ConvDroid: Lightweight Neural Network based Andoird Malware Detection

Sifan Wu[0000−0001−5752−0929] and Xi Xiao[1111−2222−3333−4444]

Graduate School At Shenzhen, Tsinghua University, Shenzhen 518055, China
wusf18@mails.tsinghua.edu.cn
xiaox@sz.tsinghua.edu.cn

Abstract. The explosive amount of Android malware have threatened the security of legitimate users. In Recent years, with the development of the neural network, more and more research is focusing on detecting malware based on the neural network. Where, most of these techniques are depending on the complex feature engineering process and have a resource and time expensive classification neural network. In this paper, we propose a novel lightweight convolution neural network model, ConvDroid, with the system call sequences as features. Different from the existing n-gram models utilizing system call sequences, we construct a two-layer convolution neural network to learn the global information of sequences instead of limited windows. Furthermore, we assess ConvDroid for accuracy, efficiency using a dataset consisting of 3567 malicious applications and 3536 benign ones. Our experiments show that ConvDroid achieves an accuracy of more than 98% with a low time cost. We further demonstrate ConvDroid's superiority against state-of-the-art approaches.

Keywords: Android malware · Convolution neural network · System call sequence.

1 Introduction

As the increasingly development of mobile devices used around the world, more and more mobile applications have been used to facilitate people's life. The explosive growth of mobile communications has placed a huge burden on mobile security. A recent report[16] shows that the number of apps in the Google Play Store has risen from 1 million apps in July 2013 to 2.6 million apps in March 2019. Mcafees statistics[15] are reporting the average person having among 60-90 apps installed on their phone. With the Android market becoming the most popular platform due to its open architecture, it has also been the main target of malware.

To protect legitimate users from the attack of Android malware, it is critically important to prevent mobile phones from the serious and widespread Android malware.The conventional methods of malware detection is to store all malware characteristics and identify new malware by matching the patterns of stored malware features. A recent survey[8] studies a wide variety of malware's malicious characteristics and classified the existing malware detection methods into two categories, including static analysis and dynamic analysis methods. Static analysis methods[3, 9, 20, 1, 17, 14] leverage static characteristics such as static code, permission, to check whether an app contains information flow or calls sensitive API call, to match the patterns with known malware datasets. However, static analysis based malware detectors are vulnerable to obfuscated skills. Dynamic analysis methods detect malware with dynamic behavior[12, 20, 21, 6, 2, 22, 23]such as system call traces[7, 22, 23] ,resource usage[6, 10] etc at runtime. Dynamic analysis can prevent obfuscated skills with a complementary way to detect malware[5, 2].

Consequently, we wish to resolve the research question that *How to simplify the complex feature extraction and construct a light-weight malware detection model in the condition of ensuring the accuracy of malware detection system.*

Since system call can represent the actual intention of the app, to maximized simplify the extraction of malware feature and learn complementary semantic information of them, we regard system call sequences as the feature. In this paper, we conduct a light-weight deep learning malware detection approach which relies on dynamic analyze method based on the system call sequence. Instead of n-gram model, we analyze the whole sequences in an efficient convolution neuron network. Specifically, we extract the system call sequences by running the application to characterize its dynamic behaviors. There exists some semantic information in system call sequences which can be utilized to distinguish malware. Next we desire to learn a system from system call sequences, from which higher level representations will be constructed, the neural network can help to resolve this. However, there exist a number of challenges and differences for this domain that have not been encountered in common neural network tasks, which make the malware detection task intrinsically interesting and relevant from a machine learning perspective other than merely introducing these techniques to a novel domain. For Android application(APP) malware, these challenges include but not limited to:

– The byte of system call sequences contains multiple modalities of information. Each byte can be represented as human-readable text, binary code, etc.

- Treating each byte as a unit of the encoded sequences, we are dealing with a sequence classification problem more than two million time steps, which as far as I know, is far exceeds the length of input to previous neural network based sequence classifier.
- Android malware has multiple levels of concept drift over time. The level of API, build tools, and libraries developers will be periodically updated, which caused it difficult to identify novel variant malware.

