2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering

Deep Android Malware Classification with API-based Feature Graph

Na Huang*, Ming Xu*†, Ning Zheng✉*†, Tong Qiao*, Kim-Kwang Raymond Choo‡
*School of Cyberspace, Hangzhou Dianzi University, HangZhou, China
†School of Computer Science and Technology, Hangzhou Dianzi University, HangZhou, China
‡Department of Information Systems and Cyber Security, The University of Texas at San Antonio, TX 78249, San Antonio, USA

***Abstract*— The rapid growth of Android malware apps poses a great security threat to users thus it is very important and urgent to detect Android malware effectively. What's more, the increasing unknown malware and evasion technique also call for novel detection method. In this paper, we focus on API feature and develop a novel method to detect Android malware. First, we propose a novel selection method for API feature related with the malware class. However, such API also has a legitimate use in benign app thus causing FP problem (misclassify benign as malware). Second, we further explore structure relationships between these APIs and map to a matrix interpreted as the hand-refined API-based feature graph. Third, a CNN-based classifier is developed for the API-based feature graph classification. Evaluations of a real-world dataset containing 3,697 malware apps and 3,312 benign apps demonstrate that selected API feature is effective for Android malware classification, just top 20 APIs can achieve high F1 of 94.3% under Random Forest classifier. When the available API features are few, classification performance including FPR indicator can achieve effective improvement effectively by complementing our further work.**

***Index Terms*—Android malware, feature selection, Structure analysis, CNN-based classifier.**

I. Introduction

With the rapid development of Internet, smartphone plays more and more important role in people's daily life. As one of smartphone operating system, Android currently occupies 82.8% of smartphone market[1]. Many examples of Android malwares(e.g., Zsone, Spitmo) have been released to the public and cause security threat, including privacy leak and property loss. Zsone [1] subscribed to paid content by sending text messages without user awareness while Spitmo [2] stole passwords of bank account by injecting into apps. Depending on a report released by 360 Internet Security Center[2], 7,573 million new Android malwares were intercepted in 2017 with an average of 21,000 per day. Android malwares increase explosively and pose a serious security threat to users thus it is very important and urgent to detect Android malwares effectively.

The primary detection method but widely used in industry area is based on signature, which detected malware by matching signature. But this kind of method is limited to detect unknown malware. In order to detect malware intelligently and scalably, machine learning is a great idea. Searching for interpretable and discriminative feature is critical in machine learning method. Many researches [3]–[6] have demonstrated effectiveness of API feature in detecting Android malware for reason that API might reflect fine-grained behavior of Android app. They preferred to select API related with probable malicious behavior to distinguish malware from benign. [7] summarized thousands of sensitive APIs related with dangerous permission. But they may be partially outdated nowadays. [3] selected API which is used more frequently in malware dataset than benign dataset. But it might select API rarely occur in malware. [5] selected restricted, used permission and suspicious API calls as a subset of feature set. But it is not easy and time consuming for non-domain expert to recognize such APIs. For the above analysis, we propose a novel selection method for API feature related with the malware class in this paper, which harmonically considers distribution of API using in malware and benign dataset. We prefer the API that not only occurs frequently in malware dataset but less frequently in benign dataset. The naive idea is that feature related with a class should have a frequent occurrence in the class but unmeant occurrence in other class.

Taking further consideration, the API related with malware also has a legitimate use in benign thus causing FP problem (misclassify benign as malware). Hence we explore structure relationship between APIs to obtain further difference, that is APIs occur in the same API block or nearby API blocks have invoking relationship. APIs in the same API block might be gang for one malicious behavior. Besides, malicious behavior generally is not conducted in a single block but more blocks thus APIs in two nearby API blocks are also suspicious. In classification, APIs and their two structure relationships are mapped to a matrix interpreted as input API-based feature graph hand-refined for CNN-based classifier. A convolution filter perceives a subregion of the feature graph at a time and extracts a subset of the API-based features, which is more refine than the single API feature.

