An Empirical Study on Android Malware Characterization by Social Network Analysis

Haojun Zhao, Yueming Wu, Deqing Zou, Hai Jin

***Abstract*—Android malware detection has always been a hot research field. Prior work has validated that graph-based Android malware detection methods are effective, and several works have been proposed to regard the call graph of an app as a social network for more efficient classification. However, a social network contains many properties and there is a lack of perception that which social network properties are more useful in differentiating malware from benign apps. Therefor, in this paper, we present the first empirical study to analyze Android malware by different social network properties. We conduct extensive statistical analysis on 100,000 Android apps and apply three feature ranking methods to research the ability of 57 social network properties on malware detection. Moreover, in an effort to validate the effectiveness of these social network properties on malware detection, we implement a tool called *SNADroid* by using these properties as features for models training and use it to complete classification. Our study reveals that the *Average Triangles Number* is the most impactful social network property in distinguishing malware from benign apps. Combined with the experimental results and in-depth analysis, we present the 15 most effective features for graph-based malware detection using social properties as a guideline.**

***Index Terms*—Android Malware, Social Network Analysis, Empirical Study.**

I. Introduction

SOCIAL networks have fundamentally changed how people produce and consume online information, further reducing access barriers and enabling new forms of interaction between people, things, information, and services. Social network analysis is a way to understand how the network interacts. As a result of the high effectiveness and scalability of social network analysis, it has been used in many study fields, such as biological networks [1], transportation networks [2], and affiliation networks [3]. Social network analysis is becoming more and more popular, and several works [4]–[7] have been proposed to use it for Android malware analysis.

Existing Android malware detection methods extract different types of features from apps and utilize them to train classifiers for conducting classification. For instance, [8], [9] pay attention to the permissions requested by applications and built models to detect malware. However, it is possible for benign applications to request more permissions than necessary, which may also lead to a high false positive rate [10]. In order to obtain more comprehensive features from an app, *Drebin* [11] uses a time-intensive static analysis to extract features, which includes not only permissions but also Application Programming Interface (API) calls. So many features can indeed improve the accuracy of detection results, however, it is not enough robust because it lacks attention to structure and contextual information of the program behaviors. To obtain more robust features, many works have been proposed to extract program semantics using graph-based methods. Techniques such as [12]–[17] have validated these graph-based methods are highly effective for Android malware detection. Nevertheless, they suffer from low scalability when conducting graph matching to detect malware because a graph of an app often owns thousands of function nodes. Most of the works have expensive running overhead (*i.e.,* the analysis time on an app of *DroidSIFT* and *Apposcopy* is 175.8 seconds [13] and 275 seconds [14], respectively), which demonstrate their low efficiency in malware detection.

To complete more efficient graph-based malware detection, *MassVet* [18] characterizes apps by building view graphs with complicated UI structures. In order to ensure the graph matching is highly scalable, *MassVet* adopts a similarity comparison algorithm that appeared in their former work [19] for the analysis of the recovered view graphs. However, *MassVet* is originally designed to detect repackaged malware, so it may lead to a false positive when processing a new malicious app. Frenklach *et al.* [20] propose a static Android application analysis method based on Application Similarity Graph (ASG) combined with neural networks, but it is easily bypassed by malware masquerading as benign software. In an effort to detect more general malware, another state-of-the-art method (*MalScan* [4]) has been proposed to detect malware effectively and efficiently. Instead of traditional heavyweight program analysis, it describes the function call graph of an application as a complex social network and adopts social-network-centrality analysis on sensitive API calls to extract program semantic features. Then they perform extensive evaluations to verify the high effectiveness of *MalScan*. However, to achieve high scalability on market-wide mobile malware detection, *MalScan* only considers one social network property (*i.e.,* node centrality). In reality, a social network includes many properties, such as density, radius, diameter, and clustering coefficient. Although social network analysis has been demon-

Haojun Zhao, Deqing Zou are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074, China (e-mail: haojunzhao@hust.edu.cn, deqingzou@hust.edu.cn).

Yueming Wu (corresponding author) is with the School of Computer Science and Engineering, Nanyang Technological University, 639798, Singapore (e-mail: wuyueming21@gmail.com).

