Detecting Android Malware Based on Dynamic Feature Sequence and Attention Mechanism

Hanlin Long
College of Computer Science and Technology
Harbin Institute of Technology (Shenzhen), China
forturnme@gmail.com

Zhicheng Tian
College of Computer Science and Technology
Harbin Institute of Technology (Shenzhen), China
19s051053@stu.hit.edu.cn

Yang Liu
College of Computer Science and Technology
Harbin Institute of Technology (Shenzhen), China
Cyberspace Security Research Center, Peng Cheng Laboratory
liu.yang@hit.edu.cn

Abstract—**The mechanism of running software on virtual machines partly ensures the security of Android system. However, with all kinds of malicious codes being developed, there has been a huge number of massive security incidents caused by malware on Android. *Malware* has various code patterns, but their behaviors are measurable. In this paper, a new method of detecting Android malware by analyzing malware's behaviors is proposed. The method is characterized by the ability to mine the *contextual* relationships between system calls and network activities. Besides, the method requires only a small data set to achieve good classification performance.**

We propose a set of methods for automatically collecting and organizing dynamic features from Android application Based on the collected features, deep neural network is used to classify software samples. We validate *the* effectiveness of the proposed method on a set of 2210 applications obtained from Androzoo. The experimental results demonstrate that the proposed method has high detection accuracy against wild malware as compared with other methods.

Keywords-Malware Detection; Attention Mechanism; Transformer Structure; Android System; Dynamic Features.

I. Introduction

Android system was first released in 2008. After 10 iterations of major versions and 29 updates on its API design, Android operating system now runs on more than 75% mobile devices around the world, making it the most popular mobile operating system in the world [1].

Android system has a basic structure of running programs based on the Linux kernel and using virtual machines. At the same time, it implements a permission management framework, users can choose whether to grant specific permissions to an application. The acquisition of some special rights is more strictly restricted. All these measures help to ensure the security of the Android system to some extent.

At present, the basic form of malware on Android system has changed from radical short-term forms such as directly destroying the system or obtaining system control to mild long-term forms such as ransomware, advertising, bundled downloads, traffic acquisition, information theft, etc. However, the threat to users is not reduced. According to a special report released by 360 Internet Security Lab [2], the number of Android malware samples collected in 2019 is significantly smaller than that in 2018, and the main malicious behaviors are tariff consumption, privacy theft, and remote control.

At the same time, due to the serious fragmentation of the Android system, the promotion of the new version of the system and security patches is hindered, and the exploitable vulnerabilities are often not repaired in time. In this case, detection is particularly important to prevent malware from running on the device. However, the existing Android malware detection schemes rely too much on static features, such as bytecode and permission list, which are easy to be camouflaged and changed, and cannot find new security threats timely and effectively. Moreover, they are too large, or too dependent on the cloud to effectively protect devices under new threats. Therefore, a reliable detection method based on dynamic features is imperative.

To solve the above problems, this study proposes a feature coding scheme based on network activity and system call sequence and a lightweight model based on attention mechanism using transformer structure. While the size and the computational cost are smaller than other models, the proposed model retains good classification performance. It achieved an f-measure of 0.74 on the data recorded by the real machine, at the same time, this scheme can characterize and learn malicious behavior well. A good detection rate can also be obtained for wild malware, which refers to that newly detected malware with new code features.

II. Related Works

There are two main entry points for works in this direction: one is to handle static information inside the application installation package (APK) and the other is to run application on a real machine to obtain dynamic information. According to the different entry points, these works can be divided into two categories: static analysis and dynamic analysis.

Static analysis refers to direct analysis according to the contents of the installation package. Its analysis basis includes byte code, API call sequence, list files, etc. Lee et al. [3] used a natural language processing method along with packet names and certifications, trained cyclic neural network model (RNN) achieves 94.2% TPR under FPR 1% limit. Amin et al. [4] used two-way long-term memory networks (Bi-LSTM) models trained by One-Hot coding to obtain a 0.999 F1-score and 1.4% FPR. Detection methods using bytecode can easily get excellent results, but resources and time overhead should be huge.

