Hindawi
Security and Communication Networks
Volume 2020, Article ID 6724513, 11 pages
https://doi.org/10.1155/2020/6724513

Research Article

SLAM: A Malware Detection Method Based on Sliding Local Attention Mechanism

Jun Chen,[1] **Shize Guo**,[1] **Xin Ma**,[1] **Haiying Li**,[2] **Jinhong Guo**,[2] **Ming Chen**,[2] **and Zhisong Pan**[1]

[1]*Command and Control Engineering College, Army Engineering University of PLA, Nanjing 210007, China*
[2]*University of Electronic Science and Technology of China, Chengdu 611731, China*

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

Received 31 December 2019; Revised 14 June 2020; Accepted 29 August 2020; Published 25 September 2020

Academic Editor: Yin Zhang

Since the number of malware is increasing rapidly, it continuously poses a risk to the field of network security. Attention mechanism has made great progress in the field of natural language processing. At the same time, there are many research studies based on malicious code API, which is also like semantic information. It is a worthy study to apply attention mechanism to API semantics. In this paper, we firstly study the characters of the API execution sequence and classify them into 17 categories. Secondly, we propose a novel feature extraction method based on API execution sequence according to its semantics and structure information. Thirdly, based on the API data characteristics and attention mechanism features, we construct a detection framework SLAM based on local attention mechanism and sliding window method. Experiments show that our model achieves a better performance, which is a higher accuracy of 0.9723.

1. Introduction

The rapid development in computers and Internet technology is also coupled with rapid growth in malicious software (malware). Malware such as viruses, Trojans, and worms also changed expeditiously and became the most severe threat to the cyberspace. Malware is usually installed and operated on a user's computer or other terminal without user's permission, which infringes on the legitimate rights and interests of users. It gains control over computer systems through changing or malfunctioning normal process execution flow.

According to the latest China Internet Security Report 2018 (Personal Security Chapter) released by 360 Security in April 2019, 360 Internet Security Center intercepted 270 million new malicious program samples on PC in 2018, with an average of 752,000 new malicious program samples on PC everyday [1]. In addition, malware often uses confusion, encryption, deformation, and other technologies to disguise itself in order to avoid being detected by antivirus software. Such a large number of malware and complex countermeasure technologies have brought serious challenges to network security.

To face these challenges, researchers conduct a series of studies. They use static analysis, dynamic analysis, and hybrid analysis for executable files, and extract a series of features, which includes Opcodes, API calls, and binaries. After that, they take machine learning to construct the detection model and achieve good results. However, in reality, deep learning in machine learning area is especially worth focusing on, due to its powerful expression ability.

Up to now, malware detection methods based on deep learning mainly focus on image [2], signal [3], and Application Programming Interface (API) sequence [4]. It may not stimulate the potential ability of deep learning model if we just simply transform malware into an input vector. How to effectively use expert knowledge to process data, transform it into the input needed by deep learning model, and design a specific deep learning model are the key to improve the effectiveness of deep learning model in detecting malware area.

We have noticed that, in the field of machine learning, the attention mechanism has been used very successfully, especially in the fields of Natural Language Processing (NLP), image, and machine Q and A. Attention mechanism has significantly improved performance, which demonstrates the powerful ability of deep learning in solving practical problems. For example, the latest XLNet model [5] builds a content-based and context-based attention mechanism by using a two-stream attention mechanism. However, the seq2seq problem and the malware classification are still different. How to effectively transfer the attention mechanism originated from translation problems to the field of malware classification according to practical problems is a subject worth exploring.

In this paper, we firstly analyze the attributes of the APIs and further divide them into 17 categories. Based on the category, we construct semantic and structure-based feature sequences for API execution sequences. Then, according to this feature sequence, we design a sliding local attention mechanism model SLAM for detecting malware. The experimental results show that our feature extraction method is very effective. Our contributions are as follows:

(1) Analyze the characters of the API execution sequence and classify the APIs into 17 categories, which provides a fine-grained standard to identify API types

(2) Implement a 2-dimensional extraction method based on both API semantics and structural information, which enhances a strong correlation of the input vector

(3) Propose a detection framework based on sliding local attention mechanism, which achieves a better performance in malware detection

The remaining of the paper is organized as follows. Section 2 is a brief background on malware classification. The detailed API execution sequence portrait is explained in Section 3.1. Also, the detailed attention mechanism is explained in Section 3.2. Data source and experimental results are discussed in Section 4. Section 5 summarizes the paper and outlines future work.