The main contributions of this paper are as follows:

- We propose a novel selection method for API feature

✉ Corresponding author: Ning Zheng(*E-mail: nzheng@hdu.edu.cn*).
[1]https://www.idc.com
[2]https://www.360.cn

related with the malware class advancing in effectiveness and not requiring domain knowledge, which harmoniously considers occurrence frequency in malware dataset and occurrence frequency difference between malware and benign dataset. The naive idea is that feature of a class should be a frequent occurrence in the class but unmeant occurrence in other class.
- In order to solve the FP problem caused by only using API feature related with the malware class, we further explore two suspicious structure relationships between API features. API feature and their structure relationship are mapped to a matrix interpreted as the hand-refine feature graph for the designed CNN-based classifier.
- We conduct detailed experiments on a real-world dataset consisted of 3,697 malware apps and 3,312 benign apps. The results show that our selected API feature is effective for Android malware detection under shallow classifier. When the available API features are few, classification performance including FPR indicator can achieve effective improvement effectively by complementing our further work based on the API feature.

The remainder of this paper is organized as follows. Section II discusses related work, Section III shows the overview of the proposed method, Section IV and Section V elaborate feature extraction and classification model respectively. Evaluation is in Section VI, conclusion and discussion are followed in Section VII.

II. Related Work

To detect Android malware, the primary detection method is based on the signature. For instance, [8] used Java code and classes as signatures to detect malware. However, the signature-based method is restricted to detect unknown malware. In fact, there are a lot of variant or novel malwares in the real world. Hence researchers have proposed many other methodes, we classify them related our work into three types of method as follows.

A. Static feature combination based method

Many researches focused on intelligent Android malware classification with machine learning technology in recent years. They extracted discriminative static or dynamic features from Android app and trained classifier to predict whether the app is malware or benign. Extracting dynamic feature is complex and time consuming, because each of the malware sample must be executed within a secure environment for a specific time for monitoring the behavior [9]. Among different kinds of static features, permission as preliminary security mechanism of Android system is widely used in many researches. However, [10] deemed intent feature is more effective than permission and [3] deemed that API feature is also more effective than permission. It can be explained that the intent and API are more fine-grained than permission in reflecting app behavior. Combining different types of static features and training shallow classifier is a kind of prevalent way [11]–[13]. [11] extracted API, permission and intent message through static analysis on Android apps and k-means clustering for these feature before the k-NN classification. However, this kind of method is easy for attackers to evade. The increasing variant or new malware and evasion technique call for novel detection method. In this paper, we focus on static API feature and develop a novel way to detect malware, which can extract refine feature and is more difficult for an attacker to evade.

B. Structure based method

Our work is also related with the method based on structure analysis. [6] implemented a prototype system called DroidSIFT, which constructed weighted contextual API dependency graphs and classified Android malware by graph matching. [14] proposed an method to detect Android piggybacked apps through sensitive subgraph analysis, which extracted five features from the most sensitive subgraph and fed them to random forest model. [15] implemented an automatic system for classifying Android malware according to fregraph, which constructed sensitive API related graph and conduct community detection to divide the sensitive API related graph into subgraph and classify Android malware family with subgraph matching. Subgraph is a kind of refine feature than a single API. These researches need designing a complex algorithm to generate subgraph manually and causing a nuisance. In this paper, we only perform structure analysis and obtain two structure relationships between APIs based on API blocks without subgraph partition. Instead, we design a CNN-based classifier to extract refine feature and detect the app is malware or benign.

C. Deep learning based method

Recently, deep learning method has shown state-of-the-art performance for malware classification. They extracted features suitable for deep learning model through static or dynamic analysis of Android app. [16] designed a Convolutional Neural Network for system API-call sequences classification. [17] first trained Recurrent Neural Network to extract features of process behavior and then trained the Convolutional Neural Network to classify feature images which are generated by the extracted features from the trained RNN. [18] designed a method based on convolutional neural network applied to syscalls occurrences through dynamic analysis. [19] used more than 200 features extracted from both static analysis and dynamic analysis of Android app and adopted deep belief network as the pre-trained neural network. [20] built deep learning model to learn features from five types of features through static analysis of Android apps and then feed them to the SVM model. [21] proposed to convert opcodes sequences into images and used convolutional neural network to detect malware. Different from these work, we extract API features related with the malware class and their structure relationship, map them to a matrix interpreted as the API-based