Dynamic analysis refers to analyzing application packages according to its runtime performance. Its analysis based on various information includes system call sequence, resource usage, network activity, etc. Qing Yu et al. [5] used a script to operate Android application, obtained system call sequence and system service usage sequence, designed an algorithm based on improved Gaussian kernel function which obtained 0.952 TPR and 0.185 FPR. Alptekin et al. [6] used several logs collected from real machines like system call sequence, broadcast events, binder IPC communication sequence, power management unit events, and resource usage statistics to obtain a 0.93 F1-score via SVM. Besides, there are methods starting from network activities. Shanshan Wang et al. [7] achieves a 97.89% detection rate on a C4.5 model evaluating TCP stream and HTTP request captured by server-side monitors.

Besides, there are some comprehensive methods combining these approaches. Among them, classic DREBIN work [8] was published in 2014 which used SVM and several static features to achieve 93.9% accuracy. Wei Wang et al. [9] obtained 99.89% TPR and 21% FPR in their test using the CNN model trained by multiple static features.

Although all these researches have achieved good results in the experimental environment, these models still have strong timeliness. Because the models detect malware by categorized malware families, they are good at detecting malicious software. But they are weak when facing malware released late and uncategorized. In the work of Shanshan Wang et al. [10], they selected a set of 337 newly released application packages as the test set, of which 242 were malware. For this malware, their method achieved only 50% of the detection rate, while the commercial security application with the best detection achieved only 56.5% of the detection rate.

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

III. METHOLOGY

This paper focuses on constructing a method for detecting malware based on aggregated dynamic features. Dynamic features applied to the method include system call sequence, network activity sequence, and key location access sequence. Key location access can be extracted by processing specific system calls. Aiming at shortcomings of existing detection schemes with weak performance facing newly released malware, and the limitation of detecting by categorized malware families, this paper proposes a solution based on aggregated dynamic feature sequences. The new method improves detection performance by utilizing contextual relationships among different dynamic features.

A. Aggregated Dynamic Feature Sequence

Aggregated dynamic feature sequence refers to an activity sequence consists of chronological system calls and network activities. Compared with traditional separated processing methods, the advantage of aggregation is that it can discover inner relationships between network activities and system calls so that malicious behaviors can be found easier.

The aggregated features mentioned in this article are collectively referred to dynamic feature items extracted with *strace* and *tcpdump* from partial key system calls and IP activities. For most system calls in Android, this article does not care about its parameters but records its system call name. For those IP activities, this article does not care about its address but records its flow direction (flow in or out from the device). Additionally, five frequently occurred system calls related to time and synchronization are skipped beyond consideration, which are named *getpid*, *getuid_32*, *gettimeofday*, and *futex*.

When aggregated features entering the model, they will be encoded as digital sequences so that model can easily process them. Finally, the feature sequences obtained are conducted as integer arrays with indefinite length.

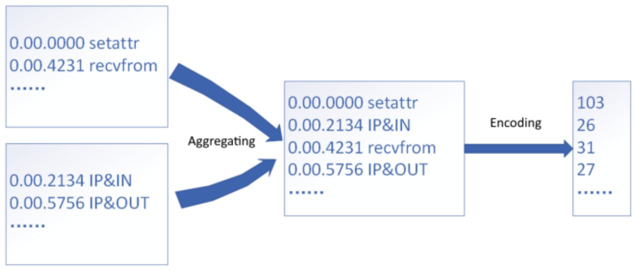


Fig.1. The generation of aggregated dynamic features.

For *ioctl* calls which often involving privacy issues, this method code the system calls with their first arguments to different their affected devices. The argument can be explained as one of the locations where cameras, GPS, microphones, or other devices that may involve sensitive information.

As shown in Figure 1, processes to generate aggregation dynamic features including two steps: combination and coding. In the combination step, network activities and system call sequences are merged into a mosaic according to timestamp sequence; in the coding step, the merged features are transformed into digital sequences.

