Hindawi
Security and Communication Networks
Volume 2019, Article ID 1315047, 9 pages
https://doi.org/10.1155/2019/1315047

Research Article

An API Semantics-Aware Malware Detection Method Based on Deep Learning

Xin Ma, Shize Guo, Wei Bai, Jun Chen, Shiming Xia, and Zhisong Pan

Command and Control Engineering College, Army Engineering University of PLA, Nanjing 210007, China

Correspondence should be addressed to Zhisong Pan; hotpzs@hotmail.com

Received 19 February 2019; Revised 28 April 2019; Accepted 20 May 2019; Published 11 November 2019

Guest Editor: Wenjia Li

The explosive growth of malware variants poses a continuously and deeply evolving challenge to information security. Traditional malware detection methods require a lot of manpower. However, machine learning has played an important role on malware classification and detection, and it is easily spoofed by malware disguising to be benign software by employing self-protection techniques, which leads to poor performance for existing techniques based on the machine learning method. In this paper, we analyze the local maliciousness about malware and implement an anti-interference detection framework based on API fragments, which uses the LSTM model to classify API fragments and employs ensemble learning to determine the final result of the entire API sequence. We present our experimental results on Ali-Tianchi contest API databases. By comparing with the experiments of some common methods, it is proved that our method based on local maliciousness has better performance, which is a higher accuracy rate of 0.9734.

1. Introduction

Malware remains a serious problem for individuals, corporations, and nation information security all the time, as attackers continuously use it as a tool to get illegal profit and damage information infrastructure. Just in the first half of 2018 alone, the 360 Internet Security Center has intercepted 140 million new malicious programs, and an average of 795,000 new malicious programs was intercepted every day. We are seriously facing malicious code attacks all the time. Traditional signature-based feature detection methods, which take a lot of manpower and require professional knowledge, are difficult to combat. In fact, a lot of malware come from the benign software which was infected by malicious code snippets. Malware authors even use polymorphism to reorder these codes and create several malicious variations. By modification, this malware is hardly recognized by antivirus software. This is the key point that traditional detection technology for malware detection should be explored.

With the successful application of deep learning in the fields of image processing, speech recognition, and software engineering, machine learning has become an important method for analyzing malware in the last 10 years. As the Kaspersky Lab said, deep learning is a special machine learning approach that facilitates the features extraction method to get a high level of abstraction from low-level data. As the machine learning outperforms a great power for data handling, many scholars propose a series of methods using machine learning to detect malware. Previous studies [1, 2] used image visualization to classify malware by the similarity of texture. The studies [3, 4] are based on API call sequence to detect malware. In [5], the researcher uses LSTM and CNN to build a learning model for malware classification and detection, which transforms ASM code to sequence and uses NLP method to handle it. Above all, although these methods have made a stride for malware classification and detection, there are still many flaws, leading to attack from the malware writer, such as self-extraction and obfuscation. The key point is that these methods of extracting features are easily disturbed. Therefore, facing a lot of confrontation, it performs badly.

In this paper, we propose an effective detection framework based on API fragments, which can be employed

on the Windows platform by extracting the API execution sequence from executable files. Firstly, we analyze the local malicious characteristics of malicious code. On this basis, we cut the API execution sequence into API fragments and train the model using API fragments. Finally, we use ensemble learning to ultimately make a decision based on the performance of each segment of the entire sequence. The results show that this method can effectively extract key features and give a good classification result. Our work contributes as follows:

(1) Analyze the important meaning of local maliciousness in the API execution sequence

(2) Implement an API fragment-based detection framework with strong anti-interference using LSTM model and ensemble learning

(3) Expound the meaning of the parameters and obtain an optimal value

2. Related Work

The field of malicious code classification and detection is currently divided into traditional methods and machine learning methods. The traditional methods rely on a large amount of expert knowledge to extract the malicious features by reverse analyzing the binary code to achieve the purpose of classification and detection [6, 7]. Features extracted by manual analysis are highly accurate. However, this requires a considerable amount of manpower [8, 9]. As the malicious virus grows exponentially, the way of extracting features by manual analysis is becoming more and more expensive for this situation. Machine learning methods are highly generalized and do not require much manual work. Machine learning, because of its powerful learning ability, can learn some feature information that cannot be extracted manually. However, these methods based on machine learning are very susceptible to interference. Some existing methods, such as converting malicious code into pictures and signal frequency [2, 5], ignore the original semantics of the code and are easily interfered. As long as the malicious code author adds some byte information, or modifies the distribution of the file, the classifier can be confused. Venkatraman and Alazab [10] used the visualization of the similarity matrix to classify and detect zero-day malware. Visualization technology helps people to better understand the characteristics of malicious code, but they have not explored the application of deep learning.

