IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 6, 2018, accepted September 11, 2018, date of publication September 17, 2018, date of current version October 12, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2870534

Dalvik Opcode Graph Based Android Malware Variants Detection Using Global Topology Features

JIXIN ZHANG[1,2], ZHENG QIN[1], KEHUAN ZHANG[2], HUI YIN[1], AND JINGFU ZOU[1]
[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
[2]Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong

Corresponding authors: Zheng Qin (zqin@hnu.edu.cn) and Kehuan Zhang (khzhang@ie.cuhk.edu.hk)

This work was supported in part by the National Natural Science Foundation of China under Grant 61472131 and Grant 61772191, in part by the National Key Research and Development Program of China under Grant 2018YFB07040, and in part by the Science and Technology Key Projects of Hunan Province under Grant 2015TP1004 and Grant 2016JC2012.

ABSTRACT Since Android has become the dominator of smartphone operating system market with a share of 86.8%, the number of Android malicious applications are increasing rapidly as well. Such a large volume of diversified malware variants has forced researchers to investigate new methods by using machine learning since it provides a powerful ability for variants detection. Since the static analysis of malware plays an important role in system security and the opcode has been shown as an effective representation of malware, some of them use the Dalvik opcodes as features of malware and adopt machine learning to detect Android malware. However, current opcode-based methods are also facing some problems, such as considering both of accuracy and time cost, selection of features, and the lack of understanding or description of the characteristics of malware. To overcome the existing challenges, we propose a novel method to build a graph of Dalvik opcode and analyze its global topology properties, which will first construct a weighted probability graph of operations, and then we use information entropy to prune this graph while retaining information as more as possible, the next we extract several global topology features of the graph to represent malware, finally search the similarities with these features between programs. These global topology features formulate the high-level characteristics of malware. Our approach provides a light weight framework to detect Android malware variants based on graph theory and information theory. Theoretical analysis and real-life experimental results show the effectiveness, efficiency, and robustness of our approach, which achieves high detection accuracy and cost little training and detection time.

INDEX TERMS Dalvik opcode graph, global topology features, information theory, similarity searching.

I. INTRODUCTION

Being an open source operating system in smart phone, Android has increased the risk and serious issues related to malicious applications. Also the increase in the scale of applications in Android market makes it an easy target for malware authors [35]. According to [1], [12], and [18], recent statistical data show that over *95%* of mobile malware targeted Android, and there were about *500K* suspicious APK files or Android related files (e.g., dex, elf, jar files) submissions in VirusTotal each day. Traditional and commercial malware detection systems have predominantly utilised string signatures to query a database of pre-known malware instances. However, it is facing the problem that so many new malware coming every day.

In recent years, various approaches have to be devised to fight malware based on their nature. Some researches have proposed their Android malware detection methods by using machine learning with the features such as operation codes (opcodes), API functions, system calls, system status, etc. through static analysis and dynamic analysis.

Since static analysis of malware does not require conditional, untrusted or sandboxed execution of malware once the original contents of the malware are visible [10], and dynamic analysis such as [9] and [26] which monitor system calls or system status can hardly pre-detect the threat while the malware hides its malicious behaviours until there is a chance to attack, some of them prefer to use static analysis. Wu *et al.* [32] proposed a system, called DroidMat

which extracts the information such as permissions, API calls from the manifest file, then applies K-means algorithm that enhances the malware modeling capability, and finally adopt kNN algorithm to classify the application as benign or malware. Fan *et al.* [16] proposed to extract APIs from the .apk file to build a matrix according to the frequencies of their calling relationships, and then compare the matrix similarities between two .apk files. However, since the APIs are coarse-grained and sparse, it will be easier obfuscated by polymorphic techniques.

Since Dalvik opcodes [5] as one of the most used static features can finely represent behaviour patterns of programs, many researches propose to extract Dalvik opcode as their features and adopt diversified machine learning methods for their approaches. Some of them present traditional machine learning methods or statistical models such as [6], [8], [37], etc. They propose to firstly select some N-tuple opcodes as their features (Usually, N is less than *3*, since large number of feature dimensions severely limits the performance. The number of feature dimensions is $OpNum^N$ where *OpNum* is the total number of single opcodes.), and then choose one of applicable machine learning methods as their classifiers to classify malicious programs and legitimate ones. Some of them adopt deep learning based methods such as [24], [36], etc. Their approaches extract features and classify them by deep learning methods their own.

However, such opcode based machine learning researches are also facing some challenges. For N-tuple opcode based traditional machine learning approaches, they face a difficult choice of the scale of the selected features. As any single N-tuple opcode cannot certainly decide whether an instance is malicious or not, and less N-tuple opcodes cannot contain enough information to cover the characteristics of a wide rage of malware, they try to select more N-tuple opcodes to improve the accuracy. But the more N-tuple opcode is, the larger feature dimensions become, which brings more noise and more complexity, as well as more time cost of training and classification. For deep learning based approaches, although it automatically extracts features and organizes relationships between features, the key problem is that it is hard to understand the real meaning of the features and the inner relationships between the features. In addition, the training process of deep learning methods costs two much time, which makes it hard to implement in practice when facing the large scale of malware and so many new malware coming every day. For an instance, using a convolutional neural networks with one convolution layer (several convolution cores), one pooling layer (several pooling cores), a full connection layer and a softmax classifier to train thousands of pre-labeled samples in a server machine needs several days.

In this paper, we propose to address these challenges based on graph theory. There exist some call graph based approaches, such as [10], [16], etc. They use Control Flow Graph or API Call Graph to represent executables, and then search similarities between subgraphs of unlabeled samples and labeled samples. However these call graph based approaches are easier obfuscated by changing the calling sequence. There also exist some file-file graph based approaches, such as [11], [29], et.. These researches propose to build file-file relationships according to the interactions between files and hosts, and then adopt belief propagation methods to detect malicious files. However, due to the cold start problem, they cannot detect new malware variants in time.

Unlike the above mentioned approaches, we build a graph of opcodes instead of APIs or system calls to finely-gained represent the dependencies between opcodes. We use transition probability instead of control flow or data flow to maintain the statistical information which contains very important characteristics to distinguish malware and benign. Specially, we use topology features to formulate the characteristics of malware. Since graph can represent the natural relationships between entities, using a graph of Dalvik opcodes can represent dependencies between operations in malware. These dependencies show some characteristics of malware. Several global topology features are presented to represent the characteristics, which will be used to detect Android malware. The topology feature is a high-level, global representation of malware, which means it covers the whole information while reducing complexity of relationships between features. This improves the overall performance. Since the number of dimensions of topology features is much smaller than the low-level features such as N-tuple opcodes, the time cost of processing will be greatly reduced. In addition, it can be easier to understand and analyze its realistic meaning by human beings. We can gain some rules or knowledge by observing the difference of global topology properties between malware and benign. These will be very attractive given the fact there are so many new malware coming every day.