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, 3], which ignore the original semantics of the code, 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] use 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 the work of [6, 11, 12], they 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. [13] 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 [14]. 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 the work of [15], they 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 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 obtain a good result in their dataset. Qian and Tang [16] analyze the API attributes and divide 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. [17] 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. [18] 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. However, 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. Ma et al. [19] analyze the local maliciousness about malware and implements an anti-interference detection framework based on API fragments, which can effectively detect malware. Anderson and Roth [20] offer a public labeled benchmark dataset for training machine learning models to statically detect malicious PE files. While they complete baseline models based on gradient boosted decision tree model without any hyperparameter optimization, it will still help researchers study further in this field.

In the work of [21], they extract features based on the frequency of the API and compare neural networks with other traditional machine learning methods. In the work of [22], the implemented Markov chain-based detector is compared with the sequence alignment algorithm, which outperforms detector based on sequence alignment. In the work of [23], they represent the sequences of API calls invoked by Android apps during their execution as sparse matrices and use autoencoders to autonomously extract the most representative and discriminating features from these matrices, which outperform more complex and sophisticated machine learning approaches in malware classification.

These methods expand the space for extracting malicious features and improve the applicable scale of the machine learning method, which achieve good results. However, they also have some limitations, mainly reflecting in the following aspects. Firstly, manual methods have high accuracy but require a lot of manpower, which make them unsuitable for analyzing a large amount of malicious code. Secondly, machine learning is greatly influenced by the training set and its practicality is weak. For example, we have performed 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. Thirdly, when the sample is confused, the training model is difficult to achieve good results.

In fact, no matter if it is converted to images [24], signals, frequency, and other characteristics, it cannot truly express malicious code. The method of extracting more efficient sequences by n-gram slicing [25, 26] 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. Recently, the XLNet model [5], which employs attention mechanisms, has achieved remarkable success in NLP, translation problems, and machine question and answer. It indicates that there is a new stride on deep learning. In response to this situation, for exploring, we further study how to apply attention mechanism in the field of malware classification.

3. Our Method

We first analyze the attributes of the API and divide APIs into 17 categories based on its functionality and official definition. After that, we analyze the characteristics of the attention mechanism and construct a sliding local attention mechanism model (SLAM) according to our data characteristics.

3.1. API Analysis. The API we studied here mainly refers to the system call function under Windows system. According to the Windows official document, the total number of Windows API is more than 10,000, but most API functions are not frequently used.

We firstly extract the 310 most commonly used API from our dataset and then classify them according to their functional characteristics and their harm to the system, which is different from the work of [16]. By studying its harm to the system, we could be better at representing the structural information for the API execution sequence. Finally, we divide these API into 17 categories and colored them, as shown in Table 1, which make the structural information more intuitive.

Based on our classification of API categories, we can represent an API execution sequence as an API category call sequence, which helps us to look at the API execution sequence from a higher API category perspective. Thus, it can be used to represent the structure information of the API execution sequence, which will help us get the information of the API execution sequence from a higher perspective. Here, we can think that it has obtained structural information for the API call sequence.

3.2. Attention Mechanism Analysis. The attention mechanism is a deep learning model which is mainly used in computer vision and NLP. Especially, employed into the complex NLP field, such as machine translation, reading comprehension, machine dialogue, and other tasks, the attention mechanism model can fully demonstrate its learning ability. In essence, the attention mechanism imitates on the processing of the human brain, that is, mainly

Table 1: API category classification description.

