
	151	0.990	0.990	API call sequence
	320	0.985	0.985	API call sequence
	7107	0.987	0.987	API call sequences

4.4. Fake Goodware Detection

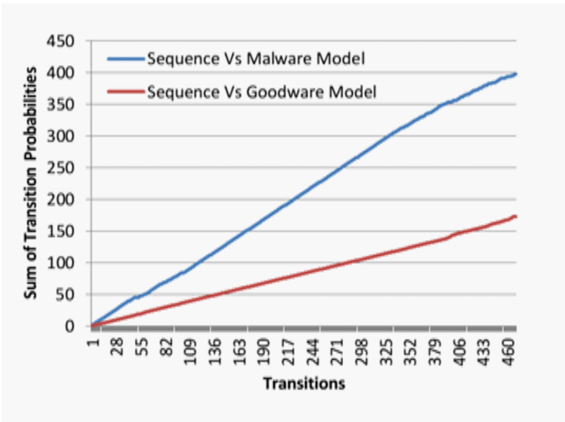
Despite our perceptible model accuracy in recognizing malware, there were particular types of malware samples falsely identified as goodware. When we investigated those kinds of examples, we discovered that malicious transitions are surrounded by many non-malicious ones. Therefore, those types of malware contain many non-malicious transitions compared to malicious ones. In other meaning, those kinds of malware samples are falsely acting as goodware ones. Our model identified these kinds of *mimicry malware* or *fake goodware* sequences through tracking their likelihood behavior.

Our experiments showed that most malware samples contain a majority of malicious transitions. However, we showed that malware transitions might also include partial non-malicious transitions, even if it does not affect its malicious collective likelihood behavior. However, in malware mimicry, we noticed that the API call sequence contains a significant amount of non-malicious transitions compared to malicious ones. In addition, we observed a *continually* changing behavior for those fake goodware samples during progressive transitions. Therefore, in our model, we used the *behavior inconsistency* as a sign, which indicates that a sequence is performing malicious activities.

To clarify our opinion, we provided sample sequences in Figures 4a and 5a. Both sequences are malware transition sequences. However, our model correctly recognizes the first sequence in Figure 4a as malware. In contrast, it misclassified the second one and identified it as a goodware. When we investigated the transition sequence in Figure 4a, we observed that it contains a plurality of malicious transition sub-sequences (transitions of contiguous 1s) in comparison to the non-malicious sub-sequences (transitions of contiguous 0 s). However, in the second sequence shown in Figure 5a, we found that it includes a plurality of non-malicious transition sub-sequences compared to the malicious sub-sequences.

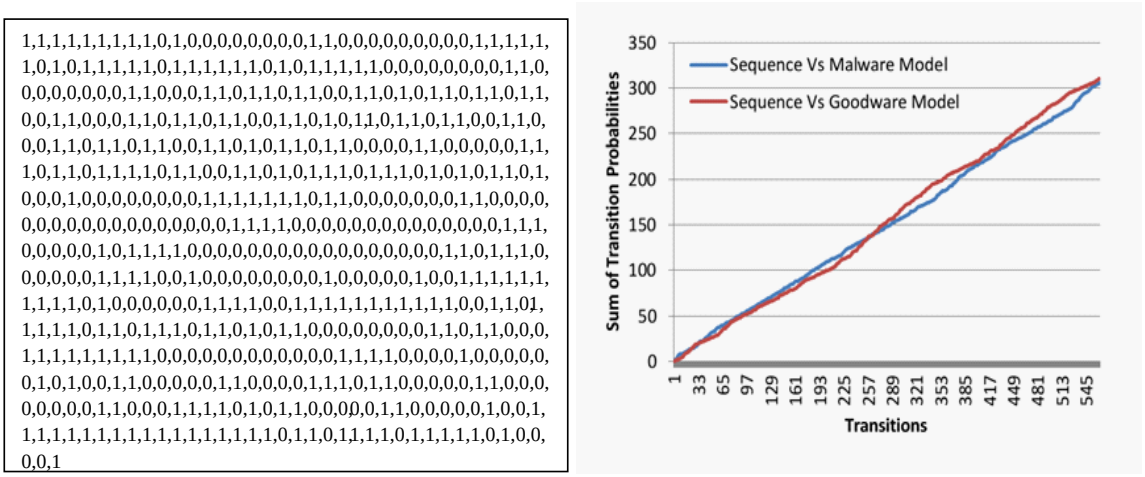
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0,1,0,1,1,1,0,0,1,1,1,0,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,11,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,0,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,0,1,1,0,0,0,0,0,0,1,0,0,1,1,1,1,
1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,1,1,1,0,1,1,1,1,1,0,1,1,1,0,0,
0,1,0,1,0,1,1,1,1,1,1,1,0,1,0,0,1,1,0,0,1,0,0,0,0,1,1,0