As a whole, we make three primary contributions: Firstly, We analyze the system call sequences from a new point of view. Each system call is tread as a character. Converting the malware detection task into a long sequences classification problem. Secondly, Contrast to the n-gram based model or LSTM-based model, we conduct a novel network architecture that can successfully process the whole sequence of over two million steps, which can learn a complementary semantic information of system call sequence. Lastly, our model can achieve an excellent performance outperform previous Android malware detection systems, and only costs a little time.

The rest of this paper is organized as follows. Section II describes the related works about malware detection and neural network for long sequence. Section III introduces the methodology in detail. Section IV presents how this methodology is applied to analyze a real-world dataset and our evaluation. Section V concludes the paper and presents some future research problems.

2 Related Works

Recent years, there have been many research effort on developing automatic Android malware detection systems using machine learning techniques[1–3, 7]. In this section, we summary these methods used in malware detection and presents several state-of-the-art works related to our work.

Many Android malware detection detection methods have been proposed that rely on static[3, 9, 20, 1, 17, 14] or dynamic [12, 20, 21, 6, 2, 22, 23] analysis.

Drebin[3] apply machine learning methods combining several static features such as permissions, API calls and so on to provide a on-device approach to malware detection. Their evaluation results indicated that their proposed work can achieve high detection accuracy. DroidChain[20] also uses API call to analyze relationship among API calls in order to distinguish malware from trusted. GroupDroid[14] extract the Control Flow Gragh by static ananlysis and encode it to feature vector for measuring similarity. Appocopy[9] explores the specified semantics signatures from applications' control flows or data flows. DroidAPIMiner[1] used frequency analysis to identify critical sensitive APIs and then uses the set of critical APIs for classification. Opcode sequences are also taken as features for malware detection. Canfora G etc.[7] utilize opcode sequence which is disassemble from APK file to design a Hidden Markov Model.

For dynamic analysis, Wang et al.[21] uses a Android input generator to produce input specific to a dynamic tool. DroidCat[4] uses a diverse set of dynamic features based on method calls and inter-component communication (ICC) Intents achieved 97% F1-measure accuracy consistently for classifying apps evolving over dataset through nine years, which 16%-27% higher than two baseline presented in their work. ICCDetector[24] model ICC patterns to identify malware that exhibits different ICC characteristics from benign apps. DroidScribe[6] generat features at a different levels, including pure system calls and higher-level behavioral patterns like file system access which conflate sequences of related system calls. Some other works[7, 22, 23] log the kernel level activities such as system calls for dynamic analysis. MALINE[7] and BMSCS[22] perform dynamic analysis based n-gram model of system call sequences. Xiao et al.[23] utilize LSTM models to classify system call sequences.Shamsi et al.[18] cluster malware instance to discover similarities of dynamic API call sequences. Whereas, the n-gram model only consider relationships among limited windows, which causes a poor performance and other methods combine dynamic analysis and static analysis may have complex process of feature extraction and lead to expensive resource and low time efficiency.