In [3, 6, 11], the authors use the ASM file generated by disassembly to convert the assembly bytecode into pixel features and then use CNN to learn. Although this method takes advantage of some program information, malware authors can still make confusion by inserting external assembly instructions. Zhang et al. [12] use SVM to build a malicious code detection framework based on semi-supervised learning, which effectively solves the problem that malicious code is difficult to be marked on a large scale, and has achieved good results. There are also some methods that are based on API calls in [13]. They treat the file as a list containing only 0 or 1, with 0 and 1 representing whether or not the associated API appears. Their experiments show that the random forest classifier achieves the best result. This method mainly relies on the malicious API which could be emerged on a series of call sequence, and only the exact execution sequence can make damage on the computer system.

In [14], the authors construct behavior graphs to provide efficient information of malware behaviors using extracted API calls. The high-level features of the behavior graphs are then extracted using a neural network-Stacked AutoEncoders. On the one hand, their method of extracting behavioral graphs is very precise and helps to express the true meaning of the program fragments. On the other hand, their input vectors are constructed based on the whole sample, and the output of the model is the classification result of the whole sample. In fact, malicious fragments are only partial, which makes the malicious behavior graph easy to be overwhelmed.

Liu et al. [4] use image texture, opcode features, and API features to describe the sample files. By using the shared nearest neighbor (SNN) clustering algorithm, they obtained a good result in their dataset. Qian and Tang [15] analyze the API attributes and divide0 them into 16 categories. They propose a map color method based on categories and occurrence times for a unit time the API executed according to its categories. Then, they use the CNN model to build a classifier. Xiaofeng et al. [16] propose a new method based on information gain and removal of redundant API fragments, which effectively reduce the length of the API call sequence. The handled API call sequence is then entered into the LSTM model for training. Uppal et al. [17] use call grams and odds ratio to select the top-ranked feature segments, which are used to form feature vectors and are used to train the SVM model.

On the one hand, the above methods based on the API execution sequence are accurate, which reflect the dynamic execution information of the program. But on the other hand, due to program execution control, in a long execution sequence, the actual malicious execution code is very small or overwhelmed by a large amount of normal execution code. If the model does not learn the key malicious information, it will easily be bypassed by malicious code specifically disguised. There are also other machine learning methods to learn the features. Anderson and Roth [18] offer a public labeled benchmark dataset for training machine learning models to statically detect malicious PE files. They also construct baseline models based on gradient boosted decision tree algorithm. Even without any hyperparameter optimization, their work will still help researchers to study further in this field. In [19], the authors extract features based on the frequency of the API and compare neural networks with other traditional machine learning methods.

These methods expand the space for extracting malicious features, and improve the applicable scale of the machine learning method, and achieve good results. But they also have some limitations, mainly reflecting in the following aspects. First, manual methods have high

accuracy but require a lot of manpower, which makes them unsuitable for analyzing a large amount of malicious code. Second, machine learning is greatly influenced by the training set and its practicality is weak. For example, we have done an experiment, in which an image-based malware classifier can achieve 0.99 accuracy rate. However, after changing dataset, its performance drops sharply to about 0.73. Third, when the sample is confused, the training model is difficult to achieve good results, which reduces its practicability.

In fact, whether it is converted to images [20], signals, frequency, and other characteristics, it cannot truly express malicious code, nor does it conform to the traditional prior knowledge of malicious code. The method of extracting more efficient sequences by N-gram slicing [21, 22] only retains the sequential features of malicious code execution. The models trained with the features extracted by the common methods will have a poor effect.

Therefore, it is worth in-depth and long-term research to explore how to design a detection framework with the help of prior knowledge of malware, so that we can apply deep learning to malware detection better.

3. Our Method

From the perspective of information security, we analyze some malware and try to figure out the essential characteristics of them. After that, we confirm the local malicious characteristics of malware. Based on this, we propose a novel feature extraction method and build a detection framework based on the deep learning model.

3.1. Local Malicious Analysis. IDA Pro [23] is a powerful disassembler that can be used to disassemble binary so that by its disassembly result, we could get a sequence of API calls and analyze the behavior of the program. Through IDA Pro, we analyze a malicious code sample Trojan (VirusShare0a83777e95be86c5701aaba0d9531015 from VirusShare website [24]) as shown in Figure 1. As a result of the disassembly of IDA Pro, we can see that the Trojan's privilege operation is invoked by these six consecutive API function calls. The first function is *GetCurrentProcess*, which gets the handle of the current process by executing this function. The second function is *OpenProcessToken*, which can be used to open the current process with the handle value obtained by the first function. The third function is *LookupPrivilegeValueA*, which is used to view the permissions of the current process. The fourth function is *AdjustTokenPrivileges*, which is used to raise the permissions for the current process. The fifth function is *CloseHandle*, which closes the currently open process. The sixth function is *ExitWindowsEx*, which exits the current window. From the first function to the sixth function, this set of functions is used to elevate privilege of the current process, thus forming a malicious behavior. Because normal behavior does not take these operations to elevate its privilege, this is the difference between normal software and malware.