Category	Index	Colored	Description
Undefine	0		Undefined API in category dictionary
Net	1		API related to network operations includes socket, wsa, etc.
File	2		API related to file operations includes read, write, copy, etc.
Process	3		API related to process operations includes thread, process, etc.
Reg	4		API related to registry operations.
Device	5		API related to device operations includes mouse, keystone, etc.
Cert	6		API related to cert operations includes encrypt, decrypt, etc.
System	7		API related to system operations includes dll, error, etc.
Service	8		API related to services operations
Window	9		API related to window operations includes findwindow, drawwindow, etc.
Memory	10		API related to memory operations includes readmemory, writememory, etc.
Privilege	11		API related to privilege operations
Com	12		API related to com operations includes createinstance, etc.
Message	13		API related to message operations includes sendmessage, recieve, etc.
Debug	14		API related to debugger operations
Shell	15		API related to shell operations
Data	16		API related to data operations includes buffer, etc.
Session	17		API related to session operations includes encrypt, decrypt, etc.

focuses on some key part from the massive input information. The attention mechanism can be described by the following formula:

$$\text{Attention}(Q, K, V) = F(Q, K)V. \tag{1}$$

In this formula, Q represents a query vector and K and V represent a set of key-value pairs. Through this formula, we can query the weight value of Q in the global context. Since different q value correspond to different weight values, it achieves the purpose of paying attention to the key parts. Recent popular deep learning models, such as BERT [27] and XLNet [5], are based on attention mechanisms and are successfully applied on the NLP field, which demonstrate their powerful machine learning capabilities. Because of the existence of context in NLP and the problem of out-of-order in sentence, it will greatly restrict the effectiveness of some deep learning model. In response to this problem, XLNet uses a two-stream attention mechanism to extract key values from both a content and context perspective, thereby it significantly improves performance. This is instructive for us to apply attention mechanism on the field of malware classification. We will explore the application of attention mechanisms according to the characteristics of malware.

3.3. Detection Framework. Based on both the API and attention mechanism analysis in the previous section, we will build our own feature extraction methods and build targeted detection framework. The whole process is divided into 4 parts: data processing, feature extraction, model construction, and result output.

3.3.1. Data Processing. We use the Cuckoo software [28] to build a virtual sandbox that captures the sequence of API calls for executable programs. We then collect all the APIs that appeared in the sample and build an API dictionary to map the API to a unique number by using word2vec [29]. Through this conversion, an API call sequence can be converted into a number sequence. The process can be defined as follows. Define transferAPI function, which can be used to obtain the API's number according to the API dictionary:

$$\text{API\_num} = \text{transfer API}(\text{API}). \tag{2}$$

3.3.2. Feature Extracting. We select 310 API which are frequently used by the samples and divide them into 17 categories. Then, based on the frequency of the category

Table 2: API execution sequence transfer description.

	7	GetSystemTimeAsFileTime	1
	10	NtAllocateVirtualMemory	11
	10	NtFreeVirtualMemory	8
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	10	NtAllocateVirtualMemory	11
	7	SetUnhandledExceptionFilter	13
	7	LdrLoadDll	4
	7	LdrLoadDll	4
	7	LdrGetProcedureAddress	23
	7	LdrUnloadDll	5
	13	NtCreateMutant	25
	10	NtCreateSection	41
	10	NtMapViewOfSection	78
	7	LdrLoadDll	4
	7	LdrGetProcedureAddress	29
	7	LdrUnloadDll	5
	13	NtCreateMutant	6
	10	NtCreateSection	9
	10	NtMapViewOfSection	24

tags appeared, a category dictionary is built so that the category can be uniquely represented as a number. Through the category dictionary, we can convert an API execution sequence into a number sequence. Because this number sequence contains the category information of the API execution sequence, it can be used to represent the structural information of the API execution sequence. For example, we obtain an API execution sequence by Cuckoo sandbox (Virus Share 0a83777e95be86c5701aa-ba0d9531015 from virus share website [30]). Then, through the category mapping, we can get its category call sequences, as shown in Table 2. The process can be described as follows.

Firstly, we define transferToAPICategory function, which can be used to obtain the API's category by category dictionary.

Secondly, we define indexAPICategory function, which can be used to obtain the index of the API category.