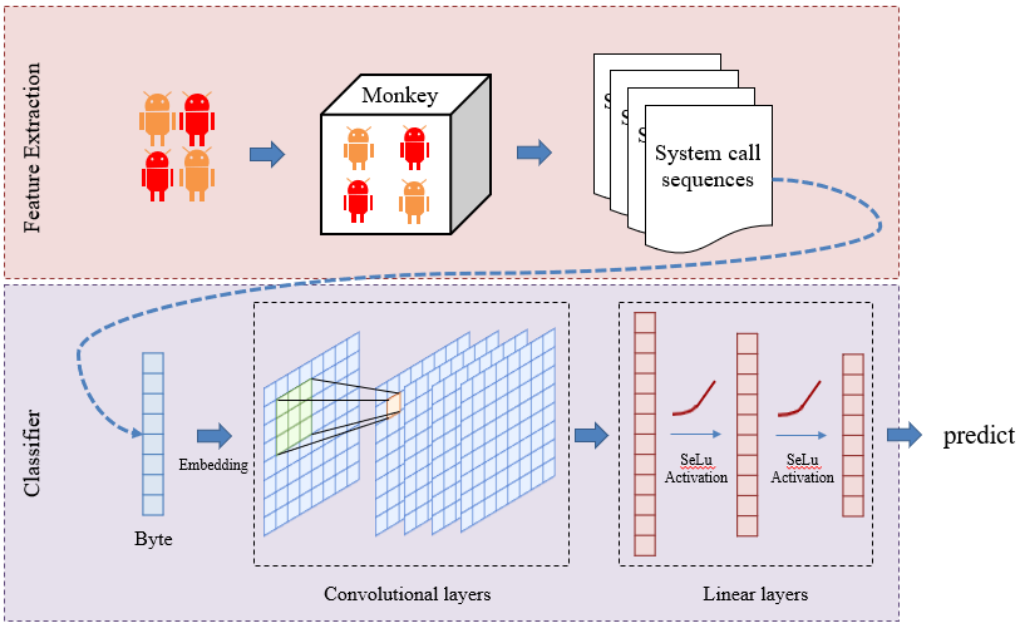
3 Proposed Method

In this section, we introduce the proposed method of our paper. Our deep learning based malware detection system is performed on system call sequences. As shown in Figure 1, our system consists of two major components: Feature Extractor and Convolution Neural Network(CNN) Based Classifier.

3.1 Feature Extractor

System call sequences are the execution trace of a mobile application, in which each element represents a system call being invoked by the application under analyze. In this work, instead of looking directly into the raw binary code of malware, we extract the system call sequences dynamically, which contain more about the behavior pattern of applications. Besides, we consider only the function name and discard all the parameters values to better analyze.

Fig. 1. Architecture of ConvDroid



After obtaining system call sequences, we encode each system call into a index number, in which the longest one has 196 system calls. Thus the encoded system call sequences can be treated as features for further classification.

3.2 CNN-Based Classifier

There are three challenges desired to overcome when design the classifier: 1) how to conceal with such a long sequence length in convolution neural network. 2) can we learn the information of both local and global sequence to improve the performance of n-gram model, which only consider limited windows size of system calls. 3) how to overcome the concept drift in malware, since the malware evolves frequently which caused it difficult for us to detect novel malware.

To resolve above challenges, we design the neural network based on convolution layers, which can iteratively extract more complex information effectively. Fig.1 illustrates the architecture of our model. The extracted system call sequence will be inputted in the neural network. Firstly, to learn the potential information of features, we utilize an embedding layer and multiple convolution layers so that we can deal with such long sequences. The activation function we chose is Rectified Linear activation function(ReLU), to improve the speed of convergence. After convolution layers, we construct a MaxPooling layer and Global Average Pooling layer to decrease the dimension of features. Next, we connect three linear layers of fully connected layer, which can explore high-level relationships information of the features. Finally, a sigmoid layer is used to output the label probabilities. All of parameters in the neural network except embedding layers are updated automatically by back propagation during training .

Embedding Layer. Before feeding into the neural network, we use an embedding layer to map each byte of system call to a fixed length feature vector. In this way we can avoid the raw byte value as it implies an interpretation that certain index values are intrinsically closer to each-other than other byte values, which we know the prior to be false, since the meaning of byte value is dependent on context. As a result, the length of sequences we feeded to the convolution part can achieve as long as two million.

This operation can be expressed as follow:

$$X_{emb} = X \times W \tag{1}$$

where X is the input, a fixed-length of system call sequences, W is the transition matrix with a size of $256 \times D$, which can transmit each byte of the sequence to a D-dimension value. Thus X_{emb} is the output of embedding layer with the size $N \times 200 \times D$, which can further used for convolution. Specifically, the parameter D is an important value associated with the classification results. In general, the more training data, the higher the dimension required[].