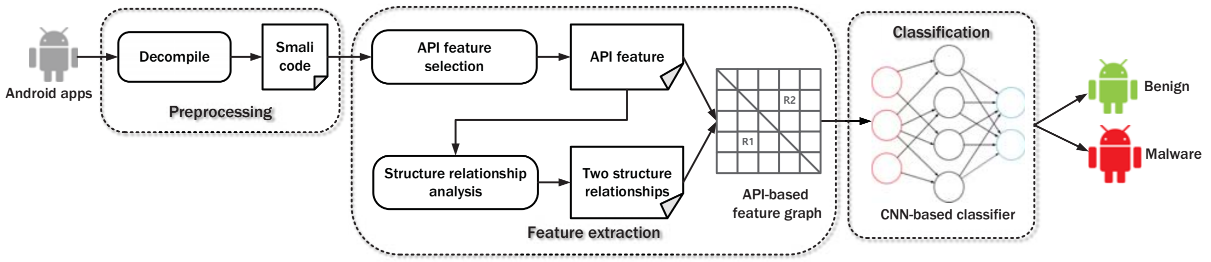


Fig. 1: The framwork of proposed method.

feature graph hand-refined. A Convolution Neural Network based classifier is designed for the API-based feature graph classification.

III. Overview

The framework of our proposed method is shown in Fig. 1. It mainly includes three phases, preprocessing, feature extraction and classification. In the preprocessing phase, we use Apktool tool[3] to decompile the dex file of Android App into readable Smali code. And in the feature extraction phase, we extract API related with the malware class and perform structure analysis to obtain two structure relationships between them by traversing the Smali code. Finally in the classification phase, API features and their two structure relationships are mapped to a matrix as input feature graph hand-refined for CNN-based classifier to detect whether the app is benign or malware. We mainly elaborate feature extraction part and classification part in the following.

IV. Feature Extraction

A. API feature selection

In Android malware classification, we are more concerned about features related with the malware class. However, using traditional feature selection method like mutual information, the selected feature is not strictly related with the malware class. This paper proposes a novel selection method for API feature related with the malware class, which harmonically considers distribution of API using in malware and benign dataset. Based on two considerations, one is the occurrence frequency in malware dataset and the other is the occurrence frequency difference between malware and benign dataset. The naive idea is that feature of a class should have a frequent occurrence in the class but unmeant occurrence in other class. Taking the two considerations into account, each API can get a relation coefficient from 0 to 1. We use following terms to evaluate the relation coefficient of API S_i.

- CM(S_i): The number of apps using S_i in malware dataset.
- CB (S_i): The number of apps using S_i in bengin dataset.
- MR(S_i): The ratio of apps using S_i to the total number of malware. MR(S_i) is calculated with Equation (1).

[3]https://ibotpeaches.github.io/Apktool/

$$MR(S_i) = \frac{CM(S_i)}{TM(S_i)} \tag{1}$$

In Equation (1), TM is the total number of malwares. From probability point of view, MR can be interpreted as conditional probability of S_i used in malware, namely P(S_i|Mal).

- BR(S_i): The ratio of apps using S_i to the total number of bengins. BR(S_i) is calculated with Equation (2). BT is the total number of bengins.

$$BR(S_i) = \frac{CB(S_i)}{BT(S_i)} \tag{2}$$

- MBR(S_i): This term measures the occurrence frequency difference of S_i between malware and bengin. MBR(S_i) is calculated with Equation (3)

$$MBR(S_i) = \frac{MR(S_i)}{MR(S_i) + BR(S_i)} \tag{3}$$

For Equation (3), MR and BR can be obtained by Equation (1) and Equation (2) respectively. MBR can measure occurrence frequency difference of S_i using between malware and bengin. From the probability point of view, MBR can be interpreted as conditional probability that how many of total apps using S_i belong to malware, namely P(Mal|S_i).

There are a few problems if only consider the MR or MBR. On the one hand, high MR of the S_i is not enough to guarantee its high relevance for reason that benign apps might also use these APIs frequently. On the other hand, only consider MBR of S_i might select some APIs rarely used in malware and bengin. In that two cases, selected API feature cannot effectively distinguish malware from benign. We use harmonic average function to integrate the MR and MBR and obtain the relation coefficient. Furtherly, we can allocate weights to set bias for MR and MBR. For the above analysis, relation coefficient of API S_i is calculated as follows.

$$RC(S_i) = \frac{(1+\alpha^2) \times MBR(S_i) \times MR(S_i)}{\alpha^2 \times MBR(S_i) + MR(S_i)} \tag{4}$$

In Equation (4), parameter α measures relative importance of MR to MBR. MR has a greater impact when $\alpha > 1$ while MBR has a greater impact when $\alpha < 1$. In particular, MR can hardly affect the final result if α is very small, which is similar to select API feature only with occurrence difference

TABLE I: Structure relationship difference of three API pairs in malware and benign

API Pair	Malware			Benign		
	Total apps	Structure related	Ratio	Total apps	Structure related	Ratio
sendTextMessage-euquals	2046	1307	63.8%	118	15	12.7%
sendTextMessage-valueOf	2019	336	16.6%	118	11	9.3%
equals-valueOf	2079	708	34.0%	565	7	1.2%

frequency between malware and benign. By the same token, MBR can hardly affect the final result if α is very big, which is similar to select API feature only with its occurrence frequency in malware dataset.

