Malware Detection by Merging 1D CNN and Bi-directional LSTM Utilizing Sequential Data

Seung-Pil W. Coleman and Young-Sup Hwang

Abstract Due to the popularity of the android platform, there is a growth in the number of devices and threats. For this reason, it is essential to build reliable tools that can detect malware android application packages (APK) on this platform. Creating effective models requires the use of rich features that are hard to generate. In this work, we extracted the Dalvik executable (.dex) byte-codes from APKs. Android application binaries are opcode sequences. Then, we trained one-dimensional convolutional Neural networks (CNN) using those sequential data. These one-dimensional CNNs detect local features and reduce the feature size. We went even farther to combine one-dimensional CNNs with a bi-directional long-short term memory network (LSTM) to detect malware. Experimental results show that our model, trained on a balanced number of samples, got an error rate of merely 5.4% on a dataset of 20,000.

Keywords Android malware detection · Data section · One dimensional convolutional neural network · Bi-directional LSTM · Sequential data

1 Introduction

The android platform is the most popular today, and it contains several hundred thousand applications in different markets. This has led to smartphones running on the Android operating system becoming a target for black hat hacker developers that have malicious intentions. Android is vulnerable compared to other platforms because it allows applications installation from multiple third-party markets. Recent studies have announced that mobile malware is finding new ways to hide, and the number of mobile malware seems to be increasing [1]. This is evident that there is a need to create a robust security solution.

S.-P. W. Coleman (✉) · Y.-S. Hwang
Department of Computer Science and Engineering, Sun Moon University, Asan, South Korea
e-mail: spil3141@naver.com

Y.-S. Hwang
e-mail: young@sunmoon.ac.kr

H. Kim et al. (eds.), *Information Science and Applications*, Lecture Notes in Electrical Engineering 739, https://doi.org/10.1007/978-981-33-6385-4_16

The most popular methods used for android malware detection are static and dynamic analysis. Static analysis is a technique widely used by researchers and industries. It involves the APK file being scanned before they can be executed on an android system. In such a case, the file is disassembled by a disassembler to obtain information such as API calls, permission lists, among others, which can then be examined. On the other hand, dynamic analysis involves methods that can monitor the behavior of applications at run-time. Some examples of this method are implemented by tools like TaintDroid [2], DroidRanger [3], and DroidScope [4]. Even though these are effective methods they have limitations. For example, even though dynamic analysis is effective at identifying malware, there is a caveat of overhead. And as for static analysis, it is fast and efficient but can easily be dodged by malware writers who can trick the disassemblers into producing incorrect code. This is accomplished by inserting errors into the source which leads to the actual code execution path being hidden or obfuscated. In this work, we choose to use the static analysis method because this method is essentially helpful on low-power and memory-limited devices such as Android devices. High optimization for performance is essential on the android operating system.

Data used to train deep learning models can come in the form of spatial, temporal, and more. Spatial data refers to location-aware information, a common example of this is a digital image. Temporal data are time-series that are collected as time progresses. These two concepts have been researched and powerful analysis tools in machine learning and deep learning have been created.

In this paper, we held the assumption that the DEX file binary, the bytes of the Dex file, is in the form of a time-series data [4]. The android binary file can be seen as containing sequences of opcode. We targeted the data section of the android application package (APK file). Our work is among the first to utilize CNN and RNN architectures. Particularly, one-dimensional convolutional neural network and bi-directional long-short term memory RNN.

The rest of this paper is arranged in the following way: we provided insight into previous works relating to this domain in Sect. 2. Our technique methodology is explained in Sect. 3. Experiments and Results with related information are presented in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Related Works

The number of researches relating to android malware detection has seen an increase since the discovery of deep learning as a possible alternative to older techniques. Deep learning applications relating to the areas of speech recognition, image classification, and natural language processing are among the several pioneers. Deep learning for android malware detection can be seen in [5] which was among the first to utilize deep learning for android malware detection. They extended the research work of [6] by employing long short-term memory (LSTM) on a large-scale. In [7], R2-D2

translates android apps into RGB (red, green, and blue) color code and transforms them into a fixed-sized encoded image for image classification.

In [8], they proposed an end-to-end solution using one-dimensional CNN, where both the spatial and temporal data features in bearing fault diagnosis are extracted and utilized. In this paper, we extended this approach in the field of android malware detection. Convolutional neural network (CNN) models, developed for image classification, can also work well on one-dimensional sequential data. In our case, this refers to the raw bytes of android applications, we extract features from sequential data and maps to a vector space. A one-dimensional CNN works the same way as two- or three-dimensional CNN. The difference is in the structure of the input data and the convolution kernel movement.