Convolution Layer. Automatically, adding more convolution layers is possible at the cost of large memory usage. We test our dataset on different number of convolution layers, with the index of convolution layers are numbered from 1 to L. The filter of each convolution layer is denoted as s_i in which i is the index of each convolution layer. The input of convolution network part is the output of embedding layer X_{emb} with the size of $N \times 200 \times 8$. The deeper convolution layer receive the output of the previous layers. After convolution layers, we use the activation function to preserve features and remove redundancies. We choose the rectified linear activation function(ReLU):

$$ReLU(z_i) = max(0, z_i) \tag{2}$$

where z_i is the output logits of convolution layer. Therefore the output of the first convolution layer is:

$$X_{conv_1} = ReLU(Conv_{\theta_1}(X_{emb})) \tag{3}$$

where θ_i is the weight and bias parameters of i-th convolution layer, which is learned from Back-Propagation. As a result, the output of convolution layer part is:

$$X_{conv_L} = ReLU(Conv_{\theta_L}(X_{conv_{L-1}})) \tag{4}$$

Pooling Layer. Following the last convolution layer, we connect two pooling layer to increase the robustness of network, one max-pooling layer and one global-average pooling. The max-pooling layer can keep the remarkable features and decrease the dimension of feature for the next layer. The Global-Average Pooling can help to regularize the network to prevent over-fitting. Specifically, max-pooling can generate a vector which consists of maximum value in each activation map in X_{conv_L} and global-average pooling average each filter map to an average value.

The output of max-pooling layer and global-average pooling is:

$$\begin{aligned} X_{maxpool} &= (max(X_{conv_L,1})|max(X_{conv_L,2})|...|max(X_{conv_L,m})) \\ X_{gloavepool} &= (gloave(X_{conv_L,1})|gloave(X_{conv_L,2})|...|gloave(X_{conv_L,m})) \end{aligned} \tag{5}$$

where m is the number of filter maps in the last convolution layer. Therefore the output size of $X_{gloavepool}$ is $1 \times 1 \times m$.

Linear Layer and Output Layer. Furthermore, we connect three full-connected linear layer after the global-average pooling layer. We push the output vector $X_{gloavepool}$ into full-connected layer to learn the relationships among features. We use SeLU[**?**] as the activation function here since SeLU can improve the robust of model and have a better performance than ReLU, which can be proved in the following experiments. The definition of SeLU is:

$$SeLU(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x <= 0 \end{cases} \tag{6}$$

Specifically, we define W_h and b_h as the weight and bias parameters of the fully-connected layer. Therefore the output of hidden layer is

$$H = SeLU(W_h X_{gloavepool} + b_h) \tag{7}$$

The result logits are then passed through the finally sigmoid layer for binary classification. The definition of sigmoid activation function is as follows:

$$y = \frac{1}{1 + e^{-H}} \tag{8}$$

where y is a probability value in the range of $(0, 1)$. When y is near 0, the sample is more like to be divided into benign and when y is around 1, the sample has bigger probability to be malware.

Loss Function Our optimization function use cross entropy loss function, which is a common and effective loss function for classification. Specifically, the cross entropy loss function is:

$$L_{ce} = -\sum_{i}^{N} q_i \log p_i \tag{9}$$

where q_i is the i-th sample's true label, and p_i is the corresponding predicted label. Our optimization objective is to minimize the cross entropy between predicted label and true label. And then we use back-propagation to update the parameter of neural network.

4 Experiments

In this section, we detail our experiments and results of proposed method to demonstrate the effecti.

4.1 Experimental Setting

We implemented ConvDroid in Python. To extract the system call sequence of applications, we leveraged Monkey[11], a dynamic analysis framework. The architecture of ConvDroid are built upon Keras framework, utilizing Tensorflow for the backend computations. The whole neural network computing is based on an NVIDIA 1080 Ti GPU.

The batch size of our model is 64 and the learning rate is 0.001 with Adam optimization. All network weights and biases were randomly initialized using the default initialization. Each byte of input sequence were first embeded to a 8-dimensional vector of the form $(\pm\frac{1}{16}, ..., \pm\frac{1}{16})$ according to its binary representation where constant 1/16 was found empirically.

4.2 Dataset and Metrics

Dataset In our experiments, the real-world dataset includes 3435 malicious applications and 3419 benign ones, in which 2841 malicious samples and 2842 samples for training and the rest for testing. The malicious applications are collected from Drebin dataset[3]. The trusted samples are crawled from Google Play market using an open-source crawler[13]. All the samples have been verified by VirusTotal[19] to make sure that malware samples were really malicious and benign samples are samples with $p = 0$ (p is the number of malware detection tool judged the sample to be malicious).