In order to achieve this goal, we have to address several challenging problems. One challenge is that although a graph of Dalvik opcode can represent the dependencies of operations in programs, however there is no evidence that shows a significant difference of operations and dependencies between malware and benign. The other challenge is that the existing global topology properties, such as graph density, the number of average neighbors, etc., cannot satisfy the need of representing the characteristics of malware. We addressed these challenges based on an important observation that the possibilities of relationships between operations can give us enough information for representing the activities of malware. In this paper, we proposed a weighted probability graph of Dalvik opcodes and several novel topology features to show the high-level representation of programs. The differential analysis in Evaluation Section shows the topology features between malware and benign can be significantly different. Since similar malicious code is often found throughout the malware landscape as attackers reuse existing code to infect different Android applications [17], the topology features between malware variants in the same family are quite similar, so a similarity search method is presented to detect malware by using these topology features.

A. CONTRIBUTIONS

The main contributions of this paper are summarized as follows.

1) We analyzed malicious operations in a very different way. Based on our observation, we found some high-level, global properties of malware, which formulate the characteristics of malware.
2) To effectively and efficiently detect Android malware, we proposed a weighted probability graph of Dalvik opcodes and several novel global topology features.
3) Implemented a prototype and evaluated with real world datasets. Evaluation results showed that our approach can stably achieve high classification accuracy of almost 94% and cost little detection time (less than *0.001s* per executable). Overall, when comparing with the state-of-art approaches, our method achieves a better performance in terms of several aspects by using only *6* topology features and a simple similarity searching method, while the Support Vector Machine (SVM) based method needs to uses *58564* 2-tuple Dalvik opcode features to achieve similar accuracy, but sacrifices much more speed.

B. PAPER ORGANIZATIONS

The remainings of this paper are organized as follows. Section 2 shows the overview of our approach. Section 3 presents our Dalvik opcode graph construction & pruning and Section 4 presents the global topology features extraction & similarity searching. Section 5 shows the evaluations and Section 6 introduces the related works. Section 7 and Section 8 show the discussion and conclusion.

II. OVERVIEW OF OUR APPROACH

In this section, we convert the Android malware/benign classification problem to a graph based topology features similarity searching problem. We propose a weighted probability graph of Dalvik opcodes and extract its global topology features for our Android malware detection approach. There are two phases in our approach: online phase and offline phase, as shown in Figure 1. In offline phase, we construct our opcode graph to prepare for pre-labeled feature database. In online phase, we extract topology features and search similarities between topology features in pre-labeled feature database. For both of the two phases, packed programs are first unpacked to remove packers. A packer is a program that encrypts or compresses other programs to packed programs so that the packer can protect the programs from disassembly. When executing a packed program, the packer decrypts or decompresses the packed program to the original unpacked program in memory and then executes the unpacked program. Then we decompress unpacked .apk executable files to .dex files and disassemble the .dex files to Dalvik opcodes profiles to obtain the opcode sequences. The module of graph construction & pruning generates a profile which contains a set of opcodes (vertices) and

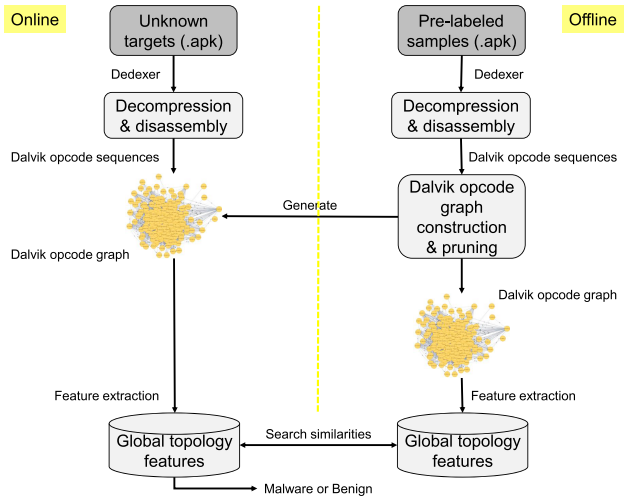


FIGURE 1. The architecture of our approach.

dependencies (edges) between opcodes as well as weights (values) which will be used to build the opcode graph.

Our approach includes the following four key steps: Dalvik opcode graph construction, Dalvik opcode graph pruning, global topology features extraction, and similarity searching.

Step 1 (Dalvik Opcode Graph Construction): Firstly, we obtain Dalvik opcode sequences from disassembling .dex files which are decompressed from .apk files (Android executables). We convert these Dalvik opcode sequences to a directed graph of vertices and edges. Each vertex represents an operation and each edge represents the sequential relationship between two operations. The weight values of the edges represent the co-occurrence probability of the two operations.

Step 2 (Dalvik Opcode Graph Pruning): Then, we prune the above mentioned Dalvik opcode graph based on information theory to reduce the complexity of the graph while maintaining the malware sensitive information as more as possible.

Step 3 (Global Topology Features Extraction): The next, based on graph theory, we extract *6* global topology features of the pruned Dalvik opcode graph. These features represent centrality of operations, activity of operation traces, density of graph, etc.

Step 4 (Similarity Searching): Finally, we search the similarities between the target and pre-labeled samples according to Manhattan Distance. The label of the most similar pre-labeled sample is also the label of the target.

III. DALVIK OPCODE GRAPH CONSTRUCTION & PRUNING

In this section, we first construct our Dalvik opcode graph, and then prune a part of nodes and edges in the graph while retaining as many as possible important connections between nodes.

A. DALVIK OPCODE GRAPH CONSTRUCTION

The Dalvik opcode graph is a directed graph of vertices which represent operations and directed edges which

represent relationships between operations. Let $G=(V, E)$ be the opcode graph, where V is a set of vertices and E denotes a set of directed edges. Each vertex v_i represents one of Dalvik opcode, such as ADD-DOUBLE, INVOKE-DIRECT-EMPTY, MONITOR-ENTER, etc. Each edge represents the dependencies between two sequential opcodes, for example, the edge $e_{i,j}=<v_i, v_j>$ represents a sequential operations from v_i to v_j.