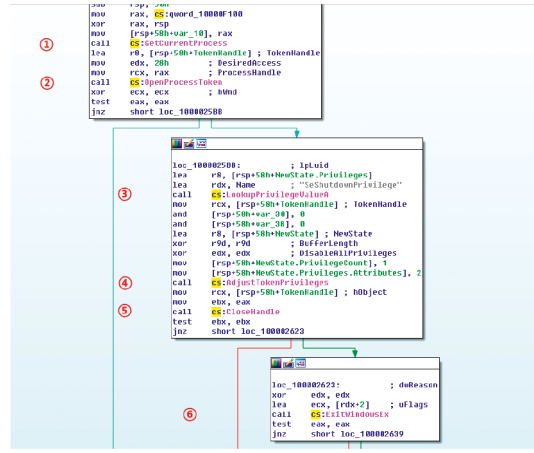

Figure 1: Trojan sample. The Figure is generated by IDA Pro. There are six different API calls marked by numbers from 1 to 6. This sequence indicates the API functions execution order.

In fact, the function of the program, whether normal or malicious, is mainly achieved through ordered API calls. Also, most of the malware is originated from benign software which is infected by malware such as viruses. From the code perspective, it is a normal code sequence that was inserted with malicious code snippets. In other words, most of the sequences of malicious code are normal, and only small segments are malicious. Malicious code also typically constructs itself by inserting normal code in this way. As shown in Figure 2, malicious code snippets are usually local and are only part of the overall code, which proves the local maliciousness of the malicious code we analyzed.

They often use the method of inserting useless APIs, conditional triggers, and confusion to counter detection. Traditional machine learning models that rely on the whole API call sequences as training data will be affected. Therefore, we propose a new extraction method based on local malicious features, and implement a framework which relies on local maliciousness to identify malicious code.

3.2. Detection System Framework. We first cut the entire API sequence into API fragments of length N. According to the local maliciousness, the API fragment of length N always retains the information of the malicious API fragment, and the API execution fragment constituting the maliciousness will always be acquired regardless of whether or not the source code is handled by confusion or deformation. Then, the tagged API fragments after de-duplication will be entered into the classifier for training, so that the classifier could detect the malicious API fragment. Furthermore, we use ensemble learning to identify malicious code by the proportion of malicious API fragments in the whole API execution sequence. These are the ideas of our entire framework.

Our framework can be divided into four parts: data processing, feature extracting, training model, and decision-making, as shown in Figure 3.

Data processing builds a word vector for all of API, and then transforms API sequences to a number list. Feature

Figure 2: The distribution of malicious code. The red parts represent malicious codes, which distribute in the whole code.

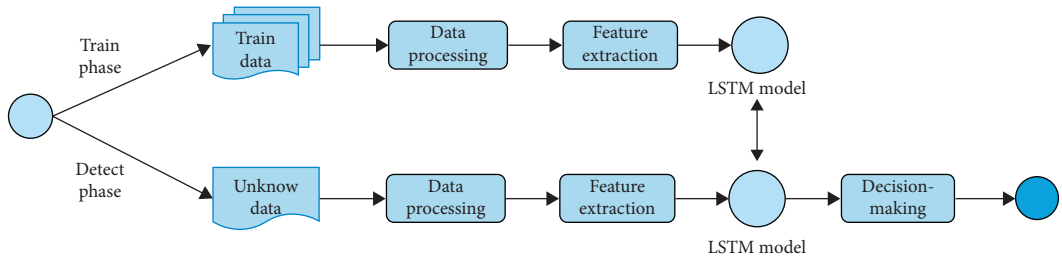

Figure 3: The framework of our method. The procedure consists of 4 steps. By data processing, the feature extracting, training model and decision-making, it gives a final classification result.

extracting uses sliding window operation to get slices with a fixed-length parameter N. We research different lengths for the sliding window to make the best result for slices. We use LSTM as our classifier to classify API fragments. Decision-making based on trained models makes a decision by voting on the basis of threshold M. Furthermore, we have a deep research on the parameters N and M. Based on this framework, our system is divided into the training phase and detecting phase, as shown in Figures 4 and 5.

Each process will be described in detail in the following sections.

(1) Data processing: in order to verify our hypothesis, we omit the process of API execution sequence extraction, which can be extracted by sandbox techniques. We use the public dataset directly, which consists of the API execution sequences that represent the executable files executing processes in the actual environment.

First, we use Word2vec [25] to build the API vocabulary by collecting all the API fragments that appear in the samples and use a number to uniquely represent an API. The samples consisting of API sequence are then replaced with a digital sequence for feature extraction, as shown in Figure 6.

(2) Feature extracting: we further analyze the malicious code and find that, in reality, in order to avoid detection, malicious code is usually inserted useless API function into the API call sequence. These malicious API calls only could be executed when the condition is triggered.

Based on the previous analysis of local maliciousness, we use the sliding window method to get the API fragment. We first set the length of the fixed sliding window to N and the step size to 1. Then, an API execution sequence with a length of M will form M − N + 1 API fragments by sliding the window. The operation diagram is shown in Figure 7.

Figure 7 above shows our sliding window approach, which is different from the N-gram method that is used in [26, 27]. The N-gram method will produce a large number of segments, and the overall position map of the different N-gram segments will not reflect the true meaning of a segment. Instead, the really useful segment information will be submerged.

