Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/4/2023329

Dynamic malware detection based on CNN-LSTM network

Jiate Chen

Zhejiang University of Technology, Hangzhou, China

chenot0221@163.com

Abstract. As cyberspace continues to evolve, so does malicious behavior across the network. Attacks in cyberspace, such as the spread of malware, bring harm and risks to users. Malware will attempt to steal user information or impair system resources. Therefore, to protect against malware, a series of detection software such as firewalls have been developed to protect users' information and systems from damage. As malware continues to evolve and develop, and the forms of attack are increasing, the corresponding malware and detection techniques should also be evolving issues. The commonly used raw detection technique is based on signature and behavior specification, which has the advantages of fast speed and less computation. However, this method is limited to known software, the target software which is an unknown software will not be applicable. Therefore, one obvious shortage of traditional malware detection techniques is that new or unknown malware are not likely to be detected. To solve this problem, this paper uses machine learning to analyze different features of malware. By this way, new malware can be detected by the model. Based on the API call of the Windows system, this paper proposed a convolutional neural network for the classification of malware types. On this basis, this paper used CNN to extract the features, considering the text characteristics of the API sequence. After subsequence debonding and meaningless suffixes deletion of the original API sequence, word vectors were obtained through training as the input of the subsequent machine model. Then, different convolution kernels were used to process contextual semantic information. Then, the LSTM model is used to extract the long-distance features from the features.

Keywords: Malware, API sequence, Detection, LSTM, CNN.

1. Introduction

In order to reduce the harm caused by malware, users need more effective malware detection technologies. Malware detection methods can be divided into static and dynamic malware detection[1].

Static malware detection technology is mainly based on the static characteristics of the target PE file. These features are extracted by the analyzer, including string patterns, opcodes, and byte sequences of the file. The static analysis method generally comprises a database which contains static characteristic data of known malicious software, so the static characteristics of the target software are matched with the existing records in the database during detection to judge whether the target software belongs to the malicious software. Considering the principle of detection, the static detection method does not need to execute the target software[2], and the detection speed is relatively fast, so it is widely used. However, static detection can not solve the problem of malware detection perfectly, the

DOI: 10.54254/2755-2721/4/2023329

main disadvantage is that it can not determine the type of malware that is not contained in the database. At the same time, malware can bypass static detection through obfuscation techniques or prevent detection by encrypting the signature. Therefore, in order to solve these problems, the dynamic malware detection method will analyze the malicious behavior of the software during execution. The main process is to run the malware in a virtual environment isolated from the user's actual system environment, and monitor the running behavior of the software, and then analyze and judge the type of the software.

The dynamic detection method analyzes various behaviors of malicious software, including accessing the network, changing the registry, requesting system calls, using memory space, and so on[3]. In order to analyze behaviors, selected features used for detection should reflect those behaviors. The sequence of Windows API calls is a good indication of how the software will behave as it is being executing. This sequence shows a series of actions by which the software requests the system to operate. Therefore, the software behavior can be analyzed by analyzing the API calling sequence of the target software in the running process, so as to determine the software type[4].

The API call sequence is actually a piece of text, whose semantics reflects the interface call of the software, and the behavior of the software can be analyzed through the context. Therefore, we can consider the natural language processing [5] of the sequence and use the general text classification model to complete the detection of malicious software. And the API sequence records the API calls of the software in chronological order, so it contains the time characteristics as well. In order to analyze The semantic information and the time characteristics contained in the sequence of API calls, this paper proposed the CNN-LSTM model and trained the model to detect malware.

2. Relative Work

Malware is widely circulated in cyberspace today. Anti-virus companies can catch tens of thousands of them every day. Manual analysis of malware samples is difficult to adapt to the growing trend of malicious software, and automated malware detection has become a mainstream approach. Traditional malware detection is based on the signature, that is, extracting unique signatures from known malware, identifying the features of the malware, and performing feature extraction and comparison on the new malware for identification[6]. Signature detection technology is efficient and fast, but it can only detect known code, and can not detect malware using code obfuscation and polymorphism technology.