Let $os_i=\{op_j, op_k, ..., op_t, op_m\}$ be an opcode sequence of a program where m is the length of the sequence. Let $OP_{total}=\{op_1, op_2, ..., op_n\}$ be the set of Dalvik opcodes where n is the total number of the opcodes, and $OP_i=\{op_j, op_k, op_t, op_m, ...\}$ be a subset of OP_{total} where the opcodes in the subset are also in os_i. Let $G_i=(V_i, E_i)$ be the Dalvik opcode graph of the program. The $G_i=(V_i, E_i)$ is a subgraph of the whole opcode graph and is automatically built by connecting the opcodes in the sequence os_i according to the sequential relationships between opcodes, where $V_i=OP_i$ and $E_i=\{e_{j,k}=<op_j, op_k>, ..., e_{t,m}=<op_t, op_m>\}$.

For example, the os_i={NEW-INSTANCE, SGET-OBJECT, INVOKE-DIRECT, INVOKE-VIRTUAL, INVOKE-VIRTUAL, INVOKE-VIRTUAL, MOVE-RESULT, ...} is an opcode sequence of a program, we obtain the OP_i={NEW-INSTANCE, SGET-OBJECT, INVOKE-DIRECT, INVOKE-VIRTUAL, MOVE-RESULT, ...} according to os_i. In the $G_i=(V_i, E_i)$ of the program, the $V_i=OP_i$ and the E_i={<NEW-INSTANCE, SGET-OBJECT>, <SGET-OBJECT, INVOKE-DIRECT>, <INVOKE-DIRECT, INVOKE-VIRTUAL>, <INVOKE-VIRTUAL, INVOKE-VIRTUAL>, <INVOKE-VIRTUAL, MOVE-RESULT>, ...}, as shown in Figure 2.

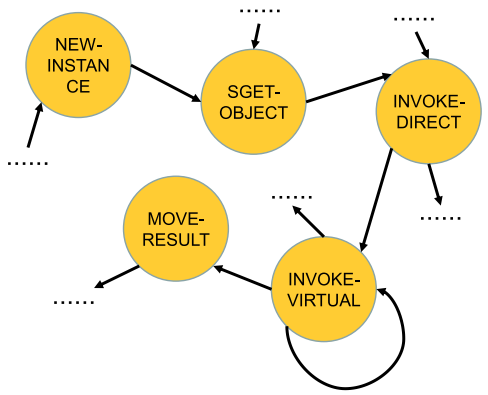


FIGURE 2. An example of Dalvik opcode graph within vertices and edges.

For each application, we build a Dalvik opcode graph, according to the sequential relationships of the opcodes in the opcode sequences. Each sequential relationship between two opcodes represents the operation dependency and the co-occurrence of the sequential operations. Since the opcode sequence is very long and the two sequential opcodes are various, the Dalvik opcode graph will be highly connected.

As we observed, the single operations and single dependencies between operations in malware are not unique, which means the single operations and the single dependencies in malware may also exist in benign. Fortunately, it has a significant difference between malware and bengin about the co-occurrence frequencies of these operations. So in this paper, we focus on the probabilities of the relationships. Such probability is the property value of the edges in the graph.

Let $val(e_{i,j})$ be the edge value of $e_{i,j}$ which represents probability of the sequential relationships from v_i to v_j, according to Eq. (1), where $N_{e_{i,j}}$ is the frequency of $e_{i,j}$ by counting the number of $e_{i,j}$ in the program.

$$val(e_{i,j}) = \frac{N_{e_{i,j}}}{\sum_{k=1,t=1}^{n,n} N_{e_{k,t}}} \tag{1}$$

B. DALVIK OPCODE GRAPH PRUNING

Since we only concentrate on malware activities, so we maintain the active dependencies in malware and prune the other dependencies to reduce the computation complexity. The pruning method is based on information entropy, according to Eq. (2), where $IG(e_{i,j})$ is the entropy, $\sum_{in-malware} val(e_{i,j})$ is the sum of edge value of $e_{i,j}$ in malware and $\sum_{in-benign} val(e_{i,j})$ is the sum of edge value of $e_{i,j}$ in benign. If $IG(e_{i,j})$>t, where t is a threshold, we maintain the edge $e_{i,j}$, otherwise we prune it. The $IG(e_{i,j})$ will be used as the weight of $e_{i,j}$ later.

$$IG(e_{i,j}) = ln(\frac{\sum_{in-malware} val(e_{i,j})}{\sum_{in-benign} val(e_{i,j})}) \tag{2}$$

The Figure 3 shows several examples of Dalvik opcode graph when the threshold t is *0*, *1*, *2*, *3*. To reduce the computation complexity while retaining as more as possible information, we choose threshold $t=0$.

IV. GLOBAL TOPOLOGY FEATURES EXTRACTION & SIMILARITY SEARCHING

Once we have constructed the opcode graph and pruned non-active connections, we extract topology features from this graph, and finally search similarities between programs by using these features.

A. GLOBAL TOPOLOGY FEATURES EXTRACTION

We propose *6* global topology features which represent global activity of programs in different angles. We extract such topology features based on graph theory which are illustrated as follows.

Node Number (NB) is the total number of vertices in the Dalvik opcode graph of a program. As a basic global topology feature, it represents the diversity of operations of the program.

Centrality (Ctr) identifies the number of the most important vertices within a graph, which represents the concentration degree of operations in the program. The important vertex is a vertex with high degree. We use the degree centrality according to Eq. (3), where $ID(v_i)$ is the in-degree of