Then, we can get that

$$\begin{aligned} c_i &= \text{transfer To API Category (API)}, \quad \text{where API} \in \text{API\_Set}, \\ i &= \text{index API Category}, \quad \text{where } (c_i) \in \text{API Category}. \end{aligned} \tag{3}$$

Furthermore, we can construct a two-dimensional input vector as shown below. Define findIndex function, which is used to obtain the index of the API according to the category dictionary. Then, we can get that

$$\begin{aligned} \text{Input\_Vector} &= \langle \text{API-Sequence}, \text{Category-Sequence} \rangle, \\ \text{Category\_num} &= \text{find Index (API)} = \text{index API Category}(c_i), \\ \text{API\_Sequence} &= \langle \text{transfer API}(\text{API}_1), \ldots, \text{transfer API}(\text{API}_{2000}) \rangle, \\ \text{Category\_Sequence} &= \langle \text{find Index}(\text{API}_1), \ldots, \text{find Index}(\text{API}_{2000}) \rangle. \end{aligned} \tag{4}$$

If the length of the sequence is not enough 2000, then 0 is added; otherwise, it is truncated. Through these operations, we can extract two-dimensional input vectors.

3.3.3. Model Construction. According to the characteristics of the API execution sequence with length of 2000, several adjacent API calls actually have practical meaning, that is, the entire API execution sequence has certain local significance. Therefore, we design a local attention mechanism to acquire the features of these adjacent APIs with local significance. Furthermore, drawing on the idea of CNN, a sliding window method in a certain step size is used to scan the entire API execution sequence. After that, we use CNN to gain the weight value of sliding local attention. The Softmax function is finally used to output result. The entire structure is shown below in Figure 1 and the entire process can be described by the following Algorithms 1–3.

In Algorithm 1, we define a function SPLIT_TENSOR, which is used to handle tensor for the Local Attention Structure. In Algorithm 2, we define a function LOCAT_ATTENTION, which is used to output local tensor. In Algorithm 3, we construct the SLAM Framework by the function MAKE_SLAM.

3.3.4. Result Output. The whole process is divided into the training phase and detecting phase. The training phase is mainly used to train the model. In the detection phase, samples are entered into the trained model to produce an output.

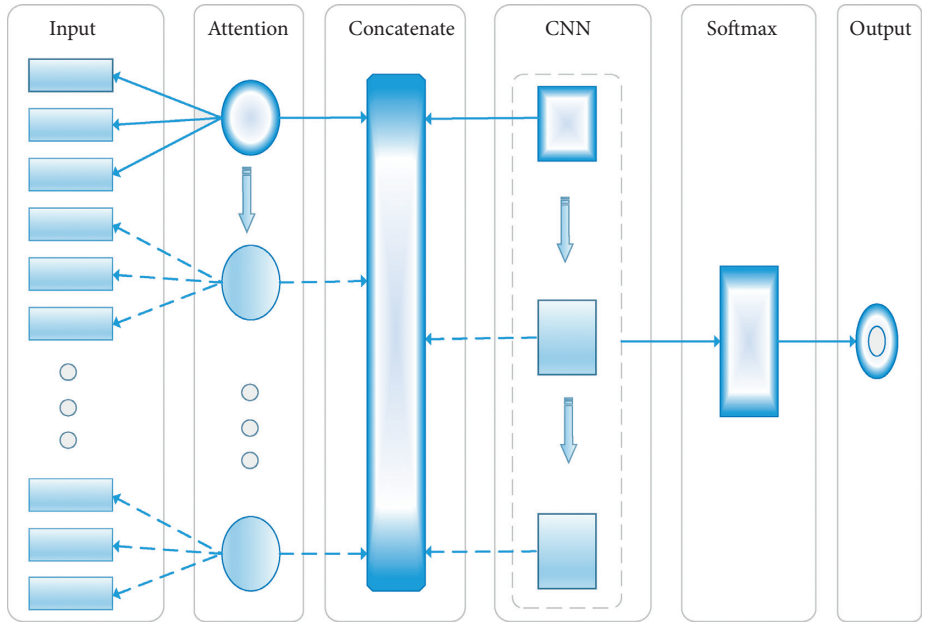

Figure 1: Sliding local attention model.