After that, we label the API fragments generated by the sample with the same category tag. In fact, the API fragment is represented as a specific digital slice. By dividing the slices, these slice libraries with different labels are formed. Different from the method of information entropy adopted in [4], a de-duplication method is adopted by us to remove the same slices in the libraries by two to two.

(3) Model: the API call sequence reflects the malicious code execution order. This order is important. In the deep learning model, the RNN model has the function of remembering the sequence and is suitable for such data characteristics. Each hidden layer output of the RNN model is retained in the network so that it could combine with the next moment input of the system to determine the output of the next moment. The long and short-term memory LSTM model is an updated version of the RNN model that overcomes back-propagation gradient dispersion and gradient explosion and is more suitable for processing sequence data. Therefore, we choose LSTM as our training model. We design a three-layer LSTM as follow: each LSTM layer is connected to a dropout layer. Then, LSTM layer connects with each other. Finally, a dense layer is connected, which is used as the classifier.

The input for the LSTM model is digital slice which is formed in the feature extraction process. The model finally outputs the digital slice category.

(4) Decision-making: decision-making is designed for the purpose of classifying the unknown sample. In the training phase, we will train multiple models. In the detecting phase, because the models we trained are used to classify API fragments, we will make a decision on the unknown sample according to the digital slices which are formed by the unknown sample. Finally, classification results are determined if voting values of multiple models reach a certain

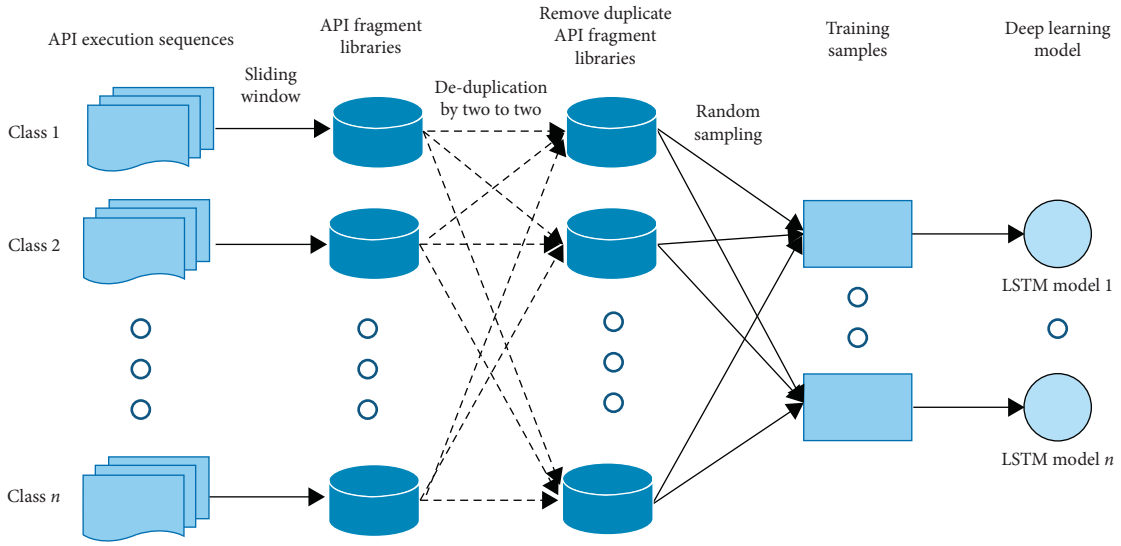

Figure 4: The training phase.

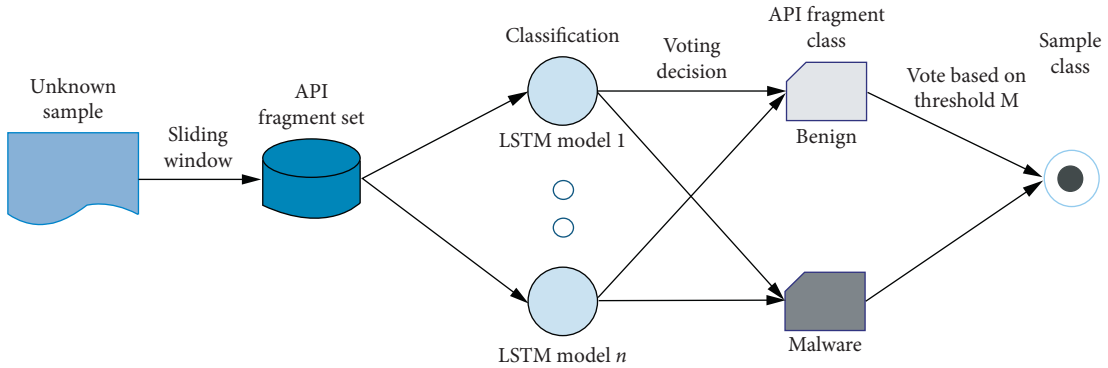

Figure 5: The detecting phase.

threshold M. We will set a threshold M and continuously optimize it during the experiment.