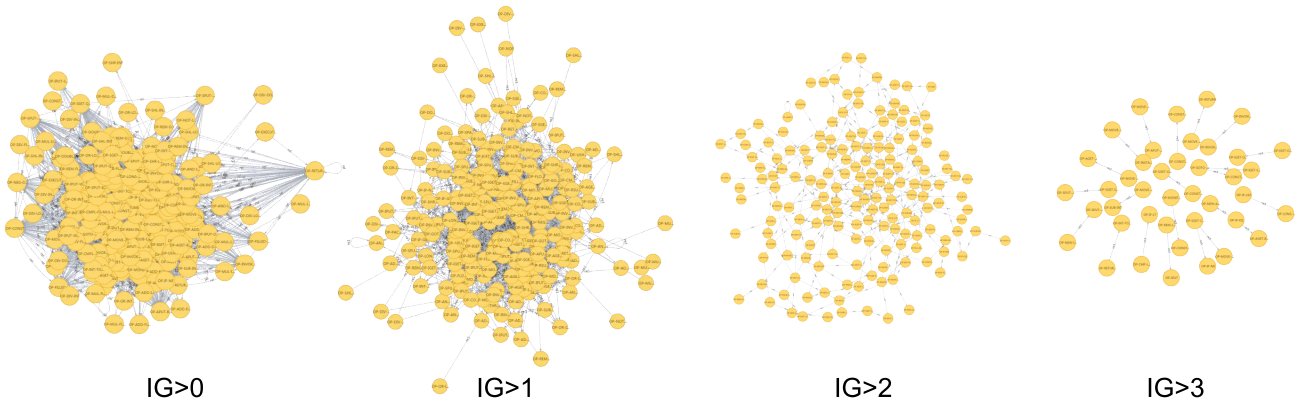


FIGURE 3. The Dalvik opcode graph.

vertex v_i and $OD(v_i)$ is the out-degree of vertex v_i. The intuition here is that the operations in most legitimate programs discretely connect with each other while the operations in some malicious programs concentratively connect with a few operations.

$$Ctr = \sum_{v_i \in V} 1, \{ID(v_i) > t | OD(v_i) > t\} \tag{3}$$

Edge Value (EV) is the sum of probability value of edges in the graph, as one of the basic global topology features, which represents co-occurrence frequencies of dependencies between operations, according to Eq. (4), where $P(e_{i,j})$ is the probability of edge $e_{i,j}$.

$$EV = \sum_{e_{i,j} \in E} P(e_{i,j}) \tag{4}$$

Graph Probability Density (GPD) is the probability density of the graph, which represents interaction intensity between operations. Similar to graph density, we introduce probability density to graph, according to Eq. (5), where NB is the total number of vertices in the graph. As an observation, the average node number of malicious programs is less than the average node number of legitimate ones, and the average edge value of malicious programs is higher than the average edge value of legitimate ones. So the graph probability density of malicious programs should be much larger than the density of legitimate programs.

$$GPD = \frac{2 \cdot \sum_{e_{i,j} \in E} P(e_{i,j})}{NB \cdot (NB - 1)} \tag{5}$$

Graph Distance (GWD) is the distance of weighted probability between the dependencies in the target program and the average dependencies in whole programs for training, according to Eq. (6), where the $Avg(e_{i,j})$ is the average value of $e_{i,j}$ in our training sets. With a intuition, since our Dalvik opcode graph is more sensitive to malicious operations and dependencies, which means that, each sequential relationships (edge) between operations (vertices) in legitimate programs are more tend to uniform distribution (*GWD* is close to *0*) while the relationships in malicious programs are more tend to non-uniform distribution and its distance towards a positive direction ($GWD > 0$).

$$GWD = \sum_{e_{i,j} \in E} IG(e_{i,j}) \cdot (P(e_{i,j}) - Avg(e_{i,j})) \tag{6}$$

Graph Activity (OTWA) is the sum of weighted probability of operation traces OT=ot_1, ot_2, ..., ot_x in the graph, which represents the probability of malicious operation trace in the graph, according to Eq. (7). The operation trace ot_i is the subgraph of the Dalvik opcode graph, which includes all reachable vertices from a initial vertex and the edges between these vertices. We extract the operation trace by using depth-first search [30]. The intuition is that the more frequent weighted operation trace in our graph is more trend to appear in malicious programs.

$$OTWA = \sum_{e_{i,j} \in OT} IG(e_{i,j}) \cdot P(e_{i,j}) \tag{7}$$

All of the above mentioned global topology features will be used to search similarities between non-labeled and pre-labeled instances in the next subsection.

B. SIMILARITY SEARCHING

To search the similarities between programs, we use Manhattan Distance to measure the distance between programs. The shorter distance is, the higher similarity is.

In this paper, we first use unity-based normalization to bring all values into the range [0,1] for each feature dimension, then use a simple similarity searching method and a simple distance equation to present the effectiveness and efficiency of our global topology features. Let $Dist(Features_i, Features_j)$ be the Manhattan distance between two programs i and j, according to Eq. (8), where $Features_i$ and $Features_j$ are the topology feature sets (vectors) of two programs, $N_{feature}$ is the number of features in the feature set and $f_{i,k}$ is the k^{th}

features in the feature set of program i.

$$Dist(Features_i, Features_j) = \sum_{k=1}^{N_{feature}} (f_{i,k} - f_{j,k}) \quad (8)$$

For each target, we detect it by searching its similarities with pre-labeled samples in training sets, and then find the most similar (minimum distance) pre-labeled sample. The label of the sample is also the label of the target.

V. EVALUATIONS

In this section, we present several experiments to show the performance of our approach. At the beginning, we present the experiment setup, the data set and the validation. And then, we discuss the performance of our approach and show that our approach can perform better by comparing with the state-of-art methods. Finally, we present a differential analysis of our topology features to explain the reasons why these topology features can be used to achieve such high performance.

A. SETUP, DATA SET AND VALIDATION

We implement our experiments on one computer. The version of its CPU is Intel i5-3470 @ 3.20GHz, the RAM is 16.0GB and the operation system is Linux Ubuntu 16.0. Our approach is developed by Java programming language.

Two data sets are considered in performance analysis of our approach for Android malware variants detection: the Android malware data set and the Android benign data set. To be close to real-life environment, the Android benign binary data set is downloaded from Google Play [4]. The Android malware instances are collected from Drebin [7] and MobileSandbox project [27]. Our final dataset contains *5550* Android benign apps and *5560* Android malware instances. As is shown in Table 1. For opcode based approaches, we use Dedexer [3] to disassemble Android instances, and obtain their dalvik opcode sequences.

TABLE 1. The Android malware data set.