(**a**)



(**b**)

Figure 4. Real malware transition behavior. (**a**) Transition sequence for real malware; (**b**) cumulative evolutionary behavior likelihood for the sequence in Figure 4a.



(**a**) (**b**)

Figure 5. Mimicry malware transition behavior. (**a**) Transition sequence for fake malware; (**b**) cumulative evolutionary behavior likelihood for the sequence in Figure 5a.

When we inspected both sequences' behavior, as shown in Figures 4b and 5b against our malware and goodware models in Figure 2a,b, we noticed an exceptional distinction between both behaviors. Figures 4b and 5b explain the relationship between the successive transitions sequence (x-axis) and the associated collective evolutionary behavior likelihood (y-axis) for sequences in Figures 4a and 5a, respectively.

In Figure 4b, we observed that, although both malicious and non-malicious behaviors are growing, they are not scaling at the same rate. In another meaning, there is a continual separation gap between both behaviors during the progressive transitions. In contrast, the behaviors in Figure 5b converge and intersect at some progressive transition.

$$BM(S) = \frac{\sum_{i=1}^{n}(Malware \mid \sum p(T(1:i)) < (Goodware \mid \sum p(T(1:i))}{L(S)} \tag{8}$$

In our model, we utilized the *behavior monitoring* (BM) as a heuristic that identifies whether a sequence retains or modifies its behavior while being examined by our model. Equation (8) describes the behavioral intersection ratio where:

- S denotes the input sequence,
- n is the total number of transitions of a given sequence,
- $\sum p((T(1:i))$ refers to cumulative transition probabilities for the sequence up to the i-*th* transition in malware and goodware models,
- The exterior summation counts the events concerning the internal comparison between the two inner sums judged as *true*.

The behavior monitoring equation originally assumes that any given sequence is non-malicious until its behavior shows the opposite. Therefore, it continually tracks the sequence transitions' likelihood probabilities' in malware and goodware models simultaneously. When the sequence is malicious, as in the transition sequences in Figure 4a, then the *accumulated malicious likelihood* will be greater than its *accumulated non-malicious likelihood*. In other words, the differences between both behavioral likelihood accumulations in real malware will always be *positive*. However, in the case of fake goodware, as in the transition sequences in Figure 5a, the differences between both behavioral likelihood accumulations are inconsistent and tend to be *negative* during progressive transitions.

Our analysis concluded that a sequence is recognized as malicious if it has a cumulative changing behavioral ratio of 10% among its transitions. We examined our conclusion with malware false positives that emerged through our experiments in Table 5. As clarified in Table 8, our heuristic is

capable of identifying malware mimicry sequences and recognizing them as a possible malicious sequence with an average detection accuracy of 0.993. The high accuracy in detecting mimicry malware adds another reliability dimension to our model in identifying malware.