We gain 10 system call sequences trace of each application by running one application 10 times. Specifically, we configure Monkey to send 2000 random UI events in one minute and stimulating following events in Android operating system: (1)reception of SMS; (2)incoming call; (3)call being answered; (4)call being rejected; (5)GPRS; (6)HDSPA; (7)battery status; (8)boot completed. This list contains the most frequently trigger payload in Android malware[**?**,**?**]. As a result, we get 71030 system call sequences in total of 129 different system calls.

Metrics We define four metrics in general, Recall, False Positive Rate(FPR), Precision and F-score, defined as following respectively.

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

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{12}$$

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{13}$$

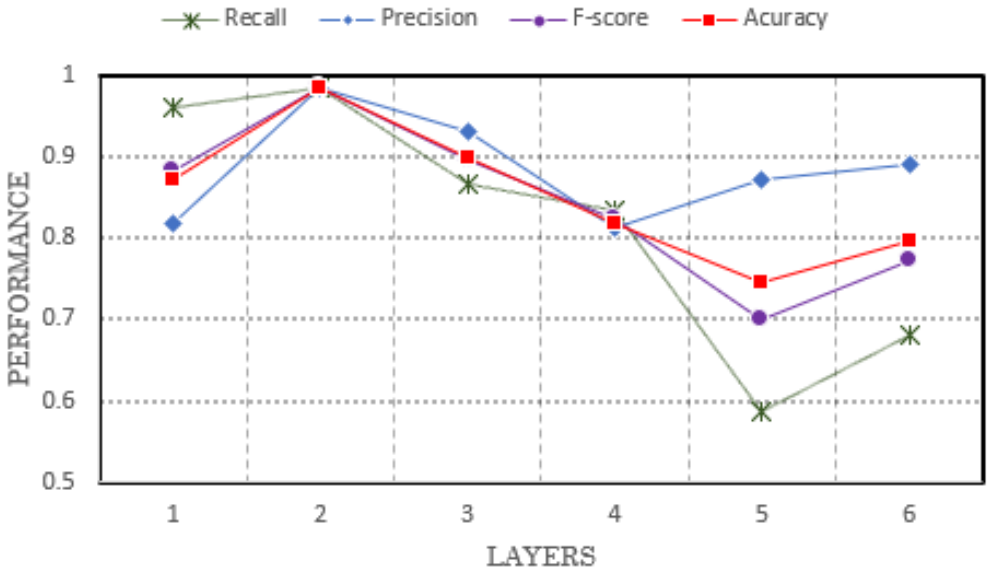
Here TP (true positive) is the number Android malware that are correctly detected, FN (false negative) is the number of Android malware that are not detected (predicted as benign application), TN (true negative) be the number of benign applications that are correctly classified and FP (false positive) is the number of benign applications that are incorrectly detected as malware. Recall indicate the accuracy of ConvDroid to detect malware correctly. And Precision is defined as the ratio of the number of true positive of the total number of items reported to be true. Accuracy is defined as the ratio of the number of samples which predicted correctly from the total samples. Similarly, F-score is defined as the weighted harmonic mean of Precision and Recall. Higher values of these metrics indicate higher efficiency of malware detection.

4.3 Effect of the Number of Convolution Layers

To learn the effect of the number of convolution layers, we analyze the relationship among the number of convolution layer and malware detection performance. In our experiment, we test the neural network with convolution layers from one layer to five layer, and the bigger result decays sharply, as shown in Fig. 2.

This plot illustrates the relationship between the number of convolution layers and performance of malware detector. Intuitively, we can learn that when there exists two convolution layers, the performance of neural network almost achieves the best with the highest recall of 98.63% and accuracy of 98.46%. When the number of convolution layer is less than two the neural netwoek cannot learn supplement information of the input. While if the number is larger than 2, it will caused the neural network to be over-fitting which leaded to a poorer performance.