Malware family	Number
FakeInstaller	925
DroidKungFu	667
Plankton	625
Opfake	613
GingerMaster	339
BaseBridge	330
Iconosys	152
Others	1909
total	5560

In order to evaluate the performance of our approach, we use k-fold cross validation in the experiments and choose the average values as our results. To avoid experimental biases, the pre-labeled malicious data sets and pre-labeled legitimate data sets have the same size. For similarity comparing, we randomly choose 2000 malware samples and 2000 benign samples as pre-labeled samples from our malware data sets and benign data sets and treat the other samples in our data sets as unknown targets.

```
sget-object     v3,The/The.The Ljava/util/Hashtable;
invoke-virtual  {v1,v0},java/lang/String/charAt ; charAt(I
move-result     v4
invoke-static   {v4},java/lang/Character/valueOf        ;
move-result-object      v4
invoke-virtual  {v2,v0},java/lang/String/charAt ; charAt(I
move-result     v5
invoke-static   {v5},java/lang/Character/valueOf        ;
move-result-object      v5
invoke-virtual  {v3,v4,v5},java/util/Hashtable/put      ;
sget-object     v3,The/The.is Ljava/util/Hashtable;
invoke-virtual  {v2,v0},java/lang/String/charAt ; charAt(I
move-result     v4
invoke-static   {v4},java/lang/Character/valueOf        ;
move-result-object      v4
invoke-virtual  {v1,v0},java/lang/String/charAt ; charAt(I
move-result     v5
invoke-static   {v5},java/lang/Character/valueOf        ;
move-result-object      v5
invoke-virtual  {v3,v4,v5},java/util/Hashtable/put      ;
add-int/lit8    v0,v0,1
goto    lfda
```

FIGURE 4. An example of the opcode sequences in .ddx file.

For each malware and benign samples as inputs, we first write a script to automatically decompress the .apk files to the .dex files, and then automatically disassemble the .dex file to the .ddx file by using dedexer mentioned before. The next we obtain the opcode sequences from the .ddx file, such as "sget-object, invoke-virtual, ...", as shown in Figure 4. The opcode sequences will be later sent to built the opcode graph according to the co-occurrence relationships between sequential opcodes. Since the .ddx files have a standard format, we can easily extract the opcode sequences and build the Dalvik opcode graph in the same way. The Dalvik opcode graph is applicable for all .ddx files.

B. STATE-OF-ART METHODS FOR PERFORMANCE COMPARISON

We compare our approach with several machine learning (deep learning) methods for dalvik opcode based Android malawre detection, such as Softmax, Back Propagation Neural Networks (BPNN), Support Vector Machine (SVM), Convolutional neural networks (CNN) with N-tuple opcodes (N = 2), similar to [6], [25], etc. For all of these methods (include our method) in the experiments, we do not abandon any features to retain more information and we use these learning methods for automatic training, weighting and classifying. We also re-implement several state-of-art approaches such as [8], [14], etc. Puerta *et al.* [14] proposed to adopt several machine learning methods to classify the Dalvik opcode vectors. Kang *et al.* [8] presented an n-opcode analysis based approach that utilizes machine learning to detect Android malware. The benchmarks we used for performance comparison include classification accuracy, detection precision, detection recall, detection false positive rate, detection time cost and disassembly time cost.

C. PERFORMANCE COMPARISON OF SEVERAL DALVIK OPCODE BASED APPROACHES

To demonstrate that our approach is effective and efficient, we implement our approach and the other state-of-art

TABLE 2. The average performance comparison with several machine learning (deep learning) methods for Android malware detection.

Method	Classification Accuracy	Detection Precision	Detection Recall	Detection 1-FPR	Training Time Cost	Detection Time Cost
Our approach (similarity)	93.6 %	92.3 %	94.0 %	92.1 %	–	< 0.001 s
SVM (our topology feature)	90.3 %	93.2 %	86.8 %	93.6 %	2.4 s	< 0.001 s
Softmax (2-tuple opcode)	73.4 %	77.0 %	66.7 %	80.0 %	1072.4 s	0.016 s
BPNN (2-tuple opcode)	–	–	–	–	–	–
CNN (2-tuple opcode)	90.1 %	88.9 %	91.7 %	88.5 %	178600.0 s	0.039 s

TABLE 3. The average performance comparison with the state-of-art approaches for Android malware detection.

Method	Classification Accuracy	Detection Precision	Detection Recall	Detection 1-FPR	Training Time Cost	Detection Time Cost
Our approach (similarity)	93.6 %	92.3 %	94.0 %	92.1 %	–	< 0.001 s
Puerta et al. (SMO)	87.4 %	86.9 %	88.0 %	86.7 %	47.9 s	< 0.001 s
Puerta et al. (Softmax)	81.7 %	82.2 %	80.8 %	82.4 %	557.0 s	< 0.001 s
BooJoong et al. (SVM, 2-opcode)	94.6 %	92.5 %	97.0 %	92.2 %	741.4 s	0.076 s
BooJoong et al. (NB, 2-opcode)	79.7 %	78.3 %	82.2 %	77.2 %	169.3 s	0.011 s

methods for performance comparisons. We also adopt SVM to classify malware and benign based on our global topology features to show that the topology features are also easier to classify. The classification accuracy is according to Eq. (9). *TPR* is the true positive rate (malware detection recall) according to Eq. (10), where *TP* is the number of malware cases which are correctly classified and *FN* is the number of malware cases which are misclassified as benign binaries. *TNR* is the true negative rate, as shown in Eq. (11), where *FP* is the number of benign cases which are incorrectly classified as malware binaries and *TN* is the number of benign binaries which are correctly classified. *FPR* is the false positive rate and *Precision* is the malware detection precision, as shown in Eq. (12) and Eq. (13).

$$accuracy = \frac{TPR + TNR}{2} \tag{9}$$

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

$$TNR = \frac{TN}{FP + TN} \tag{11}$$

$$FPR = \frac{FP}{FP + TN} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

As shown in Table 2, the experimental results show that: 1) By using a simple similarity searching method with the *6* global topology features, our approach performs well in terms of classification accuracy, detection precision, recall, FPR, training speed as well as detection speed by comparing with the other state-of-art methods. Our approach does not cost any training time but only search the maximum similarities. The results of SVM method based on our topology features show that the topology features can also be fitted to classification model. 2) The N-tuple opcode contains more information compared with single opcode, it needs stronger classifier, otherwise it is easier to lose classification ability. To address such problem, one solution is to select a few features to reduce the number of dimensions to match machine learning methods, however, in this way it lose much more original information [37]. To maintain the accuracy, the N-tuple opcodes based methods need much more features while sacrificing the training and classification speed. 3) The BPNN method fails at training and classification, because the number of dimensions of the 2-tuple opcode is too large, the initial weights are random values, so the sum of weighted features to different sigmoid units are very close that the sigmoid units are hard to activate. 4) The CNN method cost several days to train the 2-tuple opcode from thousands of samples in our server machine, which is hard to implement in practice while facing large scale of Android malware.