The details for decision-making are given as follows in Algorithm 1, which consists of SPLITWINDOW function and DECISIONMAKING function. The SPLITWINDOW function generates sets of removing duplicate API fragments. The DECISIONMAKING function makes final decision on unknown samples.

4. Experiment

4.1. Dataset. We use thedata of Alibaba-3rd-Security-Algorithm-Challenge [28] to test our method. The dataset is the API instruction sequence from the Windows executable program files in the sandbox simulation. All Windows binary executable program are desensitized. The sample data provided on this topic are collected from the Internet. Among them, the types of malicious files are the infected virus, Trojan horse program, mining program, DDoS Trojan, and extortion virus. By parsing the API data, there are a total of 12,000 samples. We randomly split the dataset into a training set and a test set. The sample distribution is shown in Figure 8.

4.2. Data Handling. In order to verify the anti-interference ability of the framework, we first randomly insert a number of invalid API functions into the API sequence for the purpose of confusion. Then, we use Word2vec to convert the API sequence into a sequence of numbers and use the sliding window to cut into the whole sequence, forming different libraries of fragments with different labels. After that, we use de-duplication method to remove redundant API fragments for each API slice library. In the training phase, because the number of malicious code files of different categories is very different, we adopt the bagging method by using random sampling with putting back to construct training sets. Then, each training set is used to train a model.

4.3. Result and Analysis. The accuracy, precision, recall, and F1-measure are selected as the evaluation indicators for the classification of samples and compared in comparison experiments, as shown below.

(1) Parameters N and M: the parameters N and M, respectively, represent the length of API slice and the threshold for judging the sample's category. We first verify the classification accuracy of API

GetSystemTimeASFileTime	1
NtAllocateVirtualMemory	11
NtFreeVirtualMemory	8
NtAllocateVirtualMemory	11
NtAllocateVirtualMemory	11
NtAllocateVirtualMemory	11
NtAllocateVirtualMemory	11
SetUnhandledExceptionFilter	13
LdrLoadD11	4
LdrLoadD11	4
LdrGetProcedureAddress	23
LdrUnLoadD11	5
NtCreateMut ant	25
NtCreateSection	41
NtMapViewof Section	78
LdrLoadD11	4
LdrGetProcedureAddress	29
LdrUnLoadD11	5
NtCreateMut ant	6
NtCreateSection	9
NtMapViewof Section	24
.....	.....
.....	.....

Figure 6: API sequence is converted to a digital sequence. Each API function corresponds to an unique number, and the entire API sequence is converted to a number for further processing.

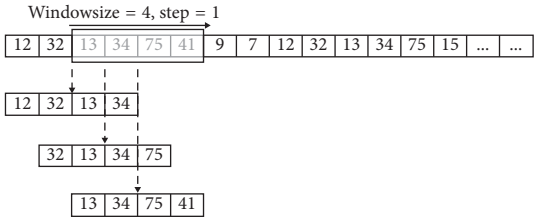

Figure 7: The operation of the sliding window. In the figure, the sliding window with window size by 4 moves rightwards with a step by 1. Each step of the window will produce a segment of length 4.

executable file for different parameters N and M. The results are shown in Figures 9–11.

As we can see from the above figures, although the length N is different, the threshold M and the accuracy are approximately subject to normal distribution. This also shows that we can definitely get an optimal value for the model. It is different from the study in [21], which determines an optimal N-gram length by the Cuckoo Algorithm [29].

Furthermore, these figures also show that, the shorter the length N, the smaller the threshold M for achieving the maximum accuracy. Conversely, the longer the length N, the larger the threshold M for achieving maximum accuracy. Furthermore, it is not difficult to infer that when the fragment length reaches the limit equal to the length of the entire sequence, the threshold at this time will be 1. It reflects the local maliciousness that we propose before, that is, the malicious fragments are only part of the entire sequence.

We then put these figures with different length parameters together, as shown in Figure 12.

It is obvious that the parameter N and M can get the best accuracy result. Also, all of the figures show the same trend that they are subject to normal distribution, which is consistent with the fact.

(2) Evaluation: we select the $N = 10$ and $M = 0.40$ as the best parameters, and evaluate different indications, that is, precision, recall, f1-score, and confusion matrix. The result is shown in Tables 1 and 2.

Table 1 shows the results of precision, recall, f1-score, and support for categories 0 and 1, respectively. Category 0 represents the normal code and category 1 represents the malicious code. It achieves high performance for classifying mission. Table 2 shows a confusion matrix for categories 0 and 1. By this, we can get the value of the true positive (TP), false positive (FP), false negative (FN), and true negative (TN). Accuracy is defined as follows:

$$\text{Accuracy (ACC)} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{FN} + \text{TN})}. \quad (1)$$

Finally, the accuracy is 0.9734, FPR is extremely 0, FNR is 0.0636, and the sensitivity is 0.9364. It proves that our method has an effective classification for malware.