Along with monitoring the sequence behavior, there is a necessity for estimating the malicious degree of the sequence. Therefore, we have adjusted with a minor change the heuristic that monitors the sequence behavior to perform as a behavior confidence factor (BCF). Through assessing the sequence behavior, the sequence is also given a behavioral evaluation. Equation (9) describes how we evaluate the malicious ratio for a sequence where:

- the numerator denotes the number of times where the inner comparison is evaluated as *false*,
- $T(S)$ denotes the total number of transitions in the sequence S.

$$BCF(Malware|S) = \frac{\sum_{i=1}^{n}[(Malware \mid \sum p(T(1:i)) < (Goodware \mid \sum p(T(1:i))]}{T(S)} \quad (9)$$

To clarify the assessment evaluation process for the behavioral sequence, we also relied on the transition sequences in Figures 4a and 5a. Figure 4a contains 469 transitions, including (423 malicious transitions and 46 non-malicious transitions). According to Equation (9), the behavior confidence factor will be $\frac{423}{469} = 0.92$, which is interpreted as the sequence is malicious with a confidence factor of 92%. However, the transition sequence in Figure 5a contains 562 transitions, including (232 malicious transitions and 330 non-malicious transitions). Accordingly, the sequence is malicious with a confidence factor 41%.

Even with the indecisive confidence factor for the sequence in Figure 5a, the behavioral monitoring value is complementing the shortage that may occur when relying only on the behavioral confidence factor. Therefore, any sequence can be classified through the behavioral monitoring heuristic and assigned a confidence score through the confidence factor equation.

Table 8. Malware false positive detection evaluation.

Dataset	Accuracy
Ki et al., 2015 [17]	0.999
Kim et al., 2018 [44]	1.000
CSDMC [43]	1.000
Catak et al., 2020 [45]	0.973
Average	**0.993**

5. Conclusions

Throughout our paper, we proved that the contextual understanding of the API call sequence has enhanced malware detection accuracy. Our proposed model has employed word embedding to understand the latent contextual relations among individual APIs. We have created an API embedding model for Windows APIs. Through clustering APIs that are contextually related, our model has overcome the API tracking impossibility problem due to its huge number. Consequently, any API call sequence for a process could be represented using a finite number of cluster sequences. Our paper proposed generic behavioral models for malware and goodware. Experiments have proved the exceptional accuracy that our model returned. We have addressed and proposed a heuristic that detected mimicry malware sequence. The comparisons with peer approaches have proven that our empirical model is promising.

Author Contributions: E.A. and S.E.-S. formulated the inferred idea. They developed the theory and performed the computations. J.W.H. verified the analytical methods. E.A. and S.E.-S. aided in investigating the representational methods of malware and goodware while J.W.H. supervised the findings of this work. All authors reviewed the results and committed to the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No. 2020R1A4A4079299).