As shown in Table 3, the experimental results show that our approach can significantly improve the overall performance by comparing the state-of-art approaches for Android malware detection. In terms of accuracy, our approach and BooJoong et al.′ approach (SVM, 2-opcode) perform better. However, BooJoong et al.′ approach costs too much training and detection time cost due to its heavy feature dimensions. BooJoong et al.′s method (SVM, 2-opcode) performs better than Puerta et al.′s methods because their features contain more information, however, as a trade-off, their method costs much more time consumption. The accuracy of NB method (BooJoong et al.) is reduced because it assumes that the features are independent of each other, however the assumption is not really true.

D. ROBUSTNESS ANALYSIS OF OUR TOPOLOGY FEATURES BASED APPROACHES

Since training proportion $\frac{N_{training}}{N_{total}}$ is sensitive to the performance of supervised methods, it is also an important factor which should be considered (where $N_{training}$ is the number of training samples and N_{total} is the number of total samples, and $\frac{N_{total}-N_{training}}{N_{total}}$ is the detection proportion). To show that our approach is robust, we implement our approach to search similarities in different training proportions of the whole data sets. As show in Figure 5, the experimental results show that

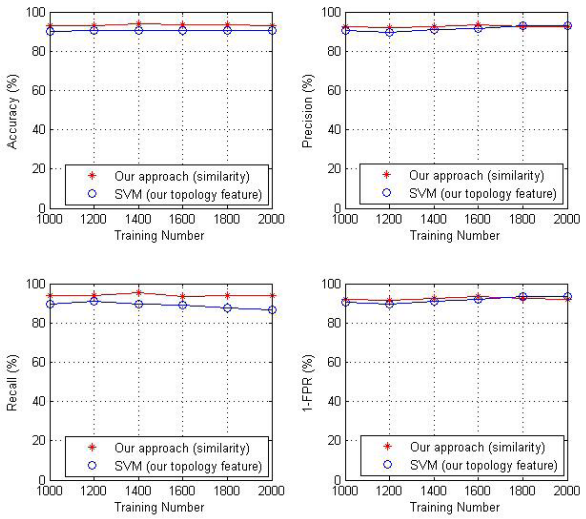


FIGURE 5. The robustness analysis of our topology features based approaches.

TABLE 4. The comparisons of global topology features between Android malware and benign.

Methods	Malware (avg)		Benign (avg)
Node Number	186.561	<	201.556
Centrality (>0.001)	**0.0730**	**>**	**0.0015**
Edge Value	0.0227	>	0.0207
Graph Probability Density	1.3524E-6	>	1.0963E-6
Graph Distance	**0.0020**	**>**	**0.0001**
Graph Activity	0.2282	>	0.1837

our approach can perform well in different training proportions. We also implement our global topology features in SVM classifier for Android malware detection. The results show that our topology features are robust both in similarity searching and SVM classification.

E. DIFFERENTIAL ANALYSIS OF GLOBAL TOPOLOGY FEATURES BETWEEN ANDROID MALWARE AND BENIGN

We analyze the difference of the global topology features in malware and benign to show that the features of malware and benign are very different. Such difference reflects the high-level characteristics of malware. As shown in Table 3, the results of comparisons show that: In our Dalvik opcode graph. 1) The average number of nodes and the average centrality of malware show that the operations in malware are more concerntrated. 2) The average value of edges, the average probability density of graph and the average weighted activity of graph show that the co-occurrence relationships between operations in malware are more frequent. 3) The results of the average weighted distance of graph show that the operations and the co-occurrence frequencies in benign are more average.

We present the distributions of the topology features between malware and benign, as shown in Figure 6. The results show a significant gap between malware and benign. The result of Ctr / NB shows that most of central nodes are in malicious programs. The other results also show the significant difference of distributions between malware and benign. From the results, the distributions of benigns are more centralized, because most of behaviours in benign are normal while malicious programs will have some special behaviours such as registry, trigger, etc.

VI. RELATED WORKS

Recent researches tend to use machine learning (includes deep learning) as well as graph computing methods to detect malware and its variants.

A. MACHINE LEARNING BASED MALWARE DETECTION METHODS

For machine learning based approaches, varied features are extracted from programs by using static analysis and dynamic analysis [35].

1) OPCODE BASED STATIC ANALYSIS

Some of them prefer to adopt opcode based static analysis. These opcodes will be embedded in a 1-D or 2-D vector and then sent to classifiers later to classify malicious programs and legitimate ones. Kang *et al.* [8] presented an n-opcode analysis based approach that utilizes machine learning to detect Android malware. With their n-opcode features, they found that Support Vector Machine (SVM) achieves best performance. Puerta *et al.* [14] proposed to adopt several machine learning methods to classify the Dalvik opcode vectors. McLaughlin *et al.* [24] used an opcode embedding matrix for representation and used a NGram based CNN model [20] to train and classify Android Dalvik opcode sequence. However, their embedding matrix is inefficient while the sequence is very long. Zhang *et al.* [36] proposed to convert opcodes into 2-D matrix, and adopted convolutional neural networks (CNN) to train the 2-D opcode matrix for malware recognition. Yan *et al.* [19] proposed to convert Android opcode into 2-D gray image with fixed size and adopt CNN to train and detect Android malware.

2) DYNAMIC ANALYSIS

Another prefer to use dynamic analysis. Massarelli *et al.* [23] proposed to extract features on resource consumption through fluctuation analysis and correlation and employ SVM method to classify malware into families. Yan *et al.* [34] proposed a method in which both the Java-level and OS level semantics are reconstructed seamlessly and simultaneously. By staying out of the execution environment and moreover, privilege based attacks can be detected. This approach also has a drawback of having very limited code coverage. Enck *et al.* [15] proposed to simultaneously track multiple sources of sensitive data and identify the data leakage. Data from privacy sensitive sources are automatically labeled (tainted) and labels are transitively applied as sensitive data moves through interprocess messages, program variables and files. Burguera *et al.* [9] collected traces from a large number of real users based on crowd-sourcing and then clustered these traces using k-means to detect malware. Shabtai *et al.* [26]