Therefore, malware detection technology based on dynamic behavior analysis is favored by researchers. API is the interface between the application program and the system, and API calls can reflect the behavior of the program to a large extent, so most of the dynamic analysis of malware is based on API sequences[7]. Zhu Xuebing et al.[8] proposed a malware detection method based on frequent subgraph mining of malware family behavior, which marks the parameters of API through dynamic taint technology, to obtain the API call relationship and form a single sample behavior dependence graph, and then mine family frequent subgraph patterns for malware detection. Rong Fengping [9] proposed a malware detection method based on malware API called sequential pattern mining. By introducing the concept of object-oriented association mining, the API call model that can represent the behavior pattern of malware is mined, which is used for malware detection. Ding [10] used dynamic taint analysis to construct a familial behavioral dependency graph to detect malware based on maximum weight subgraph matching. Du et al.[11] extracted common malicious API calling patterns from different classes of malware based on biological DNA sequence alignment algorithm. to detect malware.

Currently, most of the malware dynamic analysis detection methods are based on graph and sequence mining, and the methods based on graph matching are subject limited to the graph isomorphism problem, when the number of malware is increasing, the graph match can become increasingly complex, limiting the application of such methods. Use method based on sequence mining is limited to system call injection attack. When extraneous system calls are added to malware, it can greatly affect the effectiveness of methods based on sequence mining. Considering the above dynamic detection techniques based on malware behavior, the lack of technology, and the success of

DOI: 10.54254/2755-2721/4/2023329

deep learning in image and text classification, this paper uses dynamic analysis technology and based on the text. This classification is combined with a convolutional neural network and RNN for malware detection. There are relatively few API functions in an API sequence compared to plain text, and the call frequency of different API functions is also quite different. The behavior of the same kind of malware is quite different, and its API sequence also shows a more obvious difference.

3. Inplementaion

3.1. Windows API Call

The windows API call dataset is a public windows malware dataset offered by Tianchi Dataset, Aliyun, containing 6,145 samples of malicious files among 8 classes. The breakdown of sample size per label is available in Table 1.

Table 1. API call dataset counts.

Malware Type	Sample Size
Common	3960
Ransomware	408
DTLMiner	525
DDos	753
Worm	89
Virus	2989
Backdoor	468
Trojan Horse	1277
Total	10469

3.2. Data Processing

The initial data is in the format of ['file_id', 'label', 'API ',' TID ', 'index'], i.e. the file number, software category label, thread ID, and serial number. The raw data is processed as follows.

The meaningless suffixes will be deleted first. Windows API calls contain suffixes, so the text displayed by the same API call will be different, which will affect subsequent data processing, so they will be deleted, including A, W, ExA, ExW.

The current API data is a string and input vectors are required for model training, so the different API functions are mapped to digital labels. There are 236 classes of API functions in the data set. Therefore, each class of function corresponds to 1-236 respectively to generate an API-index dictionary and save it as ISON file for subsequent training by converting sequences into vectors.

Then, multiple records belonging to the same file are integrated into one data according to the sequence number. Only the file ID, API sequence and software category label are retained, that is, the data format is ['file_id', 'label', 'API_IDx '].

Remove duplicate substrings from API sequences. Malware may perform the same behavior continuously when running, which means that the same API function will be called consecutively many times, or the same combination of API functions will be executed consecutively many times, which will lead to the API sequence containing a lot of repeated information, and excessive redundant information will lead to the excessively long sequence and the increase of model training time. So duplicate substrings in the API sequence are removed. Finally, the data files are saved as PKL files.

3.3. Embedding Layer

To obtain feature vectors, processed data will be put into the embedding layer to generate vectors to be used in the convolution layer.

DOI: 10.54254/2755-2721/4/2023329

First, use a dictionary to convert the API calls into one-hot encoding. The vector dimension is 237, which is the number of API calls plus 1. The integer corresponding to each API in the dictionary represents the position of the number 1 in the vector, and the first index of the vector is 0. For example, if an API corresponds to an integer of 3, the generated vector puts a value of 1 at position 3. This sequence of API calls will be converted to a matrix of dimensions 237. This matrix is then used as skip-gram input to reduce its dimension. The output matrix dimension is 100.

3.4. Convolution and Maxpooling

Because API calls close to each other in program code are closely related in behavioral characterization, and malicious code variants insert irrelevant API calls in key code segments, which is similar to object elongation in images, CNN can extract local features of different dimensions by using convolution kernels of various sizes.