B. Structure relationship analysis

API feature related with the malware class also has legitimate use in benign thus causing FP problem (misclassify benign as malware). In addition, simple feature pattern of using API or not is easy to be evaded by an attacker. So we perform structure analysis based on API block to explore structure relationships between APIs and obtain further difference. In this paper, a segment that begins with ***.method*** and ends with ***.end method*** in Smali code is called an API block. As is shown in Listing 1, there are two API blocks of a malware in our collected dataset, one is ***send*** and the other is ***activate***.

Listing 1: An example of two API block

```
Block1:
.method public send
  invoke−static , Ljava/lang/String;−>valueOf
  invoke−virtual/range, Landroid/telephony/
       SmsManager;−>sendTextMessage
   ...
.end method

Block2:
.method public activate
  invoke−virtual , Ljava/lang/String;−>equals
  invoke−virtual , Lcom/software/application/Msg;−>
       send
   ...
.end method
```

We mainly analyze two structure relationships based on API block. One is APIs occur in the same API block, which is also used in [4]. For example, *valueOf* and *sendTextMessage* have the first relationship since they occur in the same block *send*. APIs in one API block might be gang for one malicious behavior. Generally, malicious behavior is not implemented in a single block but more blocks. So APIs occur in nearby API blocks that have invoking relationship is also suspicious. For example, *equals* and *sendTextMessage* have the other relationship. Taking the three APIs as an example to show the structure relationship difference between malware and benign. A statistic on our collected dataset consisted of 3,312 benigns and 3,697 malwares is shown in TABLE I. 2,046 malwares use API pair of *sendTextMessage* and *equal*, of which 1,307 malwares have the structure relationship between the API pair. And there are also 118 benigns using the API pair of *sendTextMessage* and *equal*, but only 15 benigns have the structure relationship between them. Thus it can be seen that structure relationship can provide further difference for us to distinguish malware from benign in this case.

C. API-based feature graph

After obtaining API features and two structure relationships between them, we map them to a matrix R as input feature graph for CNN-based classifier.

We divided the R into three parts, diagonal part, lower triangular part and upper triangular part. The diagonal part records the API feature using, lower triangular part records the first structure relationship and upper triangular part records the second relationships. The eigenvalue is 1 if the corresponding feature exists otherwise is 0. After that, API using knowledge of app is fully conveyed in the R. For CNN-based classifier, R can be interpreted as an API-based feature graph hand-refined. When using CNN-based classifier for the API-based feature graph classification, a convolution filter perceives a subregion of the feature graph at a time and extracts a subset of the API-based features, which is more refine than the single API feature.

V. Classification

We design a CNN-based classifier for the API-based feature graph classification. The architecture of CNN-based classifier is shown in Fig. 2. It is mainly consisted of feature selection phase and classification phase. The feature selection phase only employs one convolution layer but has many convolution filters. Each convolution filter slides step by step on the feature graph to generate a neuron matrix. After convolution, ReLU activation function [22] which is denoted as Max(0, x) is employed due to its low computation of learning process.

Convolution layer seems to select features with an idea of global search, because every subregion of input is explored by sliding convolution filter. However, not all subregions are effective for classification task hence pooling layer is connected behind convolution layer to select critical neurons.