4. Experiment Result and Evaluation

4.1. Dataset and Environment. In order to make our model more convincing, here we use the public dataset (the data set of Alibaba 3rd Security Algorithm Challenge [31]). The dataset consists of API call sequences which are generated by the windows executable program in the sandbox simulation. It is mainly composed of normal program, infected virus, Trojan Horse program, mining program, DDoS Trojan, and extortion virus. We treat these five malicious types as a same malicious type. Then, the dataset is classified into two categories, that is, normal samples and malicious samples. The sample size of the dataset is shown in Table 3.

As shown in Table 3, the normal data is 110000 and the malware data is 27770. Since the number of normal samples and the number of malicious samples are very different, we adopt a random sampling method to construct the dataset, and a total of 9192 samples are selected.

The runtime environment of the experiment includes Ubuntu 14.06 (64 bit), 16 GB memory, 10G Titank GPU.

4.2. Experiment Result and Analysis. In order to evaluate our model, we choose Accuracy, Precision, Recall, and $F1$-Score as evaluation criteria. Define TP for True Positive, which is the number of samples classified as normal category correctly. Define FN for False Negative, which is the number of samples classified as malicious category correctly. Define TN for True Negative, which is the number of samples classified as malicious category wrongly. Define FP for False Positive, which is the number of samples classified as normal category wrongly. Then, these evaluation criteria could be defined as follows:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}},\\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}},\\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}},\\
F1 - \text{Score} &= \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}.
\end{aligned} \tag{5}$$

We adopt the 10-fold crossvalidation method to validate our model SLAM and obtain their average value for evaluation.

The confusion matrix for our model SLAM is as shown in Table 4. The accuracy of our model SLAM is shown in Figure 2. As can be seen from Figure 2, the accuracy of our model SLAM is at least 0.9586, the highest is 0.9869, and the average is 0.9723, which achieves a good classification effect.

We select the best 7-fold model to further evaluate its other indicators. The ROC curve for our model SLAM is shown in Figure 3. From Figure 3, we can see that the ROC curve area is about 0.9870.

The classify report for our model SLAM is as shown in Table 5. From Table 5, we can see that the Precision, Recall, and $F1$-score indication are about 0.9869. From the results of these experiments, we can see that our model SLAM achieves a good classification result.

By further analyzing our model, we can know that it can obtain the local information contained in the API execution sequence through the local attention mechanism, which will be beneficial to the classifier. Also, it successfully scans the entire API execution sequence by sliding the window, which obtains a broad view. Meanwhile, the two-dimensional input vector we construct contains both API semantics and structure information, which will greatly enhance the relevance of the input information. In general, effectively extracting data

```
Input: tensor, index, step_size
Output: temp_tensor
  function SPLIT_TENSOR (tensor, index, step_size)
    construct a Lambda expression according to keras
    Initialize temp_tensor
    temp_tensor = cut tensor according to its index from index to index + step_size
    return temp_tensor
end function
```

Algorithm 1: Split tensor vector.

```
Input: query, key, value
Output: local_tensor
    function LOCAL_ATTENTION (query, key, value)
      Initialize local_tensor
      Initialize F function (from Attention mechanism)
      local_tensor = F (q, k) v
      return local_tensor
    end function
```

Algorithm 2: Construct local attention structure.

```
Input: input_vector, step_size
Output: output
    function MAKE_SLAM (input_vector)
      Initialize length
      Initialize CNN function from CNN layer
      Initialize Softmax function from Dense layer
      Initialize concatenate as a middle layer
      length = get the length of input_vector
      for (index = 0, index < length, index += step_size) do
        temp_tensor = SPLIT_TENSOR (input_vector, index, step_size)
        local_tensor = LOCAL_ATTENTION (temp_tensor, temp_tensor, temp_tensor)
        append local_tensor into concatenate
        end for
        tensor = CNN (concatenate)
        output = Softmax (tensor)
        return output
    end function
```

Algorithm 3: Construct SLAM framework.

features and designing a targeted model framework based on data characteristics is the reason why our model SLAM achieves good results.