Fig. 2. Performance of network with different number of convolution layers



4.4 Effect of the Parameters in Neural Network

We further analyze the effect of the parameters of convolution layer to malware detection performance. From above results, we choose the ConvDroid with two convolution layers. In this experiment, we change the kernel size, filters and activation of convolution layers with other parameters and structures stabled. Table 1 lists the result of five kinds of parameter settings of the two convolution layers, kernel size1, filter1 and kernel size2, filter2 indicate the kernel size and filters of the first convolution layer and second convolution layer respectively.

Table 1. Performance of network with different parameters of convolution layers

kernel size1	filters1	kernel size2	filters 2	Recall	Precision	F-score	Accuracy
8	32	16	32	0.9781	0.7873	0.8724	0.8548
8	64	16	64	0.9764	0.8394	0.9027	0.8933
16	32	16	64	0.9495	0.8319	0.8868	0.8770
16	64	16	64	0.9495	0.8522	0.9035	0.8958
32	32	16	32	0.9611	0.9724	0.9717	0.9800
32	64	16	64	0.9158	0.9205	0.9181	0.9172

Table 2. Performance of different activation function of linear layers

activation	Recall	Precision	F-score	Accuracy
relu	0.9057	0.9340	0.9197	0.9397
selu	0.9426	0.9818	0.9672	0.9757
tanh	0.9175	0.9348	0.9261	0.9257

As shown in Table 1, we obtain a best performance of 98.00% accuracy when the kernel size and filters of first convolution layer are both 32, the kernel size and filters of the second one are 16 and 32 respectively. When the kernel size of first convolution layer is 8, the performance is the worst of all. The reason is that dealing with such a long sequence, we need a bigger enough receptive field for the contextual information. From Table 2 we can see that ReLU instead of SeLU in the linear layers can have 4% relative drop.

4.5 Comparison with Other Approach

We compared our approach with other three malware detection models. Marko[7], BMSCS[22] and Xiao et al.[23] are typical malware detection methods utilizing system call sequences. Respectively, Marko leverage the relationship among system calls and Xiao use a LSTM classier for system call sequences. We evaluate them with the same dataset as ours. The result metrics are listed in follwing Table 3.

Table 3. Comprision with other methods

	Recall	Precision	F-score	Accuracy
Marko[7]	0.8796	0.8241	0.8509	0.8453
Xiao [23]	0.9132	0.9477	0.9301	0.9311
BMSCS[22]	0.8711	0.8099	0.8394	0.8326
ConvDroid	0.9863	0.9389	0.9620	0.9846

As shown in Table 3, ConvDroid shows an excellent performance than other state-of-art results in Recall, F-score and Accuracy, only Precision little lower than Xiao's result. We can draw a conclusion that ConvDroid is outperform than other system call-based methods.

4.6 Runtime Efficiency

To support a high-performance malware detection, we further analyze the time cost of ConvDroid including the time of building the classifier and classification. In this section, we test ConvDroid on different network architecture. The results (See Fig. 3) indicate that the larger the number of convolution layers the shorter of the time cost. The reason is that when the number of convolution layers is getting larger, the learning ability of neural network becomes better which lead to a quicker convergence speed. On the other hand, when the number of convolution layers is stabled, the kernel size and layers have a positive correlation with the runtime, since when the kernel size and layers getting larger, the number of parameters in the neural network increasing drastically, caused a slower convergence speed.

Fig. 3. Runtime Efficieny of different number of convolution layers

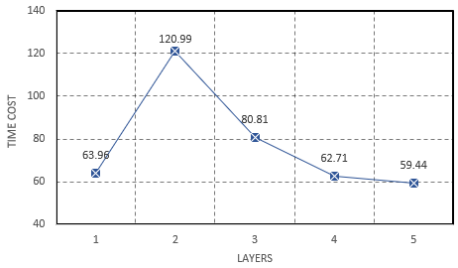


Table 4. Runtime Efficieny of different number of convolution layers

kernel size1	filters1	kernel size2	filters2	time(s)
16	32	16	64	99.44
32	32	32	64	165.84
64	64	32	64	350.49

5 Conclusion

In this paper, we propose a novel malware detection system ConvDroid that leverage system call sequences and lightweight convolution neural network to detection malware. Our proposed method enables neural network to learn the behaviors of malware and achieve a high performance at low time cost. Compared to other work, our detection system is more lightweight and outperform than state-of-the-art results.