3 Methods

In this section, we describe the methodology of our research. The core of our approach can be divided into three parts. The first is the data section extraction, next is data preprocessing, and finally the model design Fig. 2.

3.1 Data Section Extraction

The goal of this step is to gather the raw bytes of the data section, embed it in a vector space, and group them to form the dataset. A DEX file contains many sub-sections shown in Fig. 1 among these are the header, string ids, data section, code item, etc. [11]. Our method extracts information on the data section binary from the header which contains information on every other section. This information includes the offset and size of the data section.

The tool used to disassemble the APKs was Androguard [9]. Android applications are developed in Java and compiled into optimized bytecodes for the Dalvik virtual machine. This bytecode can be directly accessed with the help of Androguard.

3.2 Data Preprocessing

In this step, the features were made suitable for the deep learning model. This involved padding the features to a fixed length and scaling it with normalization. Normalization is a feature scaling method to change the values of individual features to use a common scale (a range of 0 ~ 1) without distorting differences in the range of values or losing information.

The input shape of our model must be of dimension [timestep, feature]. However, the resultant dataset after the data extraction step from Sect. 3.1 contains elements of

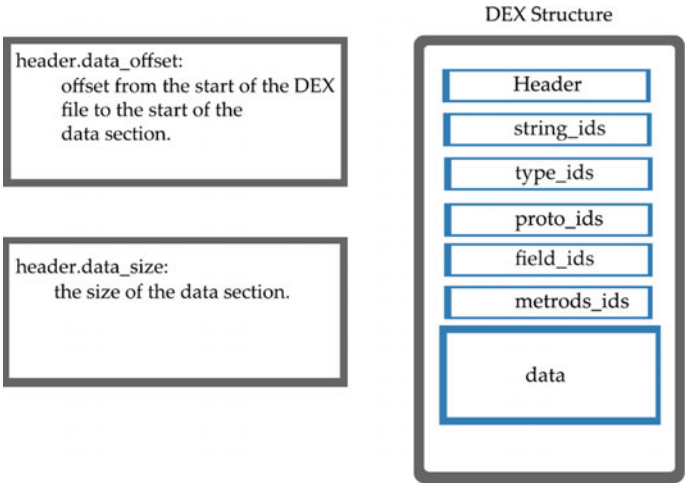

Fig. 1 Data section extraction

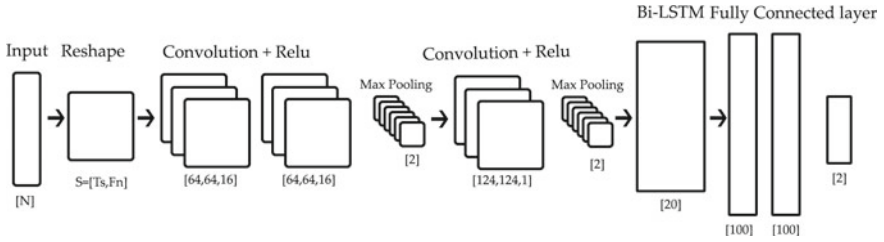

Fig. 2 Deep learning neural network architecture (one dimensional convolutional neural network and Bi-directional long-short term memory network)

various lengths. We reshape every sample in our dataset to achieve this fixed length. The process involved the use of a post-padding algorithm that appends the value of zero to the end of every sample less than a predefined threshold or cut if greater. Next, we used a window size to reshape the features into a sequential timestep. This new fixed input shape "S" and the window size can be defined as follows

$$S = [T_s, F_n]; \quad T_s = N/F_n; \quad F_n = 4 \tag{1}$$

where T_s is the timestep (window size), and F_n is the number of features per timestep for a given sample. For clarity, our final dataset has N equals ten million features, the new shape S becomes (2.5 million Timestep, 4 features). Before deciding on the number of features per timestep, other sizes were tested (4, 8, 16, etc.) but a timestep with 4 bytes gave the best performance.

3.3 Model Design

With the sample data padded and scaled, the next step is to decide on the malware detection architecture. Instead of using machine learning algorithms for pattern recognition and feature extraction, a deep learning solution was chosen. This work made use of the convolutional and recurrent deep learning architectures. As shown in Fig. 2, our model is composed of three one-dimensional CNN layers, a Bi-directional layer, and two dense layers (fully connected layers).

The one-dimensional CNN layers extract spatial and local temporal features from the sequences of normalized features. The pooling layers reduce the size of each feature map thus leading to a reduction in computational efficiency. The bi-directional LSTM layer extracts long-term temporal patterns that are analyzed by the fully connected layers. The fully connected layers then perform binary classification. Binary classification is accomplished by a classifier that can distinguish between two classes or labels.