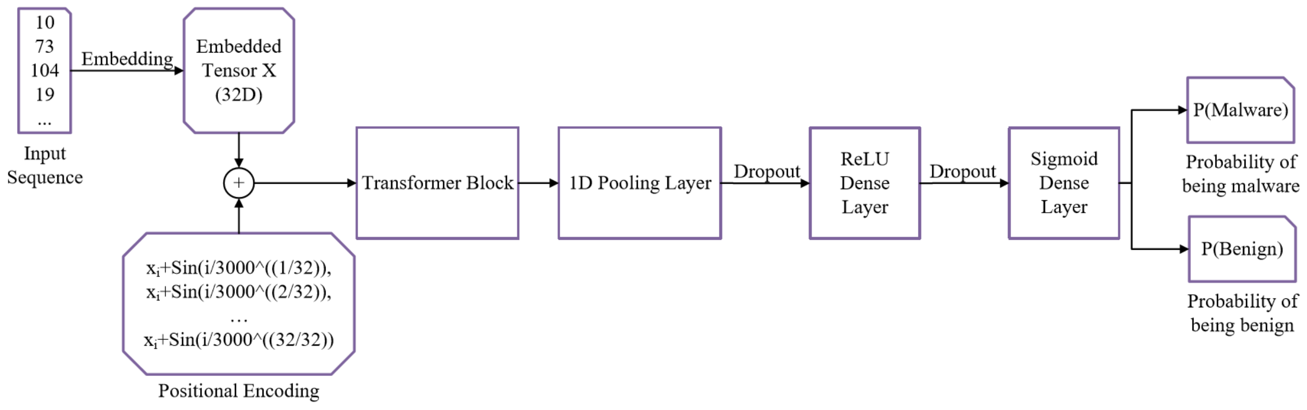


Fig. 2. Model overview.

B. *Model Based on Transform Structure*

This paper follows the methods of natural language processing to deal with dynamic feature sequences. The dynamic feature sequence is regarded as a piece of text, each item inside is a word, and the dynamic feature sequence with malicious activity is just like a piece of negative text. However, the dynamic feature sequence itself is different from natural language. The number of words in the dynamic feature sequence is only 176, and the length of the sequence is relatively long, ranging from one thousand to tens of thousands. To adapt to these differences in the dynamic feature sequence, it is necessary to modify the model for them.

The transformer structure is proposed by Vaswani et al. in 2017, which firstly aimed at settling with the information storing problem in the natural language processing area [11]. By using the attention mechanism exclusively, the model based on the transformer structure achieved both high performance and reactively low training cost, and it was proved to be an outstanding model structure.

The model used in this paper runs on attention mechanism and is applied with the transformer structure. The model consists of an input layer, an embedded layer with positional encoding, transformer blocks, a one-dimensional global average pooling layer, random deactivation layers (Dropout), sandwiched with Relu layers, a Sigmoid Dense layer, and a Softmax activation layer (output layer). Among them, the positional encoding used in this paper is the additive sine positional encoding method. The transformer block is a 32-dimensional 2-head structure, the length of the sequence accepted by the model is set to 3000, and the feedforward neurons used to adjust attention are set to 64. To adapt to the small alphabet size of the dynamic features, the embedded coding is specified as 32 dimensions. The final model has 111 646 parameters and sized only 1.4MB. The structure overview of the model is shown in Figure 2.

Due to the small number of samples and large sequence length, the gradient disappears frequently in the model of one-dimensional input and one-dimensional output. To solve the problem, another output is added to the model, and the two outputs represent the probability that the sequence belongs to malware or benign application. During the training, the 0-1 tag indicating whether it is malware is one-hot coded. The categorical cross-entropy is used as the loss function, and Adam optimizer is used to optimize learning.

IV. Evaluations

A. *Classification Performance*

To measure the classifying performance, the f-measure is selected as the evaluation criterion, and the f-measure is calculated according to the output value that characterizes the probability of malware and a threshold of 0.5 to determine the label. The formula for calculating the f-measure is shown as follows:

$$Fmeasure = 2 \times \frac{precision \times recall}{precision + recall} \qquad (1)$$

Among them, precision is calculated as:

$$precision = \frac{true\ positive}{classified\ positive} \qquad (2)$$

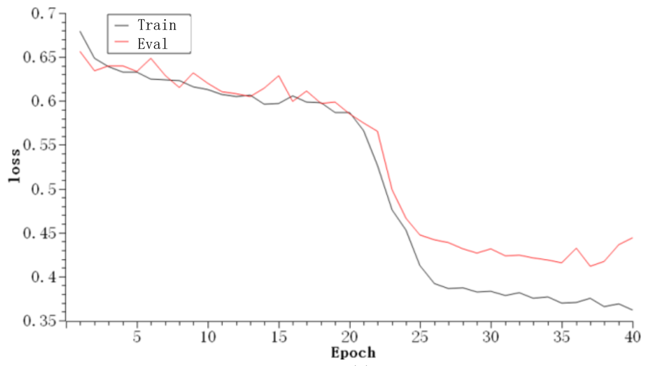
Precision is the fraction of successfully classified positive samples among all samples classified as positive. And recall is calculated as:

$$recall = \frac{true\ positive}{potential\ positive} \qquad (3)$$

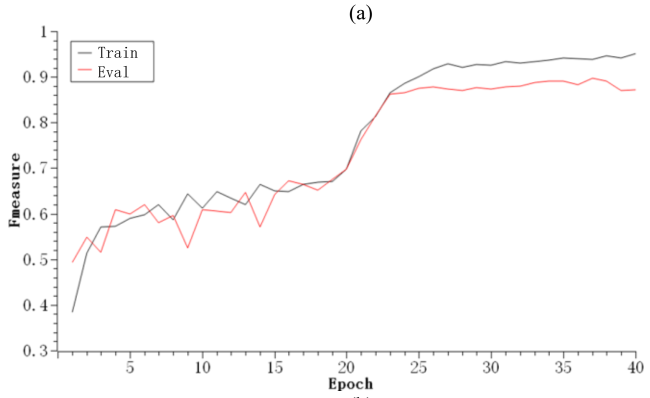
Recall is the fraction of the positive samples that are successfully classified.

To verify the effectiveness of this scheme, 2400 applications are randomly selected from the newly updated Androzoo dataset [12]. These applications are run on a virtual machine to extract their dynamic feature sequences. After the process of merging, coding, and sorting, 2210 sets of dynamic feature sequences are obtained, of which 917 are generated by malware, and the rest are generated by normal applications. the length of these feature sequences varies from 1000 to tens of thousands.

1768 of them are used as the training set and the remaining 442 are used as verification set. The model mentioned in the first section of this chapter is used for up to 40 epochs of training. The change of loss function and f-measure with the increase of epoch number is shown in Figure 3.



(a)



(b)

Fig.3. Change of loss and f-measure among epochs on train and evaluation sets: (a) loss-epoch curve, (b) f-measure-epoch curve.

As can be seen from Figure 3, the classifier performs well on fitting the features of both the training set and the test set. Although over-fitting begins to appear in the 22nd Epoch, the value of loss function on the training set and test set remained at a low level, and the corresponding classification performance remained stable. In the experiment, the highest f-measure obtained from the test set is 0.89.

An additional experiment is conducted based on data collected from real Android 10 mobile phones running the very same applications. Because of the complexity of the runtime environment of real Android phone devices, most of the malicious behaviors were not triggered when collecting features, and the f-measure yielded by the model drops to 0.74.

B. *Detection of Wild Malware*

Wild malware refers to the malware that has been discovered and detected in a recent period by several antivirus manufacturers, and cannot be recognized by other antivirus software yet. This malware often has new code characteristics, meantime the matching databases of the anti-virus software do not communicate with each other, so various anti-virus software on the market are often unable to detect this malware in a timely and comprehensive manner.

In this paper, the detection rate is selected as the standard to quantitatively evaluate the performance of detecting wild malware. The detection rate is calculated as follows:

$$Detection\ Rate = \frac{Detected}{Malware\ Samples} \times 100\% \qquad (4)$$

Where Malware Samples represents the total number of malware samples, and Detected represents the number of malware samples marked positive by the model.

As described in the research of Wang et al. [10], when they wrote their paper, the detection rate of wild malware by commercial antivirus software on the market was less than 50%, and the effect was not satisfactory. This is one of the main problems that this paper tries to solve.

In this paper, all the data obtained in the previous section are used to train the proposed model, which is used as the classifier in the integrated detection system. Then, 172 malware collected from VirusTotal [13] website are selected to test the classification performance.

We compare the proposed model with several commercial anti-virus software such as McAfee, Kaspersky, NOD32, Avast, TrendMicro, and Symantec, Comodo. The detection rates arranged from high to low are shown in Table 1.

TABLE I. DETECTION RATE AGAINST WILD MALWARE.

Rank	Name	Detection rate %
1	NOD32	65.3
2	Kaspersky	61.2
3	**Proposed Method**	**59.3**
4	McAfee	54.9
5	Symantec	47.9
6	Comodo	46.8
7	Avast	45.8
8	TrendMicro	45.6

The result shows that compared with the situation in 2017 [10], the anti-virus software on the market has made great progress in the detection of wild malware, with a detection rate of more than 45%. Among the best performers, NOD32 detected 65.3% of the malware. The detection rate of the detection scheme in this paper is 59.3%, ranking the third in the software used in this experiment, which is higher than commercial anti-virus software such as McAfee, Avast, TrendMicro, and so on.