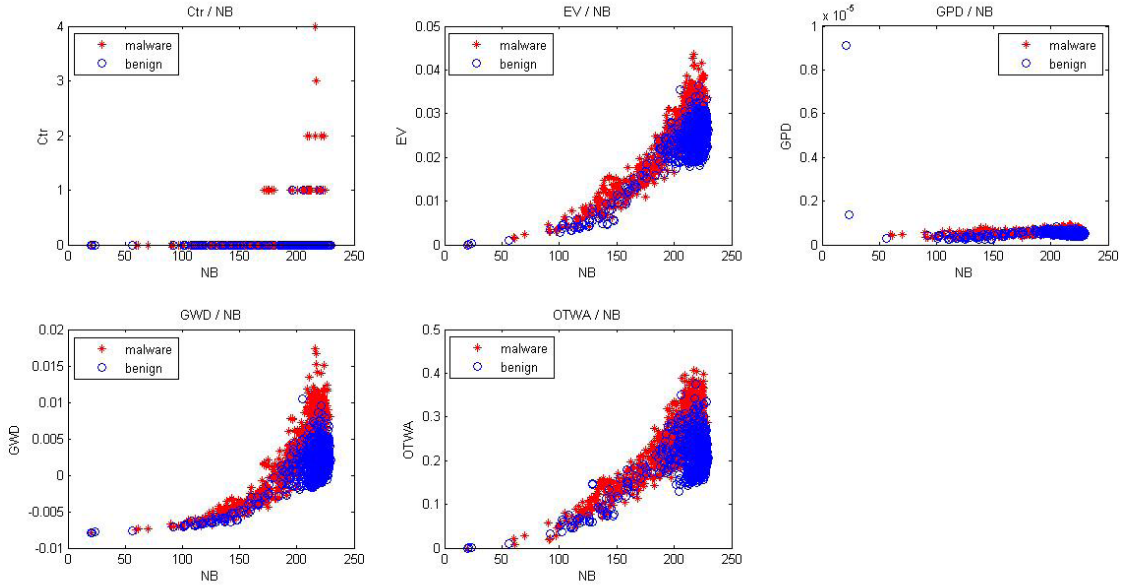


FIGURE 6. The distribution of Ctr / NB, EV / NB, GPD / NB, GWD / NB, OTWA / NB between malware and benign.

used machine learning algorithms to continuously monitor device state to differentiate between benign and malicious programs.

B. GRAPH BASED MALWARE DETECTION METHODS

Some researchers prefer to adopt graph based methods for malware detection, such as control flow graph, data flow graph, file-file graph, etc.

1) CONTROL DATA FLOW GRAPH

Their researches extract the control flow or data flow of API calls or system calls to build graph and then use graph matching or subgraph similarity searching methods to label the unknown samples according to known executables. Ming *et al.* [16] proposed to construct frequent subgraphs of API calls to represent the common behaviors of malware samples that belong to the same family. Tian *et al.* [31] proposed to utilize class-level dependence graph and method-level call graph to represent an app, and extract static behaviour features to detect Android malware. Cesare *et al.* [10] proposed a technique that performs similarity searching of sets of control flow graphs. Martín *et al.* [22] used third-party calls to bypass the effects of obfuscation, and then combined clustering and multi-objective optimisation to classify third-party call groups. Kolbitsch *et al.* [21] proposed to build a graph of data flows between the system calls, and then use graph matching for malware detection.

2) FILE VIA FILE GRAPH

These approaches extract the relationships between files and other entities, such as hosts, domains, etc. to build a graph, and then use belief propagation to label the unknown nodes according to the labeled nodes. Tamersoy *et al.* [29] proposed to generate file via file relationships according to the interactions between files and machines, and then adopt belief propagation methods to assign scores to every unlabeled file node. If the score of a file node is bigger than a threshold, then the file will be treated as a malicious file. Similar to [13]. Stringhiniq *et al.* [28] proposed a semi-supervised Bayesian label propagation to propagate the reputation of known files across a download graph that depicts file delivery networks (both legitimate and malicious).

VII. DISCUSSION

Static analysis based malware detection methods always rely on disassembly tools and other reverse engineering techniques. If a malicious sample is encrypted or compressed by packers, the disassembly tools cannot work. This problem limits the static analysis based malware detection methods in some situations.

However, in most cases, packers can be unpacked by unpacking techniques such as [33], which recovers original software sources. Since packed softwares have to unpack their inner original codes before executing the original codes, so that unpacking techniques always have a chance to get the original codes. For novel packing techniques, unpacking is often a dynamic process making effective static analysis against novel malware a hybrid approach. Additionally, snapshots of process images can be taken at runtime, thus avoiding the most common packing issues and can be used to statically identify if those processes belong to a known malware family [10].

In addition, dynamic analysis based methods may obtain software behaviours in runtime, but also face other limitations. The malicious samples can bypass detections by hiding their malicious behaviours until satisfying some conditions, such as non-virtual machine, burst point, etc. There does not exist a perfect technique which can break all of the limitations.

VIII. CONCLUSION

In this paper, we propose a novel Android malware detection approach which achieves high accuracy and speed. In our approach, we first propose a weighted probability graph of Dalvik opcodes and then prune a part of edges to reduce the computation complexity. After pruning, we extract a few topology features as representation of programs. By searching the similarities between the target program and pre-labeled samples with these features, we decide the target program is malicious if it is maximally similar to one of pre-labeled malware.

In the future, our technique can not only be used to detect malicious program, but can also be used to detect if a system is under threat by analyzing the interactions between system events, such as processes, files, sockets, etc.

REFERENCES

[1] (2014). *Mobile Malware*. [Online]. Available: http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easyway-you-stay-safe

[2] (2016). *IDC: Smartphone OS Market Share*. [Online]. Available: http://www.idc.com/promo/smartphone-market-share/os

[3] (2017). *Dedexer*. [Online]. Available: http://dedexer.sourceforge.net

[4] (2017). *Google Play*. [Online]. Available: http://vxheaven.org/vl.php

[5] (2018). *Dalvik Opcodes*. [Online]. Available: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

[6] A. Ali-Gombe, I. Ahmed, G. G. Richard, III, and V. Roussev, "OpSeq: Android malware fingerprinting," in *Proc. 5th ACM Program Protection Reverse Eng. Workshop (PPREW)*, 2015, p. 7.

[7] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. 21st Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2014, pp. 1–12.

[8] B. Kang, S. Y. Yerima, K. Mclaughlin, and S. Sezer, "N-opcode analysis for Android malware classification and categorization," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)* Jun. 2017, pp. 1–7.

[9] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proc. ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2011, pp. 15–26.

[10] S. Cesare, Y. Xiang, and W. Zhou, "Control flow-based malware variant-detection," *IEEE Trans. Depend. Sec. Comput.*, vol. 11, no. 4, pp. 307–317, Jul./Aug. 2014.