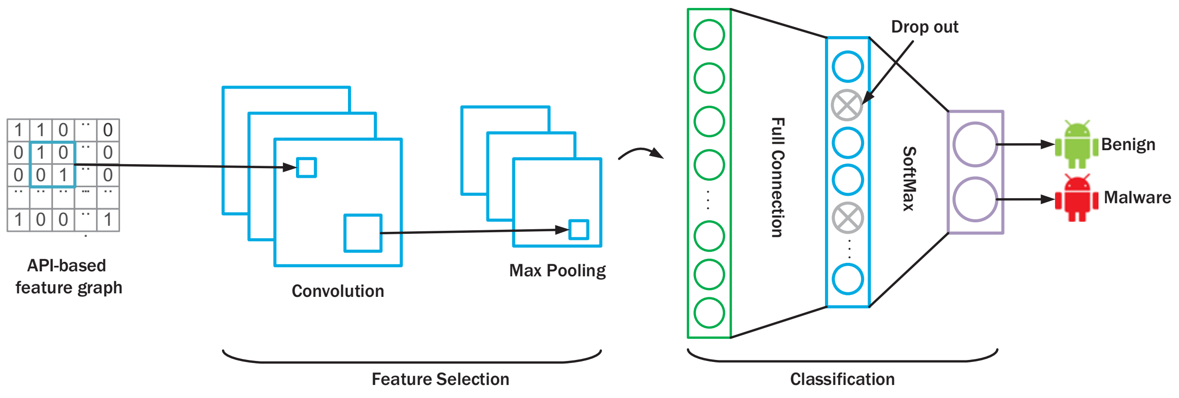


Fig. 2: The architecture of CNN-based classifier.

Classification layer is consisted of full connected hidden layer and full connected output layer. Neurons arrange with form of a matrix in the convolution layer and they need to be flattened before the classification layer. In order to avoid over-fitting, we design a dropout regularization rate p% after the full connected hidden layer, which means neurons with a probability of (1 – p%) are abandoned [23]. SoftMax classifier is applied in the full connected output layer and output normalized prediction probabilities of malware and benign class.

During the process of learning, we use cross-entropy [24] as loss function and ADAM [25] as optimization algorithm.

VI. Expriment

In this section, we first introduce our dataset collected from real-world and then do three sets of expriment to evaluate our method based on the dataset. The first set of expriment evaluates our selection method for API feature (Section IV-A). The second set of expriment evaluates our overall method proposed. And run-time overhead of feature extraction is calculated in the end.

A. Expriment setup

Our method is evaluated on a real-world dataset containing both Android malware and benign apps. The malware dataset contains 3,697 apps collected from virusshare[4]. And the benign dataset is collected from Google Play[5] and contains 3,312 apps from 32 different categories. All experiments are performed under the environment of 64 bit Windows 10.0 operating system with Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHZ plus 8G of RAM. All our evaluation performance indicators are shown in TABLE II.

[4]https://virusshare.com
[5]https://play.google.com/store/apps

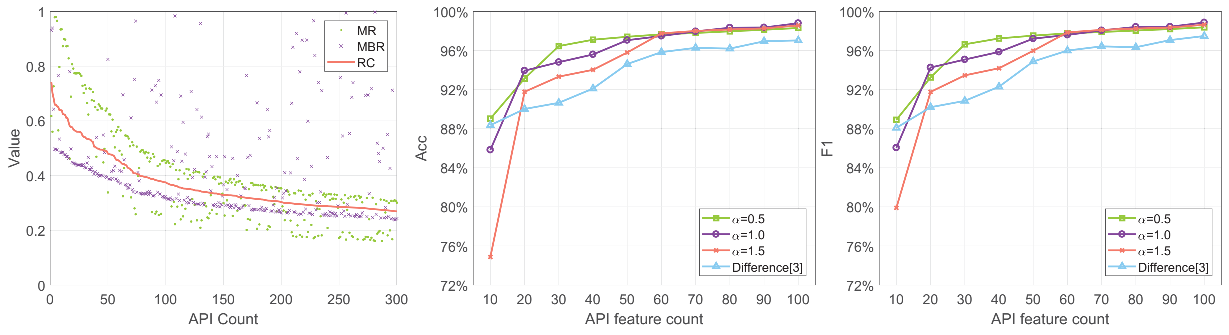
TABLE II: Descriptions of the used indicators

Indicators	Abbr.	Descrptions
True Positive	FP	Number of apps misclassified as malware
True Positive	TP	Number of apps correctly classified as malware
False Negative	FN	Number of apps misclassified as bengin
True Negative	TN	Number of apps correctly classified as bengin
Accuracy	Acc	(TP+TN)/(TP+TN+FP+FN)
Recall Rate	R	TP/(TP+FN)
FP Rate	FPR	FP/(FP+TN)
Precision	P	TP/(TP+FP)
F-measure	F1	2PR/(P+R)

B. Evaluation of our selection method for API feature

As introduced in Section IV, each API can get a MR, MBR and RC value. Fig. 3 (a) shows MR, MBR and RC of top-300 APIs with $\alpha = 1$. From Fig. 3 (a), we observe that RC of the API is almost average of the MR and the MBR generally. It is worth noting that API with high MBR does not always have high RC, and API with low MR does not always have low RC. It can be illustrated that our method can take both MR and MBR into full consideration to assess the relevance of the API to the malware class.