References

1. Aafer, Y., Du, W., Yin, H.: Droidapiminer: Mining api-level features for robust malware detection in android. In: International conference on security and privacy in communication systems. pp. 86–103. Springer (2013)
2. Afonso, V.M., Amorim, M.F., Gregio, A.R.A., Junquera, G.B., De Geus, P.L.: Identifying android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques **11**(1), 9–17 (2015)
3. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Effective and explainable detection of android malware in your pocket (2014)
4. Cai, H., Meng, N., Ryder, B., Yao, D.: Droidcat: Unified dynamic detection of android malware. Tech. rep., Department of Computer Science, Virginia Polytechnic Institute & State (2016)
5. Chen, S., Xue, M., Tang, Z., Xu, L., Zhu, H.: Stormdroid: A streaminglized machine learning-based system for detecting android malware pp. 377–388 (2016)

6. Dash, S.K., Suareztangil, G., Khan, S.J., Tam, K., Ahmadi, M., Kinder, J., Cavallaro, L.: Droidscribe: Classifying android malware based on runtime behavior pp. 252–261 (2016)
7. Dimjasevic, M., Atzeni, S., Ugrina, I., Rakamaric, Z.: Evaluation of android malware detection based on system calls pp. 1–8 (2016)
8. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M., Rajarajan, M.: Android security: A survey of issues, malware penetration, and defenses. IEEE Communications Surveys and Tutorials **17**(2), 998–1022 (2015)
9. Feng, Y., Anand, S., Dillig, I., Aiken, A.: Apposcopy: Semantics-based detection of android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 576–587. ACM (2014)
10. Galal, H.S., Mahdy, Y.B., Atiea, M.A.: Behavior-based features model for malware detection. Journal of Computer Virology and Hacking Techniques **12**(2), 59–67 (2016)
11. Google: Ui/application exerciser monkey (2013), http://developer.android.com/tools/help/monkey.html
12. Grace, M.C., Zhou, Y., Zhang, Q., Zou, S., Jiang, X.: Riskranker: scalable and accurate zero-day android malware detection pp. 281–294 (2012)
13. liato: android-market-api-py (2017), https://github.com/liato/android-market-api-py
14. Marastoni, N., Continella, A., Quarta, D., Zanero, S., Preda, M.D.: Groupdroid: automatically grouping mobile malware by extracting code similarities. In: Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. p. 1. ACM (2017)
15. Mcfee: Mcafee mobile threat report,2019 (2019), https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf
16. Play, G.: Number of available applications in the google play store from december 2009 to march 2019 (2019), https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
17. Saracino, A., Sgandurra, D., Dini, G., Martinelli, F.: Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing **15**(1), 83–97 (2018)
18. Shamsi, F.A., Woon, W.L., Aung, Z.: Discovering similarities in malware behaviors by clustering of api call sequences. pp. 122–133 (2018)
19. Total, V.: Virustotal-free online virus, malware and url scanner. Online: https://www. virustotal. com/en (2012)
20. Wang, Z., Li, C., Yuan, Z., Guan, Y., Xue, Y.: Droidchain: A novel android malware detection method based on behavior chains. Pervasive and Mobile Computing **32**, 3–14 (2016)
21. Wong, M.Y., Lie, D.: Intellidroid: A targeted input generator for the dynamic analysis of android malware (2016)
22. Xiao, X., Wang, Z., Li, Q., Xia, S., Jiang, Y.: Back-propagation neural network on markov chains from system call sequences: a new approach for detecting android malware with system call sequences. IET Information Security **11**(1), 8–15 (2016)
23. Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K.: Android malware detection based on system call sequences and lstm. Multimedia Tools and Applications **78**(4), 3979–3999 (2019)
24. Xu, K., Li, Y., Deng, R.H.: Iccdetector: Icc-based malware detection on android. IEEE Transactions on Information Forensics and Security **11**(6), 1252–1264 (2016)