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

References

1. Zelinka, I.; Amer, E. An Ensemble-Based Malware Detection Model Using Minimum Feature Set. *Mendel* **2019**, *25*, 1–10. [CrossRef]
2. Gandotra, E.; Bansal, D.; Sofat, S. Malware analysis and classification: A survey. *J. Inf. Secur.* **2014**, *5*, 9. [CrossRef]
3. Amer, E.; Zelinka, I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* **2020**, *92*, 101760. [CrossRef]
4. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Venkatraman, S. Robust intelligent malware detection using deep learning. *IEEE Access* **2019**, *7*, 46717–46738. [CrossRef]
5. Yan, J.; Qi, Y.; Rao, Q. Detecting malware with an ensemble method based on deep neural network. *Secur. Commun. Netw.* **2018**, *2018*. [CrossRef]
6. Qiao, Y.; Yang, Y.; He, J.; Tang, C.; Liu, Z. CBM: free, automatic malware analysis framework using API call sequences. In *Knowledge Engineering and Management*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 225–236.
7. Galal, H.S.; Mahdy, Y.B.; Atiea, M.A. Behavior-based features model for malware detection. *J. Comput. Virol. Hacking Tech.* **2016**, *12*, 59–67. [CrossRef]
8. Zeng, N.; Wang, Z.; Zineddin, B.; Li, Y.; Du, M.; Xiao, L.; Liu, X.; Young, T. Image-based quantitative analysis of gold immunochromatographic strip via cellular neural network approach. *IEEE Trans. Med Imaging* **2014**, *33*, 1129–1136. [CrossRef] [PubMed]
9. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
10. Chumachenko, K. Machine Learning Methods for Malware Detection and Classification. Bachelor's Thesis, University of Applied Sciences, Kempten, Germany, 2017
11. Corona, I.; Maiorca, D.; Ariu, D.; Giacinto, G. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop, Scottsdale, AZ, USA, 7 November 2014; pp. 47–57.
12. Smutz, C.; Stavrou, A. Malicious PDF detection using metadata and structural features. In Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, FL, USA, 7–11 December 2012; pp. 239–248.
13. Šrndic, N.; Laskov, P. Detection of malicious pdf files based on hierarchical document structure. In Proceedings of the 20th Annual Network & Distributed System Security Symposium, San Diego, CA, USA, 24–27 February 2013; pp. 1–16.
14. Ranveer, S.; Hiray, S. Comparative analysis of feature extraction methods of malware detection. *Int. J. Comput. Appl.* **2015**, *120*. [CrossRef]
15. Chakkaravarthy, S.S.; Sangeetha, D.; Vaidehi, V. A Survey on malware analysis and mitigation techniques. *Comput. Sci. Rev.* **2019**, *32*, 1–23. [CrossRef]
16. Alaeiyan, M.; Parsa, S.; Conti, M. Analysis and classification of context-based malware behavior. *Comput. Commun.* **2019**, *136*, 76–90. [CrossRef]
17. Ki, Y.; Kim, E.; Kim, H.K. A novel approach to detect malware based on API call sequence analysis. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 659101. [CrossRef]
18. Zhao, Y.; Bo, B.; Feng, Y.; Xu, C.; Yu, B. A Feature Extraction Method of Hybrid Gram for Malicious Behavior Based on Machine Learning. *Secur. Commun. Netw.* **2019**, *2019*. [CrossRef]
19. Fan, M.; Liu, J.; Luo, X.; Chen, K.; Tian, Z.; Zheng, Q.; Liu, T. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1890–1905. [CrossRef]
20. Lin, Z.; Xiao, F.; Sun, Y.; Ma, Y.; Xing, C.C.; Huang, J. A Secure Encryption-Based Malware Detection System. *TIIS* **2018**, *12*, 1799–1818.