We can obtain top-k API feature subsets from results ranked by the RC value. Varying k, we have API feature subsets in different sizes. SC of top-100 APIs decrease rapidly, so we vary k from 10 to 100 by increasing 10 at a time to obtain the API feature subset. Random Forest [26] classifier are



(a) RC distribution of API (b) Acc (c) F1

Fig. 3: Evaluation of our selection method for API feature.

employed to evaluate the classification performance, because it shows outstanding classification performance among numerous shallow classifiers based on our best knowledge. Each app is first represented as a feature vector which eigenvalue is 1 if the app uses API otherwise is 0. Then the classification labels of malware are attached with 1 while benign apps are attached with -1. Once obtaining feature vectors and labels of training apps, classification model can be trained with the help of scikit-learn library[6]. After that, test samples can be predicted malware or benign via the trained classification model. In order to have a reliable performance measure and fair comparison, all 3,312 benign apps and 3,697 malware apps are mixed together and evaluated via tenfold cross validation.

In order to evaluate our selection method comprehensively, we set parameters α in Equation (4) as 0.5, 1.0 and 1.5 respectively to compare their classification performances. In addition, we also compare our method with the selection method in [3] which employed used usage difference in malware dataset and benign dataset. The Acc and F1 results are shown in Fig. 3 (b) and (c) respectively, the x-axis indicates the number of API feature and the y-axis indicates Acc or F1 indicator. The results of F1 are consistent with those of Acc, because malware and benign apps in our data set are almost balanced. Our selected API features have effective classification performance generally, especially only top-20 API features can achieve high F1 of 94.3% with $\alpha = 1.0$. Parameter α do affect classification performance especially when the number of API feature is small. When the number of API feature is less than 60, The classification performance under $\alpha = 0.5$ is better than that under $\alpha = 1.0$ or 1.5. However, classification performances under different parameters become comparative as the number of API feature increases. Moreover, our selection method outperforms the method of [3]. In conclusion, evaluation results demonstrate that our proposed selection method for API feature is effective for Android malware classification.

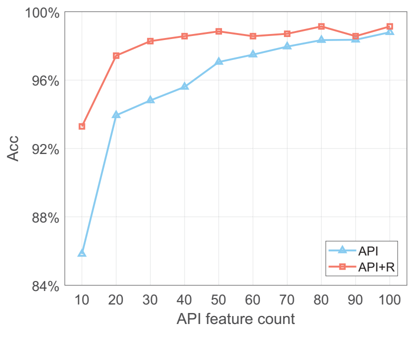
[6]https://scikit-learn.org

C. *Evaluation of our overall method proposed*

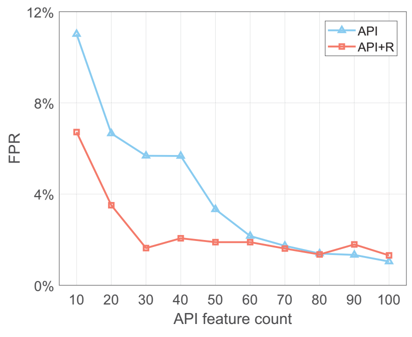
Evaluation of our overall method proposed has two parts, one is illustrating our further work based on API and the other is comparing our proposed method with two baseline methods.

1) Evaluation of our further work based on API feature: Besides selecting API feature related with the malware class, we further explore two structure relationships between them. Selected API feature and their structure information are mapped to an API-based feature graph which is classified by a CNN-based classifier. The expriment details of the CNN-based classifier are as follows. 32 or 8 convolution filters which size of 5×5 and step of 1 are used in the single convolution layer. Sampling size of max pooling layer is 2×2 and step is 2. And neurons turn to 1024 after the fully connected layer, the dropout regularization rate is set as 50%. Our CNN-based classifier is developed using TensorFlow framework [27]. The network weights are randomly initialized using the default TensorFlow initialization and are optimized with a learning rate of le-6 for 20,000 iterations, using a mini-batch size of 50. All the 3,312 benign apps and 3,697 malware apps are mixed together and evaluated via tenfold cross validation.