[11] D. H. P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, "Polonium: Tera-scale graph mining and inference for malware detection," in *Proc. SIAM Int. Conf. Data Mining*, 2011, pp. 131–142.

[12] K. Chen *et al.*, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-Play scale," in *Proc. Usenix Conf. Secur. Symp. (USENIX)*, 2015, pp. 659–674.

[13] L. Chen, W. Hardy, Y. Ye, and T. Li, "Analyzing file-to-file relation network in malware detection," in *Proc. 16th Int. Conf. Web Inf. Syst. Eng. (WISE)*, 2015, pp. 415–430.

[14] J. G. de la Puerta, B. Sanz, I. Santos, and P. G. Bringas, "Using Dalvik opcodes for malware detection on Android," *Logic J. IGPL*, vol. 25, no. 6, pp. 938–948, 2017.

[15] W. Enck *et al.*, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, p. 5, 2014.

[16] M. Fan *et al.*, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[17] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of Android malware using embedded call graphs," in *Proc. ACM Workshop Artif. Intell. Secur. (AISec)*, 2013, pp. 45–54.

[18] H. Huang *et al.*, "A large-scale study of Android malware development phenomenon on public malware submission and scanning platform," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2018.2790439.

[19] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 1, pp. 1–16, Feb. 2018, Art. no. 7247095.

[20] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. (Apr. 2014). "A convolutional neural network for modelling sentences." [Online]. Available: https://arxiv.org/abs/1404.2188

[21] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. 18th USENIX Secur. Symp.*, 2009, pp. 351–366.

[22] A. Martín, H. D. Menéndez, and D. Camacho, "MOCDroid: Multi-objective evolutionary classifier for Android malware detection," *Soft Comput.*, vol. 21, no. 24, pp. 7405–7415, 2017.

[23] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni. (Sep. 2017). "Android malware family classification based on resource consumption over time." [Online]. Available: https://arxiv.org/abs/1709.00875

[24] N. McLaughlin, J. M. del Rincon, B. Kang, and S. Yerima, "Deep Android malware detection," in *Proc. ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2017, pp. 301–308.

[25] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, no. 10, pp. 64–82, 2013.

[26] A. Shabtai, Y. Elovici, U. Kanonov, Y. Weiss, and C. Glezer, "'Andromaly': A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.

[27] M. Spreitzenbarth, E. Florian, S. Thomas, C. F. Felix, and J. Hoffmann, "Mobile-sandbox: Looking deeper into Android applications," in *Proc. 28th Int. ACM Symp. Appl. Comput. (SAC)*, 2013, pp. 1808–1815.

[28] G. Stringhini, Y. Shen, Y. Han, and X. Zhang, "Marmite: Spreading malicious file reputation through download graphs," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2017, pp. 91–102.

[29] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association: Large scale malware detection by mining file-relation graphs," in *Proc. ACM Int. Conf. Knowl. Discovery Data Mining (SIGKDD)*, 2014, pp. 1524–1533.

[30] R. Tarjan, "Depth-first search and linear graph algorithms," in *Proc. 12th Annu. Symp. Switching Automata Theory (SWAT)*, Oct. 1971, pp. 114–121.

[31] K. Tian, D. D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged Android malware with code-heterogeneity features," *IEEE Trans. Depend. Sec. Comput.*, to be published, doi: 10.1109/TDSC.2017.2745575.

[32] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in *Proc. 7th Asia Joint Conf. Inf. Secur.*, Aug. 2012, pp. 62–69.

[33] L. Xue, X. Luo, L. Yu, S. Wang, and D. Wu, "Adaptive unpacking of Android apps," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 358–369.

[34] L.-K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *Proc. USENIX Secur. Symp.*, 2012, pp. 569–584.

[35] R. Zachariah, K. Akash, M. S. Yousef, and A. M. Chacko, "Android malware detection a survey," in *Proc. IEEE Int. Conf. Circuits Syst. (ICCS)*, Dec. 2017, pp. 238–244.

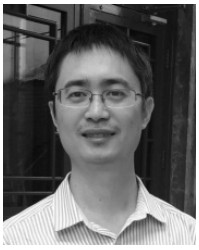
[36] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, "IRMD: Malware variant detection using opcode image recognition," in *Proc. 23rd IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 1175–1180.

[37] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao, and Y. Hu, "Malware variant detection using opcode image recognition with small training sets," in *Proc. 25th IEEE Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–9.

JIXIN ZHANG received the B.S. degree in mathematics and the M.S. degree in computer science and technology from the Wuhan University of Technology in 2007 and 2011, respectively. He is currently pursuing the Ph.D. degree with the College of Information Science and Engineering, Hunan University, and doing research with the Department of Information Engineering, The Chinese University of Hong Kong. His primary researches focus on system security, machine learning, and knowledge graph.

ZHENG QIN received the B.S. degree from the Wuhan University of Technology in 1991 and the Ph.D. degree in computer software and theory from Chongqing University, China, in 2001. Then, he was with industry from 2001 to 2005. He is currently a Professor of computer science and technology with Hunan University, China. His main interests are information security, computer network, and big data. He is a member of the China Computer Federation and ACM.

KEHUAN ZHANG received the B.S. and M.E. degrees from Hunan University in 2001 and 2004, respectively, and the Ph.D. degree from The Chinese University of Hong Kong (CUHK) in 2012. He was with industry from 2004 to 2007 before starting his Ph.D. Program at Indiana University, Bloomington. During his Ph.D. Program, he was an Intern at the IBM T. J. Watson Research Center in 2010. He is currently a Professor with the Department of Information Engineering, CUHK. His primary research interests are: security and privacy in computer network, Web, clouds, smart phones, embedded systems and other distributed computing systems. He has published several high quality papers on all of the four top conferences in network and system security area, including IEEE Oakland, ACM CCS, USENIX Security, and NDSS. He is also an active external reviewer of these top conferences.



HUI YIN received the B.S. degree in computer science from Hunan Normal University, China, in 2002, the M.S. degree in computer software and theory from Central South University, China, in 2008, and the Ph.D. degree from the College of Information Science and Engineering, Hunan University, China, in 2018. He is currently an Assistant Professor with the College of Applied Mathematics and Computer Engineering, Changsha University, China. His interests are information security, privacy protection, and applied cryptography.

JINGFU ZOU received the B.S. degree from the Xi'an University of Finance and Economics in 2015 and the M.E. degree in computer technology from Hunan University. His main research interests are knowledge graph and personalized recommendation.