21. Fan, C.I.; Hsiao, H.W.; Chou, C.H.; Tseng, Y.F. Malware detection systems based on API log data mining. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung, Taiwan, 1–5 July 2015; Volume 3, pp. 255–260.
22. Li, Z.; Sun, L.; Yan, Q.; Srisa-an, W.; Chen, Z. Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Guangzhou, China, 10–12 October 2016; Springer: Cham, Switzerland, 2016; pp. 597–616.
23. Souri, A.; Hosseini, R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Hum.-Cent. Comput. Inf. Sci.* **2018**, *8*, 3. [CrossRef]
24. Ficco, M. Comparing API Call Sequence Algorithms for Malware Detection. In Proceedings of the Workshops of the International Conference on Advanced Information Networking and Applications, Caserta, Italy, 15–17 April 2020; Springer: Cham, Switzerland, 2020; pp. 847–856.
25. Lu, H.; Zhao, B.; Su, J.; Xie, P. Generating lightweight behavioral signature for malware detection in people-centric sensing. *Wirel. Pers. Commun.* **2014**, *75*, 1591–1609. [CrossRef]
26. Liu, W.; Ren, P.; Liu, K.; Duan, H.X. Behavior-based malware analysis and detection. In Proceedings of the 2011 First International Workshop on Complexity and Data Mining, Jiangsu, China, 24–28 September 2011; pp. 39–42.
27. Lee, T.; Choi, B.; Shin, Y.; Kwak, J. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. *J. Supercomput.* **2018**, *74*, 3489–3503. [CrossRef]
28. 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.
29. Amer, E. Enhancing efficiency of web search engines through ontology learning from unstructured information sources. In Proceedings of the 2015 IEEE International Conference on Information Reuse and Integration, San Francisco, CA, USA, 13–15 August 2015; pp. 542–549.
30. Youssif, A.A.; Ghalwash, A.Z.; Amer, E.A. HSWS: Enhancing efficiency of web search engine via semantic web. In Proceedings of the International Conference on Management of Emergent Digital EcoSystems, San Francisco, CA, USA, 21–24 November 2011; pp. 212–219.
31. Biggio, B.; Corona, I.; Maiorca, D.; Nelson, B.; Šrndić, N.; Laskov, P.; Giacinto, G.; Roli, F. Evasion attacks against machine learning at test time. In Proceedings of theJoint European Conference on Machine Learning and Knowledge Discovery in Databases, Prague, Czech Republic, 23–27 September 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 387–402 .
32. Biggio, B.; Corona, I.; Nelson, B.; Rubinstein, B.I.; Maiorca, D.; Fumera, G.; Giacinto, G.; Roli, F. Security evaluation of support vector machines in adversarial environments. In *Support Vector Machines Applications*; Springer: Cham, Switzerland, 2014; pp. 105–153.
33. Biggio, B.; Roli, F. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognit.* **2018**, *84*, 317–331. [CrossRef]
34. Brückner, M.; Kanzow, C.; Scheffer, T. Static prediction games for adversarial learning problems. *J. Mach. Learn. Res.* **2012**, *13*, 2617–2654.
35. Demontis, A.; Melis, M.; Biggio, B.; Maiorca, D.; Arp, D.; Rieck, K.; Corona, I.; Giacinto, G.; Roli, F. Yes, machine learning can be more secure! A case study on android malware detection. *IEEE Trans. Dependable Secur. Comput.* **2017**, *16*, 711–724 [CrossRef]
36. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*; Springer: Cham, Switzerland, 2017; pp. 62–79.
37. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; pp. 533–537.
38. Wang, Q.; Guo, W.; Zhang, K.; Ororbia, A.G.; Xing, X.; Liu, X.; Giles, C.L. Adversary resistant deep neural networks with an application to malware detection. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 23–27 August 2017; pp. 1145–1153.
39. Ketkar, N.; Santana, E. *Deep Learning with Python*; Apress: Berkeley, CA, USA, 2017; Volume 1.

40. Müller, A.C.; Guido, S. *Introduction to Machine Learning with Python: A Guide for Data Scientists*; O'Reilly Media, Inc.: Newton, MA, USA, 2016.
41. Syakur, M.A.; Khotimah, B.K.; Rochman, E.M.S.; Satoto, B.D. Integration K-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2018; Volume 336, p. 012017.
42. Andrews, B.; Davis, R.A.; Breidt, F.J. Maximum likelihood estimation for all-pass time series models. *J. Multivar. Anal.* **2006**, *97*, 1638–1659. [CrossRef]
43. Intelligence and Security Informatics Data Sets, BWorld Robot Control Software. Available online: http://www.azsecure-data.org/ (accessed on 5 July 2019).
44. Kim, C.W. NtMalDetect: a machine learning approach to malware detection using native API system calls. *arXiv* **2018**, arXiv:1802.05412.
45. 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]
46. Ahmed, F.; Hameed, H.; Shafiq, M.Z.; Farooq, M. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, Chicago, IL, USA, 9–13 November 2009; pp. 55–62.
47. Rieck, K.; Trinius, P.; Willems, C.; Holz, T. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.* **2011**, *19*, 639–668. [CrossRef]
48. Qiao, Y.; Yang, Y.; Ji, L.; He, J. Analyzing malware by abstracting the frequent itemsets in API call sequences. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, Australia, 16–18 July 2013; pp. 265–270.