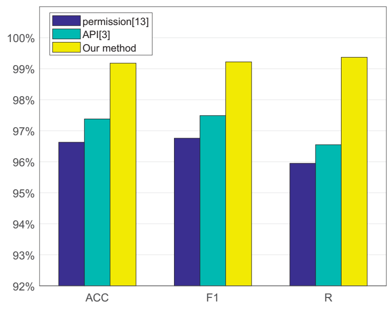
For API feature selected with $\alpha = 1.0$ in the first set of experiment, we further complement the structure relationship and employ the CNN-based classifier. The results are shown in Fig. 4 (a) and (b), the x-axis is the number of API feature and the y-axis is Acc or FPR results, the *API* legend indicates method in the first set of experiments (Section VI-B) and the *API+R* legend indicates our further work method based on API feature. We can observe that Acc and FPR indicators of the *API+R* method outperform those of the *API* method especially when the number of API feature is small. For example, the *API+R* method achieves 7.4% improvement in Acc indicator and 4.3% improvement in FPR indicator when only use top-10 API features. When the available API features are few, our further work can obtain effective improvement. Moreover, the improved FPR indicator demonstrates our motivation of structure relationship analysis is achieved. However, classification performance of the *API+R* method become comparative with



(a) Acc



(b) FPR



(c) Comparison of our method with two baseline methods

Fig. 4: Evaluation of our overall method proposed

that of the *API* method as the number of API feature increases. When the available API features are enough and sufficiently distinguishable, the *API* method is more lightweight.

2) Comparison with two baseline methods: We compare our method with two baseline methods proposed by [13] and [3]. [13] proposed a method for malware detection based on requested permission feature. [3] proposed a method for malware detection based on API feature which selected by usage difference in malware and benign dataset. They both used shallow classification model. The classification performances of our method and the two baseline methods are illustrated in Fig. 4 (c). We can observe that our method outperforms the two baseline methods. Since API feature is more fine-grained than permission in reflecting app behavior. Moreover, we improve the selection method for API feature comparing with [3], complement structure relationship and use deep CNN-based classifier.

D. Run-time overhead of feature extraction

For run-time overhead of feature extaction, we mainly analysis two cases as follows.

- API. We scan the whole smali files to extract API feature.
- API + R. We scan the whole smali files to extract not only API feature but also their structure relationships by the way.

The average run-time overhead of different Smali size in our collected dataset is shown in TABLE III. Two observations are obtained from TABLE III. First, run-time overhead is within 10 seconds when Smali size is less than 20M while run-time overhead increases sharply as Smali size becomes bigger. We draw conclusion that run-time overhead of feature extraction is sensitive to Smali size because scanning the whole smali files is exponentially complex. Second, extracting structure relationship of API feature results in extra run-time overhead. But most of time are still spent in scanning the whole Smali files, extracting structure relationship by the way only increases a few seconds. So structure relationship analysis can be adopted when using API feature. In conclusion, API feature is expensive in run-time overhead especially when the Smali size is big. However, once extract API features, extracting structure relationship between them does not result in too much extra run-time overhead.

TABLE III: Average run-time overhead of different Smali size

Feature	Smali size			
	0-10M	10-20M	20-50M	50-100M
API	1.1s	8.7s	21.5s	43.3s
API+R	1.8s	11.3s	25.6s	48.8s

VII. Conclusion and Discussion

In this section, we first sketch our work and then explain reasons why our method is effective in detecting Android malware. Limitations of our method are illustrated in the end.

In this paper, we propose a novel method for Android malware classification based on API feature. We select API feature related with malware and explore structure relationship between these APIs, then map them to a matrix as input API-based feature graph hand-refined for CNN-based classifier. For API feature selection, we harmonically consider occurrence frequency in malware and occurrence frequency difference between malware and benign using harmonic average function. Evaluations show that selected API feature can effectively distinguish malware from benign app, just top-20 APIs can achieve high F1 of 94.3%. However, APIs related with the malware class can also have a legitimate uses in benign thus causing FP (misclassify benign as malware) problem. Hence taking a step forward in this paper, we explore structure relationship between them and obtain further difference, that is APIs are in the same API block or nearby API blocks that have invoking relationship. API features and their structure relationships are mapped to a matrix as input feature graph for CNN-based classifier. When the available API features are few, classification performance including FPR indicator can achieve effective improvement effectively by complementing our further work based on the API feature.