4.4. Comparison. In order to verify the validity of our method, we design three comparative experiments. The first comparison test, without splitting the API call sequence, is directly converted into a digital sequence and sent to the same LSTM model for training. In the second comparative experiment, the same feature extraction method as the framework we proposed was used, and CNN was selected as the training model for evaluation. The CNN model is set as follows: we set up a three-layer convolution with 3, 4, and 5 as convolution kernel sizes, then it is connected to the max-pooling layer, further to a dropout layer, and finally to the fully connected layer. In the third comparative experiment, based on the same feature extraction method, SVM is selected as the training model for evaluation. The SVM model is set as follows: the penalty coefficient C is set to 10000. RBF is selected as the core, and its parameter gamma is set to 0.001, and the parameter degree is set to 3.

We choose accuracy as the evaluation criteria, and the results are shown in Figure 13. As can be seen from the results, the accuracy of our model is 0.9734, and the accuracy of using the entire API sequence is 0.8246. The accuracy of using the CNN model is 0.9325. The accuracy of using the SVM model is 0.9143.

Through comparative experiments, we can find that our method, the second, and the third methods are more accurate than the method based on the overall API execution sequence. It shows our method, that is the feature extraction method based on local maliciousness extraction API fragment, is effective. Because the sample has been confused, it also reflects that our feature extraction method has a degree of anti-interference ability. After adopting the same feature extraction method, by comparison, it can be found that the

Input: *sample*, *length* (the length of sample), *N* (the length of the window), *M* (threshold for voting), *C* (a set of all trained model for classification)
Output: *set* (store all API slices to be cut)
(1) **function** SplitWindow (*sample*, *length*, *N*)
(2) initial place in the beginning of the sample
(3) **repeat**
(4) split the sample with the solid window
(5) move the window with a step 1
(6) **until** move to the end of sample
(7) move all API slices into set
(8) Remove duplicates
(9) **return** *set*
(10) **end function**
(11)
Input: *set* (generated by Call SplitWindow ()), *M* (threshold for voting), *C* (a set of all trained model for classification)
Output: *category* (normal or malicious)
(12) **function** DECISION MAKING (*set*, *m*, *C*)
(13) **for** each $s \in$ set **do**
(14) **for** each $f \in C$ **do**
(15) $p = f(s)$
(16) **if** $p > 0.5$ **then**
(17) *s* is belong to normal slice
(18) **else**
(19) *s* is belong to malicious slice
(20) **end if**
(21) record the result for *s*
(22) **end for**
(23) **end for**
(24) $number = account(s_{malicious})$
(25) $total = account(s_{all})$
(26) **if** $number/total > m$ **then**
(27) **return** malicious
(28) **else**
(29) **return** normal
(30)

Algorithm 1: Classifying an unknown sample.

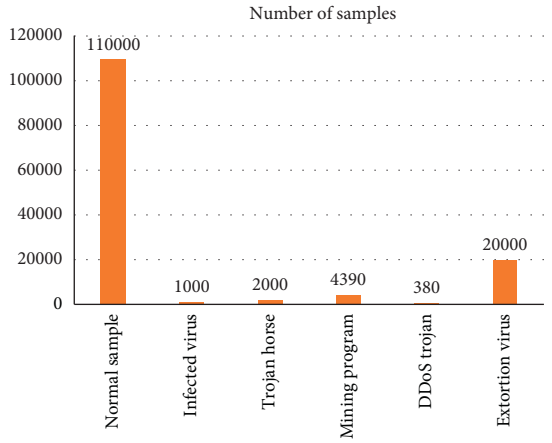

Figure 8: Data catagory numbers.

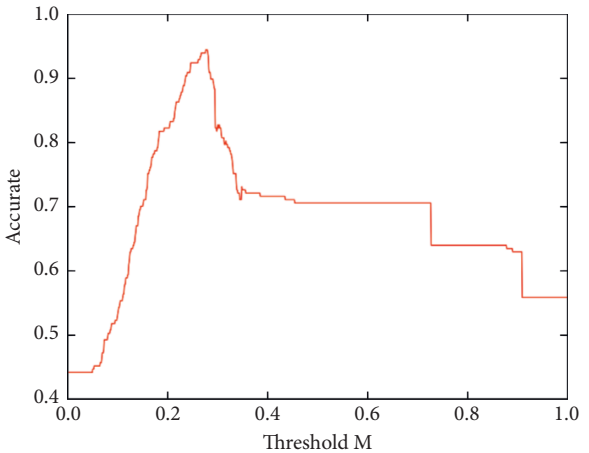

Figure 9: $N = 2$.

accuracy of the LSTM-based model is higher than that of the CNN and SVM models. Further analysis shows that our data are derived from API execution sequences with sequence characteristics. Our model is based on LSTM, which is very effective for processing data with sequential relationships. The CNN model is very effective for learning image features and is very effective for learning data with local features. SVM is a classic traditional machine learning model, but its learning ability is weaker than deep learning models such as