4.3. Comparison with Other Input. To verify the validity of our 2-dimensional feature extraction method, we compare them with different feature extraction method by our model SLAM. We use the 1-dimensional API index sequence with no structural information as a comparison and use the accuracy rate as an indicator. We still use 10-fold cross-validation and the results are shown in Figure 4.

The comparison results of the average accuracy are shown in Table 6. From Table 6, we can see that the 1-d input accuracy is 0.9484 and the 2-d input accuracy is 0.9723. It can be seen that the 2-dimensional feature extraction method is higher than the 1-dimensional feature extraction

Table 3: The sample size of the dataset.

Normal	Malware
110000	27770

Table 4: The confusion matrix of our model.

	Normal	Malware
Normal	475	6
Malware	6	432

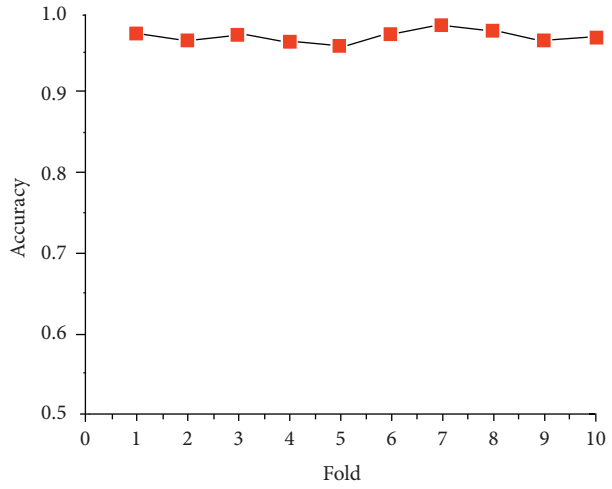

Figure 2: The accuracy of SLAM for 10-fold crossvalidation.

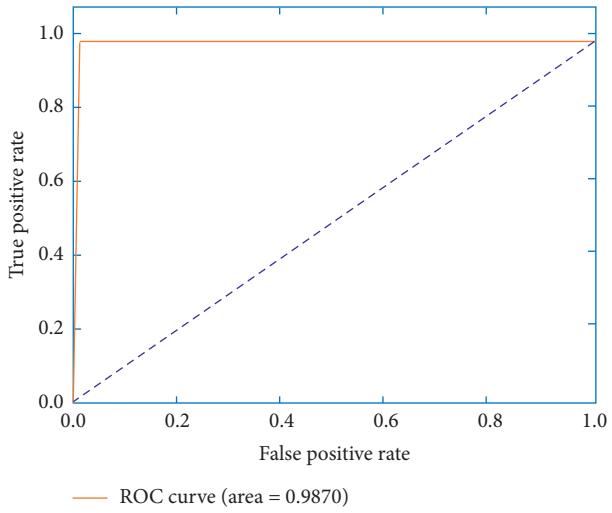

Figure 3: The ROC curve for SLAM.

Table 5: The classify report for our model.

	Precision	Recall	$F1$-score	Support
Normal	0.9875	0.9875	0.9875	481
Malware	0.9863	0.9863	0.9863	438
Macro avg	0.9869	0.9869	0.9869	919

method by an average of nearly 3 percentage points. This proves the effectiveness of our 2-dimensional feature extraction method based on semantics and structure.

4.4. Comparison with Other Models. For comparison, we choose three baseline models.

4.4.1. Baseline Model 1. Random Forest is an emerging, highly flexible machine learning algorithm with broad application prospects, which is often used in many competitions. In the work of [32], they also use random forest as one of models, and the result of the random forest model were the best. Therefore, we choose random forest as our baseline model, and its parameters are set as follows: $n\_estimators = 500$ and $n\_jobs = -1$.

4.4.2. Baseline Model 2. In the work of [33], they use an Attention_CNN_LSTM model to detect malware, which we call it ACLM and treat it as our baseline model.