The reason why our method can achieve effective performance in Android malware classification is common roles of API feature selection, structure relationship analysis and CNN-based classifier. First, we can obtain API feature related with malware more accurately through harmonically considering distribution of API using in malware and benign. Second, explore structure relationship between API feature can provide further difference between malware and benign to improve FP problem. Third, the convolution filter of CNN-based classifier perceives a subregion of the feature graph at a time and extracts a subset of the API-based features, which is more refine than the single API feature.

Our method is also subject to several limitations. It is difficult to obtain actual API using and their structure relationships once app is encrypted or obfuscated. In that case, dynamic analysis can be adopted in order to obtain invoking structure relationship between APIs. In addition, training effective CNN-based classifier is time-consuming and experience needed.

VIII. ACKNOWLEDGMENT

This work is funded by the Cyberspace Security Major Program in National Key Research and Development Plan of China under grant No. 2016YFB0800201, the Natural Science Foundation of China under grant No. 61572165, 61702150 and 61803135, the State Key Program of Zhejiang Province Natural Science Foundation of China under grant No. LZ15F020003, the Key Research and Development Plan Project of Zhejiang Province under grant No. 2017C01065 and the Public Research Project of Zhejiang Province under grant No. LGG19F020015.

REFERENCES

[1] C. A. Castillo, "Android malware past, present, and future," *White Paper of McAfee Mobile Security Working Group*, pp. 1–16, 2011.

[2] N. Etaher, G. R. S. Weir, and M. Alazab, "From zeus to zitmo: Trends in banking malware," in *IEEE Trustcom/bigdatase/ispa*, 2015.

[3] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International Conference on Security & Privacy in Communication Systems*, 2013.

[4] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Acm Sigkdd International Conference on Knowledge Discovery & Data Mining*, 2017.

[5] 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 *Ndss*, vol. 14, 2014, pp. 23–26.

[6] Z. Mu, D. Yue, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[7] K. W. Y. Au, F. Z. Yi, H. Zhen, and D. Lie, "Pscout: Analyzing the android permission specification," in *Acm Conference on Computer & Communications Security*, 2012.

[8] Z. Min, M. Sun, and J. C. S. Lui, "Droidanalytics: A signature based analytic system to collect, extract, analyze and associate android malware," in *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013.

[9] P. V. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[10] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *Computers & Security*, vol. 65, no. C, pp. 121–134, 2017.

[11] D. J. Wu, C. H. Mao, H. M. Lee, and K. P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security*, 2012.

[12] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *IEEE International Conference on Tools with Artificial Intelligence*, 2014.

[13] W. Wei, W. Xing, D. Feng, J. Liu, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics & Security*, vol. 9, no. 11, pp. 1869–1882, 2017.

[14] F. Ming, J. Liu, W. Wei, H. F. Li, Z. Tian, and T. Liu, "Dapasa:detecting android piggybacked apps through sensitive subgraph analysis," *IEEE Transactions on Information Forensics & Security*, vol. 12, no. 8, pp. 1772–1785, 2017.

[15] F. Ming, J. Liu, X. Luo, C. Kai, T. Chen, Z. Tian, X. Zhang, Q. Zheng, and T. Liu, "Frequent subgraph based familial classification of android malware," in *IEEE International Symposium on Software Reliability Engineering*, 2016.

[16] R. Nix and J. Zhang, "Classification of android apps and malware using deep neural networks," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 1871–1878.

[17] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 2. IEEE, 2016, pp. 577–582.

[18] F. Martinelli, F. Marulli, and F. Mercaldo, "Evaluating convolutional neural network for effective mobile malware detection," *Procedia Computer Science*, vol. 112, pp. 2372–2381, 2017.

[19] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," *Acm Sigcomm Computer Communication Review*, vol. 44, no. 4, pp. 371–372, 2014.

[20] N. Mclaughlin, J. M. D. Rincon, B. J. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, and G. J. Ahn, "Deep android malware detection," in *Acm on Conference on Data & Application Security & Privacy*, 2017.

[21] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, "Irmd: malware variant detection using opcode image recognition," in *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*. IEEE, 2016, pp. 1175–1180.

[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on International Conference on Machine Learning*, 2010.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] R. Y. Rubinstein and D. P. Kroese, *Applications of CE to Machine Learning*, 2004.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computer Science*, 2014.

[26] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 157–176, 2004.

[27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: A system for large-scale machine learning," 2016.