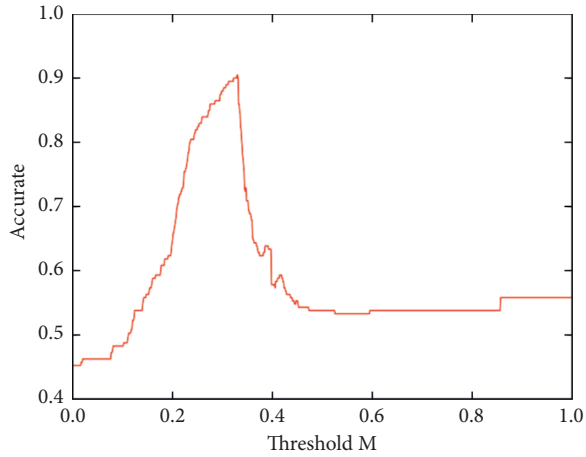

Figure 10: $N = 8$.

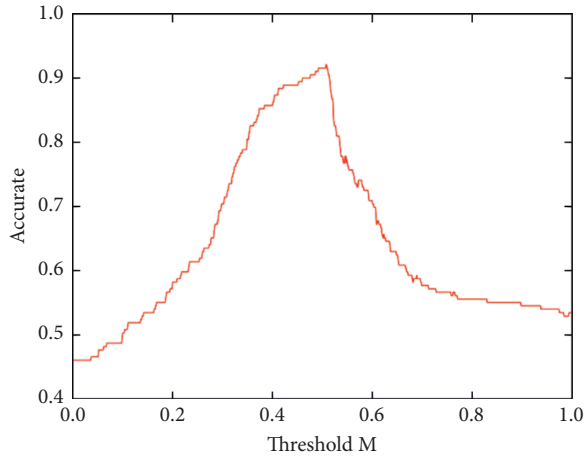

Figure 11: $N = 22$.

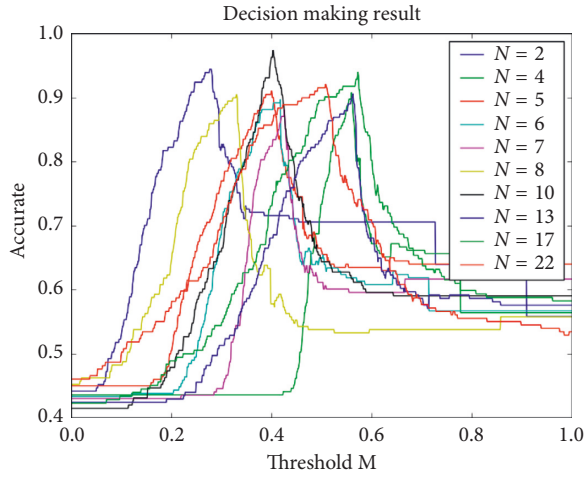

Figure 12: Different N.

LSTM and CNN. From the above analysis, we can see that our LSTM model is better for this kind of data with a sequential relationship. It is also the reason why the SVM is the lowest, CNN is the second, and our LSTM model is the highest.

Table 1: Classify report.

Classid	Precision	Recall	f1-score	Support
0	1.00	0.94	0.97	110
1	0.92	1.00	0.96	78
Avg/total	0.97	0.96	0.96	188

Table 2: Confusion matrix.

Classify	Benign	Malware
Benign	103	7
Malware	0	78

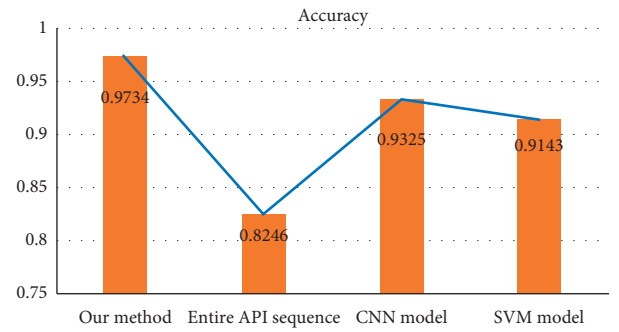

Figure 13: Comparative test.

As far as previous research is concerned, they usually did not delve into the local meaning of the API sequence. The general methods are to transform the API sequence as a whole and enter it into the model for training.

Meanwhile, the innovation of past methods about API sequence is focused on the elimination of redundant API functions or the construction of the corresponding conversion sequence [4, 15, 16]. But no matter how they reform, these ideas are still based on the overall API sequence. It makes their methods easily disturbed once malware developers take countermeasures. This inevitably leads to the lack of anti-interference for their model, which makes it difficult to have practical significance and to exert the powerful ability of deep learning.

In fact, the confused sample we constructed does not cover all the confrontation techniques, but it strongly proves the effectiveness of our extraction method based on local maliciousness and verifies our innovation.

5. Conclusion

We analyze the local maliciousness of malicious code, based on which we design a deep learning-based ensemble learning detection framework for API fragments. We use interference-handled samples for training and validation. The results show that our method can effectively resist interference. Our method also effectively explores the application of deep learning in the field of malicious code detection, which has a strong practical significance. In the future work, we will further study the combination of prior knowledge and deep learning.