V. CONCLUSION

This paper proposes a method to detect the dynamic features of Android malware. This method refines the system calls to extract additional key location access information, then organizes the network traffic and system call sequence in time order. Finally, it identifies the malware with the help of the simple and fast-training model based on the transformer structure.

So far as we are aware, this paper is the first attempt to apply the transformer structure to Android malicious behavior identification based on the system call sequence. The traditional detection method based on deep learning using traditional CNN or RNN requires large models and huge data sets. In contrast, the method proposed in this paper requires only a simple model with a size of 1.4MB, with fast training and predicting speed. f-measure of 89% is obtained, and the detection rate for wild malware is 59.3%, which is comparable to the commercial anti-virus software. Limited by the size of the data set, the model classification performance is not very amazing, but it still proves that the transformer model has great potential and unique advantages in this problem.

Future research works can be carried out in the following directions:

In the aspect of the model, with the help of Hook framework such as Tai-chi in the Android community, we can dynamically extract the system call sequence and network activity sequence on the device, which gives full play to the method's advantage of small model and less needed resources. Follow this approach, a set of dynamic malicious behavior alarm system which runs on Android mobile phone can be built. In the aspect of features, we can use the model structure that has been verified on its formal dynamic features to build a server-oriented high-precision detection system (or malware early warning system). This will contribute to reducing the threat of malware to Android devices.

Acknowledgment

This work is partly supported by the National Key Research and Development Program of China (2017YFB0802204), Key Research and Development Program for Guangdong Province, China (2019B010136001), and Basic Research Project of Shenzhen, China (JCYJ20190806142601687). Correspondence to: Yang Liu.

References

[1] StatCounter Mobile. (2019, Nov. 27). "Operating System Market Share Worldwide StatCounter Global Stats," [Online]. Available: http://gsa.statcounter.com/os-market-share/mobile/worldwide.

[2] 360 Fiber Labs. (2020, June 1). "Special Reports for Android Malware 2019," [Online]. Available: https://rs-beijing.oss.yunpan.360.cn/Object.getFile/360report/MjAxOeW5tOaJi+acuuWuieWFqOeKtuWGteaKpeWRii5wZGY=.

[3] W. Y. Lee, J. Saxe, and R. Harang, "SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks," in Deep Learning Applications for Cyber Security, M. Alazab and M. Tang, Eds. Cham: Springer International Publishing, 2019, pp. 197-210, doi:10.1007/978-3-030-13057-2_9.

[4] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in android byte-code through an end-to-end deep system," Future Generation Computer Systems, vol. 102, pp. 112-126, 2020/01/01/ 2020, doi:10.1016/j.future.2019.07.070.

[5] Q. Yu, X. Luo, and Z. Wang, "Dynamic Detection of Malicious Code on Android Based on Improved Multi-feature Gaussian Kernel," Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2019, pp. 221-228, doi:10.1007/978-3-030-32591-6_24.

[6] H. Alptekin, C. Yildizli, E. Savas, and A. Levi, "TRAPDROID: Bare-Metal Android Malware Behavior Analysis Framework," in 2019 21st International Conference on Advanced Communication Technology (ICACT), 2019, pp. 664-671, doi:10.23919/ICACT.2019.8702030.

[7] S. Wang, Z. Chen, Q. Yan, K. Ji, L. Wang, B. Yang, and M. Conti, "Deep and Broad Learning Based Detection of Android Malware via Network Traffic," in 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018, pp. 1-6, doi:10.1109/IWQoS.2018.8624143.

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in Symposium on Network and Distributed System Security (NDSS), 2014, vol. 14, pp. 23-26, doi:10.14722/ndss.2014.23247.

[9] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 8, pp. 3035-3043, doi:http://dx.doi.org/10.1007/s12652-018-0803-6.

[10] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," Journal of Network and Computer Applications, vol. 133, pp. 15-25, doi:10.1016/j.jnca.2018.12.014.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in the Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS), 2017, pp. 6000-6010, doi:10.5555/3295222.3295349.

[12] K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," in 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), 2016, pp. 468-471, doi:10.1145/2901739.2903508.

[13] Chronicle LLC. (May 17). "VirusTotal" [Online]. Available: https://www.virustotal.com/gui/.