4.4.3. Baseline Model 3. In the work of [5], they use a two-stream attention mechanism model based on content and

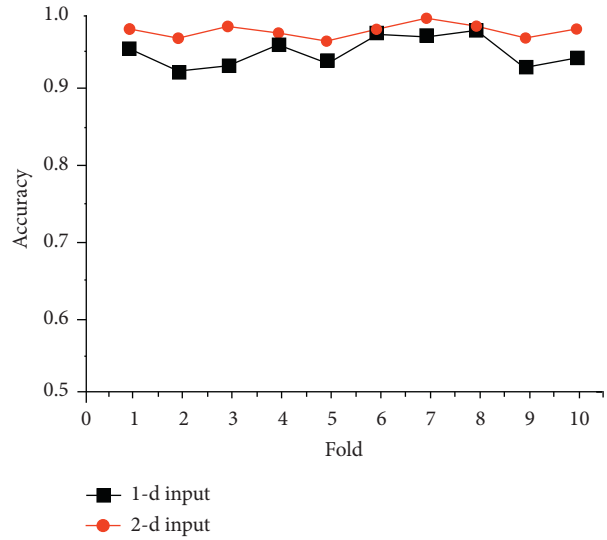

Figure 4: Comparison with different dimensions.

Table 6: Comparison with different inputs.

Input	Accuracy
1-dimensional API	0.9484
2-dimensional API	0.9723

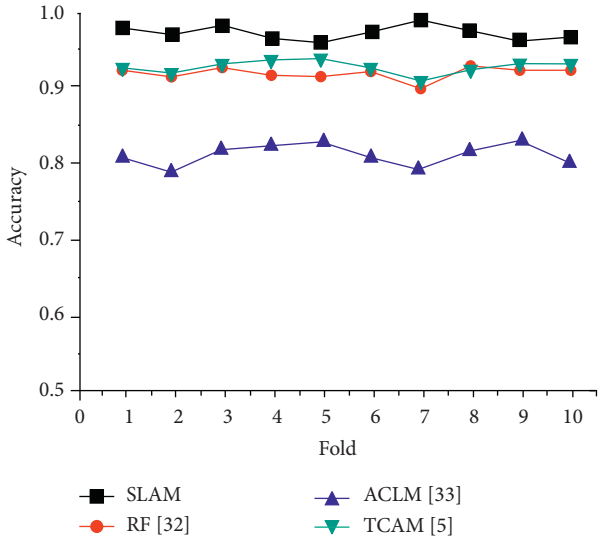

Figure 5: Comparison accuracy with 10-fold crossvalidation.

context information to resolve the NLP problem. By drawing on their ideas, we construct a two-stream CNN-Attention model as a baseline model called TCAM.

4.4.4. Comparison Result. We use 10-fold crossvalidation to verify these models. The results of the comparison are shownin Figure 5.

We count the average accuracy of these models based on 10-fold crossvalidation. The comparison results are shown

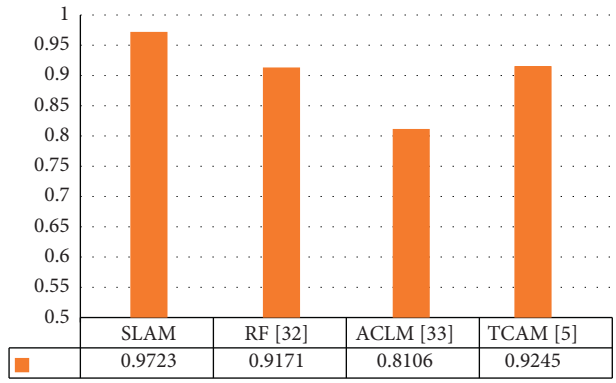

Figure 6: Comparison average accuracy for models.

Table 7: The classify report for our model.