After the matrix is obtained by the embedding layer, the matrix is input to the convolution layer. Filters are used at this level to extract feature vectors. The filter sizes are 3, 4 and 5 respectively, and the number of each type of filter is 100. The matrix needs to be activated by the ReLU function after the filter operation

$$m_i = \theta(wx'_{i:i+s-1} + b) \tag{1}$$

Where θ denotes the ReLU function and b denotes the bias vector. The convolution kernel is utilized to step size 1 sliding process x1:f in the sequence direction to get a feature map:

$$m = < m_1, m_2, ..., m_{f-s+1} > \tag{2}$$

3.5. LSTM Layer

When processing API sequences, the CNN model is used to consider the semantic information of the sequences. However, the processed API sequences have a time-axis correlation, which means the API sequence indicates the order in which the software calls the API. In the process of time-related data sequence, such as the text with up-down correlation, the traditional neural network cannot process this relation, so the RNN recurrent neural network is used instead.

Temporal recursive neural network and structural recursive neural network are collectively referred to as recursive neural network. The forward propagation process of DNN is input X and weight matrix. After bias value B is calculated, the result is sent to hidden layer 2 as input, and the activation function outputs the result after calculation. After another round of calculation, the input and output layer is activated before the output result is obtained.

In the circulating neural network, because the parameters are shared and the gradient of the output result is related to the current and past results, the optimization of the back propagation algorithm needs to add the gradient of W. However, in the traditional neural network, there is no need to share parameters between layers, so it is not required.

RNN can be obtained from the general structure and the back propagation algorithm. The key is that the information of the previous task will be considered when processing the current task and the characteristics of the time series will be advanced. However, if the time interval between relevant information and the current processing content is too large, the recurrent neural network cannot learn information well. Because it is difficult to learn long-term sequence problems by using BPTT algorithm due to gradient divergence. The LSTM model will be used to solve similar problems that may arise when considering issues related to API sequences.

LSTM network is used to solve problems with a long time span. Considering the general cyclic neural network, it can be found that there is the reuse of neural network modules in its structure, such as a TANH layer. However, in the LSTM model, the general structure is similar but there is some improvement. It is modified in the reuse module, adding four operations and mutual input between them. The meaning of gate means that the information passing through can be screened. The

DOI: 10.54254/2755-2721/4/2023329

realization principle is to use sigmoID function to calculate with input, and the output result is vector. Each element in the vector is in the interval of 0,1, which means the weight of information passing.

In LSTM, discarding information is processed first for input information, which is realized by the forgetting gate. The forgetting gate receives the output of the previous cell and the input of the current cell, and outputs a value in the range of 0,1 through sigmoid. The next step is to decide how much information to add to the cellular state in a two-step process. Firstly, the sigmoID function of Input Gate Layer is used to judge the information to be updated, and then the alternative information vector is output by TANH Layer. Finally, the above two outputs are combined to realize the update of cell state. Finally, multiply the old state by ft, discard unnecessary information and add sigmoid function to output the result multiplied by the current cell state value.

Finally, in the output stage, the output value will be determined based on the cell state. The first step is to determine the output portion by sigmoid. The cell state is then calculated as the TANH input and then multiplied by the sigmoid layer output to determine the final result.The calculation formula is as follows.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-i} + b_i) \tag{3}$$

$$f_t = \sigma\left(W_{xf}x + W_{hf}h_{t-1} + b_f\right) \tag{4}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{5}$$

$$C_t = f_t C_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{6}$$

$$h_t = o_t \tanh(C_t) \tag{7}$$

Where σ represents the sigmoid function, W and b respectively represent the weight matrix and bias vector corresponding to the subscript, for example, Wxi is the weight matrix from the input layer to the input gate, xt is the input at a certain time point, that is, the vectorized representation of API, and it, ft and ot respectively represent the input gate, forgetting gate and output gate. Ct is the memory unit and ht is the output of the hidden layer.

3.6. Result

To evaluate and compare the proposed model, use the TextCNN model without LSTM layer as a comparison. Table 2 shows the confusion matrix of the CNN-LSTM model. Table 3 shows the confusion matrix of the TextCNN model.

Table 2. Confusion matrix of CNN-LSTM model.