4 Experiments

4.1 Dataset

All the data are from the following sources: Google play (the period between October 2016 and February 2017), Amazon, APK pure, AMD, and Drebin [10]. The malware samples are from the Drebin and AMD archives while the benign samples were combined samples from Amazon, Google play, and APK pure. The dataset we used contains 20,000 applications comprising of malware as well as benign android packages with a ratio of 1:1 (10,000 malicious applications and 10,000 benign ones). For the experiments, we used 19,000 samples for training, 500 for validation, and 500 for testing, summing up to a total of 20,000 (Table 1).

Table 1 Dataset sources

Source	Malware (%)	Benign (%)
Google play	0	20
Amazon	0	12.5
APK pure	0	17.5
AMD	35	0
Drebin	15	0

Table 2 Evaluation result from tested models

Models	Train accuracy	Test accuracy	Precision	Recall	F1-Score
3 × 1D Conv, 1 × Bi-LSTM	0.983	0.946	0.947	0.946	0.945
1 × 1D Conv, 1 × Bi-LSTM	0.91	0.902	0.9	0.9	0.9

Table 3 Experiment to determine F_n

Models	Epoch	Train accuracy	Test accuracy
(N/ F_n,16)	10	0.81	0.806
(N/ F_n,8)	10	0.84	0.832
(N/ F_n,4)	10	0.84	0.846

4.2 Experiment Result

The table below depicts the result of experiments undertaken to evaluate the performance of our deep learning model.

Table 2 shows the training accuracy, test accuracy, precision, recall, and F1-score of our final experiments. The table displays results from the two best models. The first is the 3 1D CNN layers architecture, and the other was a simpler model with a single convolutional layer. Our best model generalized on the test dataset with merely a 5.4% error rate. Also, this experiment proved that going deeper with convolutional layers yields better performance.

Table 3 shows the result of the experiment to determine the number of features per timestep. A simplified version of our dataset with 4, 8, and 16 feature channels enabled us to achieve our desire outcome. Each sample in this simplified dataset was made of merely 30,000 bytes. The model simplification involved reducing the layers and units per layer. Using 4 bytes per timestep gave the best testing model performance. Out of the result, we realized that larger feature channels did not correlate to better generalization.

5 Conclusion

In this work, we addressed the challenges of android malware detection through the introduction of a possible solution utilizing bi-directional LSTM and one-dimensional convolutional neural networks. Our method handles android binary files as sequences of opcode for malware detection.

In practice, this work demonstrates the process of analyzing both the temporal and spatial aspects of an android application for malware detection. To improve the achieved results, in future work, we plan to investigate methods that handle the large input size of our proposed model and the case of malware family detection.

Acknowledgements This research was supported by the National Research Foundation of Korea (NRF) and funded by the Ministry of Science and ICT (no. 2018R1A2B2004830).

References

1. McAfee Mobile Threat Report Q1 (2020)
2. Enck W, Gilbert P, gon Chun B, Cox LP, Jung J, McDaniel P, Sheth A (2010) Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of USENIX symposium on operating systems design and implementation (OSDI), pp 393–407
3. Zhou Y, Wang Z, Zhou W, Jiang X (2012) Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In: Proceedings of network and distributed system security symposium (NDSS)
4. Yan L-K, Yin H (2012) Droidscope: seamlessly reconstructing os and dalvik semantic views for dynamic android malware analysis. In: Proceedings of USENIX security symposium
5. Bilar D (2007) Opcodes as predictor for malware. Int J Electron Secur Digit Forensics 1(2):156–168
6. Vinayakumar R et al (2018) Detecting android malware using long short-term memory (LSTM). J Intell Fuzzy Syst 34(3):1277–1288
7. Hsien-De Huang TT, Kao H-Y (2018) R2-D2: color-inspired convolutional neural network (CNN)-based android malware detections. In: 2018 IEEE international conference on big data (Big Data). IEEE
8. Hao S et al (2020) Multisensor bearing fault diagnosis based on one-dimensional convolutional long short-term memory networks. Measurement:107802
9. Anthony D (2019) Androguard documentation. Release 3.4.0
10. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens CERT (2014) Drebin: effective and explainable detection of android malware in your pocket. In: Yan J, Yong Q, Qifan R (eds) NDSS, LSTM-based hierarchical denoising network for Android malware detection. Security and communication networks 2018, vol 14. pp 23–26
11. Chiossi, R. 2014. "A deep dive into DEX file format. https://elinux.org/images/d/d9/A_deep_dive_into_dex_file_format--chiossi.pdf