	Accuracy	Precision	Recall	*F*1-score	Support
SLAM	0.99	0.99	0.99	0.99	919
RF [32]	0.9270	0.93	0.93	0.93	919
ACLM [33]	0.81	0.82	0.82	0.82	919
TCAM [5]	0.9325	0.93	0.93	0.93	919

below in Figure 6. In Figure 6, our SLAM model accuracy is 0.9723, the RF model accuracy is 0.9171, the ACLM model accuracy is 0.8106, and the TCAM model accuracy is 0.9245.

Also, we select the best ones from the 10-fold cross-validation of these models and compare them by Accuracy, Precision, Recall, and *F*1-score. The results of the comparison are shown in Table 7.

From these comparison results in Figures 5 and 6 and Table 7, we can see that our model has a better classification effect. The Accuracy, Precision, Recall, and *F*1-score for our model SLAM are all about 0.99. According to these results, we conduct an in-depth analysis.

For the RF [32] model, it is a classic traditional machine learning method, which basically represents the limit of the

traditional machine learning method, but it is difficult to go beyond deep learning.

For the ACLM [33] model, due to the API execution sequence of up to 2000, the extraction based on the attention mechanism will be diluted. Because it is in such a long sequence, it will be difficult to really notice the key parts.

For the two-stream TCAM [5] model migrated according to the content and context idea, some of its ideas are worth learning, but the malware is different from the NLP. Thus, it still needs to be improved according to the target.

Our model SLAM is based on the sliding local attention mechanism, which can well match the data characteristics of the API execution sequence, so that it achieves the best classification effect.

5. Conclusion

We analyze the characteristics of the API execution sequence and present a 2-dimensional extraction method based on semantics and structure information. Furthermore, according to the API data characteristics and attention mechanism, we design and implement a sliding local attention detection framework. The experimental results show that our feature extraction method and detection framework have good classification results and high accuracy. In the future work, we will further explore the application of attention mechanisms in the malware detection area.

Data Availability

The dataset of Alibaba 3rd Security Algorithm Challenge can be obtained from 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. 2018YFB0805000).

References

[1] 360 Security Report, 2019, http://zt.360.cn/1101061855.php?dtid=1101062370did=610142397.

[2] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security—VizSec'11*, Pittsburgh, PA, USA, July 2011.

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

[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] Z. Yang, Z. Dai, Y. Yang et al., "XLNet: generalized autoregressive pretraining for language understanding," 2019, https://arxiv.org/abs/1906.08237.

[6] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proceedings of the 2005 in 2005 IEEE Symposium on Security and Privacy (S&P'05)*, 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, San Diego, CA, USA, August 2004.

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

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

[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. G. Llaurad, *Convolutional Neural Networks for Malware Classification*, Rovira i Virgili University, Tarragona, Spain, 2016.

[12] L. D. Vu Duc, "Deepmal: deep convolutional and recurrent neural networks for malware classification," 2018, https://arxiv.org/pdf/2003.04079.

[13] 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.

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

[15] F. Xiao, Z. Lin, Yi 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.

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

[17] 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.

[18] 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, New Delhi, India, September 2014.

[19] X. Ma, S. Guo, W. Bai, J. Chen, S. Xia, and Z. Pan, "An API semantics-aware malware detection method based on deep learning," *Security and Communication Networks*, vol. 2019, Article ID 1315047, 9 pages, 2019.

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

[21] 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.

[22] M. Ficco, "Comparing API call sequence algorithms for malware detection," in *Advances in Intelligent Systems and Computing*, Springer, Berlin, Germany, 2020.

[23] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and API-images," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 26–33, 2020.

[24] 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*, ACM, Chicago, IL, USA, pp. 21–30, October 2011.

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

[26] 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.

[27] J. Devlin, M. W. Chang, K. Lee et al., "Bert: pre-training of deep bidirectional transformers for language understanding," 2018, https://arxiv.org/abs/1810.04805.

[28] Cuckoo Sandbox, 2019, https://cuckoosandbox.org/.

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

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

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

[32] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: a systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, pp. 236–250, 2019.

[33] S. Luo, Z. Liu, B. Ni et al., "Android malware analysis and detection based on attention-CNN-LSTM," *Journal of Computers*, vol. 14, no. 1, pp. 31–44, 2019.