	0	1	2	3	4	5	6	7
0	976	0	3	0	0	4	0	13
1	3	93	1	0	0	1	0	2
2	5	0	218	0	0	8	2	6
3	3	1	4	106	0	5	0	45
4	1	0	1	9	0	2	0	7
5	9	1	4	21	0	805	5	13
6	7	0	4	5	0	2	54	31
7	15	5	22	57	0	14	23	162

DOI: 10.54254/2755-2721/4/2023329

Table 3. Confusion matrix of TextCNN model.

	0	1	2	3	4	5	6	7
0	982	0	1	0	0	2	2	9
1	6	86	6	0	0	2	0	0
2	15	4	195	0	0	4	3	18
3	3	1	4	103	0	7	0	46
4	1	0	1	9	0	2	0	7
5	7	0	5	13	0	811	6	16
6	11	0	6	5	0	8	14	59
7	24	5	21	52	0	11	15	170

To further the evaluation of the model, several indicators were used. Table 4 shows the indicators including accuracy, recall rate, F1-score and the average.

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{8}$$

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{9}$$

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{10}$$

$$\mathrm{BF1} = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Preision} + \mathrm{Recall}} \tag{11}$$

Table 4. Classification report of CNN-LSTM model.

	precision	recall	F1-score	support
0	0.94	0.99	0.96	996
1	0.90	0.86	0.88	100
2	0.82	0.82	0.82	239
3	0.57	0.63	0.60	164
4	0.00	0.00	0.00	20
5	0.96	0.95	0.95	858
6	0.35	0.14	0.20	103
7	0.52	0.57	0.55	298
Accuracy			0.85	2778
Macro Avg	0.63	0.62	0.62	2778
Weighted AVG	0.84	0.85	0.84	2778

4. Conclusion

This report proposed CNN to extract features to classify malware based on its API. The accuracy of this model reached 85%. The main reason for the relatively low accuracy is the lack of the data set. For some types of software, like common software and virus, since data set contains relatively abundant samples, this model performs well while classifying these two types. However, for worm software, there are less than 100 samples in the data set. Therefore, the model cannot classify test samples well, leading to low accuracy.

More samples are needed. Not only more amount, but also more types are needed to improve this model. In this way, accuracy can be increased and this model can classify more types.

Another point is that this model cannot explain each result. Since we use skip-gram to lower the dimension of the feature vectors, it is impossible to figure out which API call sequence affects the

DOI: 10.54254/2755-2721/4/2023329

result. To solve this problem, a model that remains the API call information is needed. Thus we can find out the weight of each API call in the process.

References

[1] Zong, H. (2019). Research on deep learning algorithm of Android malware static detection based on behavior pattern. (Doctoral dissertation, Beijing University of Posts and Telecommunications).

[2] Oliver I. & Pereira R. (2013). Malicious software detection. US, US8392996 B2.

[3] Hisham, Shehata, Galal, Yousef, Bassyouni & Mahdy, et al. (2016). Behavior-based features model for malware detection. Journal of Computer Virology & Hacking Techniques.

[4] Iwamoto, K. , & Wasaki, K. (2012). Malware classification based on extracted API sequences using static analysis. Asian Internet Engineeering Conference (pp.31-38). ACM.

[5] Manning, C. D. , & H Schütze. (1999). Foundations of statistical natural language processing. MIT Press,.

[6] Ahmadi, M. , Ulyanov, D. , Semenov, S. , Trofimov, M. , & Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. ACM.

[7] Salehi, Zahra, Ghiasi, Mahboobe, Sami, & Ashkan. (2017). Maar: robust features to detect malicious activity based on api calls, their arguments and return values. Engineering Applications of Artificial Intelligence: The International Journal of Intelligent Real-Time Automation.

[8] Zhu Xuebing, Zhou Anmin, & Zuo Zheng. (2019). Malicious code detection based on frequent subgraph mining of family behavior. Information Security Research, 5(2), 9.

[9] Rong Fengping, Fang Yong, Zuo Zheng, & Liu Liang. (2018). Macspmd: Malicious Code Detection Based on Sequence Pattern Mining of Malicious API Calls. Computer Science, 45(5), 8.

[10] Ding, Y. , Xia, X. , Sheng, C. , & Li, Y. (2018). A malware detection method based on family behavior graph. Computers & Security, 73(MAR.), 73-86.

[11] Du, D. , Sun, Y. , Ma, Y. , & Xiao, F. . (2019). A novel approach to detect malware variants based on classified behaviors. IEEE Access, PP(99), 1-1.