Data Availability

Research data related to this article are deposited in the Ali-Tianchi contest dataset repository: Ali-Tianchi contest dataset (https://tianchi.aliyun.com/competition/entrance/231668/information).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the grants from the National Key Research and Development Program of China (Project no. 2017YFB0802800).

References

[1] K. Kosmidis and C. Kalloniatis, "Machine learning and images for malware detection and classification," in *Proceedings of the 21st Pan-Hellenic Conference on Informatics—PCI 2017*, p. 5, ACM, Larissa, Greece, September 2017.

[2] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the International Symposium on Visualization for Cyber Security*, pp. 1–7, ACM, Pittsburgh, PA, USA, July 2011.

[3] L. D. Vu Duc, *DEEPMAL: Deep Convolutional and Recurrent Neural Networks for Malware Classification*, 2018.

[4] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, no. 9, pp. 1336–1347, 2017.

[5] L. Nataraj, "A signal processing approach to malware analysis," Dissertations & theses—Gradworks, University of California, Santa Barbara, CA, USA, 2015.

[6] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pp. 32–46, IEEE, Oakland, CA, USA, May 2005.

[7] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *Proceedings of the USENIX Security Symposium*, vol. 13, p. 18, San Diego, CA, USA, August 2004.

[8] D. M. Chess and S. R. White, "An undetectable computer virus," in *Proceedings of the Virus Bulletin Conference*, vol. 5, pp. 1–4, Orlando, FL, USA, 2000.

[9] F. Cohen, "Computer viruses," *Computers & Security*, vol. 6, no. 1, pp. 22–35, 1987.

[10] S. Venkatraman and M. Alazab, "Use of data visualisation for zero-day malware detection," *Security and Communication Networks*, vol. 2018, Article ID 1728303, 13 pages, 2018.

[11] D. Gibert Llauradó, "Convolutional neural networks for malware classification," Thesis, Universitat Rovira I Virgili, Tarragona, Spain, 2016.

[12] K. Zhang, C. Li, Y. Wang, X. Zhu, and H. Wang, "Collaborative support vector machine for malware detection," *Procedia Computer Science*, vol. 108, pp. 1682–1691, 2017.

[13] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," in *Proceedings of the 2011 Ninth Australasian Data Mining Conference*, vol. 121, pp. 171–182, Australian Computer Society, Inc., Ballarat, Australia, December 2011.

[14] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, Article ID 8195395, 10 pages, 2019.

[15] Q. Qian and M. Tang, "Dynamic API call sequence visualization for malware classification," *IET Information Security*, vol. 13, no. 4, pp. 377–367, 2018.

[16] L. Xiaofeng, Z. Xiao, J. Fangshuo, Y. Shengwei, and S. Jing, "ASSCA: API based sequence and statistics features combined malware detection architecture," *Procedia Computer Science*, vol. 129, pp. 248–256, 2018.

[17] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of API sequences," in *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2337–2342, IEEE, Noida, India, September 2014.

[18] H. S. Anderson and P. Roth, "Ember: an open dataset for training static PE malware machine learning models," 2018, https://arxiv.org/abs/1804.04637.

[19] M. Alazab and S. Venkatraman, "Detecting malicious behaviour using supervised learning algorithms of the function calls," *International Journal of Electronic Security and Digital Forensics*, vol. 5, no. 2, pp. 90–109, 2013.

[20] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pp. 21–30, ACM, New York, NY, USA, August 2011.

[21] G. Bala Krishna, V. Radha, and K. V. G. Rao, "ELC-PPW: ensemble learning and classification (LC) by positional patterns weights (PPW) of API calls as dynamic *N*-grams for malware perception," *International Journal of Simulation–Systems, Science & Technology*, vol. 18, no. 1, 2017.

[22] C. Liangboonprakong and O. Sornil, "Classification of malware families based on *N*-grams sequential pattern features," in *Proceedings of the 2013 8th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 777–782, IEEE, Melbourne, Victoria, Australia, June 2013.

[23] IDA Pro, 2019, https://www.hex-rays.com/products/ida/.

[24] Virusshare Website, 2019, https://virusshare.com/.

[25] Word2vec, 2019, https://code.google.com/p/word2vec/.

[26] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici, "Improving malware detection by applying multi-inducer ensemble," *Computational Statistics & Data Analysis*, vol. 53, no. 4, pp. 1483–1494, 2009.

[27] Y. Ye, L. Tao, Q. Jiang, Z. Han, and L. Wan, "Intelligent file scoring system for malware detection from the gray list," in *Proceedings of the 15th ACM Sigkdd International Conference on Knowledge Discovery & Data Mining*, Paris, France, June 2009.

[28] Alitianchicontest, https://tianchi.aliyun.com/competition/introduction.htm?spm=5176.11409106.5678.1.4354684cI0fYC1&raceId=231668.

[29] Cuckoo Algorithm, 2019, https://en.wikipedia.org/wiki/Cuckoo_search.