Hai Jin is with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China (e-mail: hjin@hust.edu.cn).

strated to be effective in malware detection, we have no idea which social network properties are more useful in detecting Android malware.

In this paper, we present the first empirical study to explore the ability of different social network properties on Android malware detection. In practice, although there have been some graph-based works demonstrating the effectiveness of using social networks for malware detection, they all use only simple and individual social network node property (*i.e.,* node centrality). In other words, there is a lack of research to study the ability of different social network properties on graph-based malware detection, especially those social network properties at the whole graph scale. Therefore, we conduct empirical research through various social network properties to mine which properties are more different between malicious and benign apps call graphs as well as more effective for malware detection. As social network properties are an effective way to measure the logical properties of a graph, our empirical conclusions can be extended to not only social network-based but also all graph-based malware detection works and provide them solid theoretical support.

Specifically, we first construct an Android app dataset by crawling APK files from AndroZoo [21], our final dataset includes 75,000 benign apps and 25,000 malicious apps. Given an APK file, we then extract the function call graph by lightweight static analysis. After obtaining the call graph, we treat it as a social network and utilize social network analysis to dig out 57 social network properties (Table I). Then three feature ranking methods are employed to rank these properties for finding the most informative properties on malware detection. From the research results, we observe that *Average Triangles Number*, *Average Katz Centrality*, *Degree Assortativity Coefficient*, and *Maximal Harmonic Centrality* are more capable of differentiating malware from benign apps, while *Minimal Harmonic Centrality*, *Minimal Closeness Centrality*, *Minimal Betweenness Centrality*, and *Minimal Shortest Path Length* have no ability to differentiate between malware and benign applications. We interpret in detail why these social network properties are highly distinguishable for machine learning model and their distribution characteristics between benign and malware. In an effort to validate the effectiveness of these social network properties on Android malware detection, we leverage these properties to construct feature vectors and feed them into machine learning models. We develop a prototype system, *SNADroid*, and evaluate it with our dataset. The experimental results show that *SNADroid* is capable of detecting Android malware.

In summary, this paper makes the following contributions:

- We present the first empirical study on the impact of different social network properties on Android malware detection. We systematically evaluate the malware detection capabilities of 57 commonly used social network properties through extensive comparative experiments and three feature ranking methods (i.e., T-test, normalized mutual information, and maximum information coefficient).
- We implement an automatic Android malware detection system called *SNADroid* using social network properties and prove the effectiveness of different social network properties on Android malware detection.
- Through in-depth analysis of the experimental results, we distill a total of 15 most effective social network properties in three categories as a guideline for graph-based malware detection work.

II. Methodology

A social network is a social structure that consists of a group of social actors, such as individuals or organizations, along with the relationships and interactions between them[1]. Social networks are valuable for measuring graph properties and uncovering the underlying topological relationships within graphs. Similarly, when analyzing a program, we can view the functions within the program as user nodes in a social network, and the calling relationships between these functions as social relationships between the user nodes. By adopting this perspective, we can leverage the concept of social network to describe and analyze the function call graph of the program. This approach allows us to apply social network analysis to gain insights into the structure and behavior of the program. For example, we can examine properties such as centrality, clustering, and connectivity within the function call graph, enabling us to understand the patterns of function interactions and identify important functions within the program.

A. Method Overview

Figure 1 presents the procedures of our empirical study which includes four main phases: *Static Analysis*, *Feature Extraction*, *Feature Ranking*, and *Classification*.

- ***Static Analysis:*** This stage aims to extract the function call graphs of apps through static analysis, where each node is a function, either an API call or a user-defined function.
- ***Feature Extraction:*** After obtaining the call graph of applications, we regard the call graph as a complex social network and extract 57 different social network properties from the graph.
- ***Feature Ranking:*** In this phase, we utilize three feature ranking methods (*i.e.,* T-test, normalized mutual information, and maximal information coefficient) to research the capability of 57 social network properties on distinguishing benign apps from malware.
- ***Classification:*** Given the rankings of 57 social network properties, we select the corresponding social network properties according to their rankings to construct the feature vectors. These feature vectors are used to train classifiers for Android malware detection.

In particular, we implement a custom Android malware detection system called *SNADroid* for subsequent experimental work.

B. Static Analysis and Feature Extraction

To extract the function call graph from an app's APK file, we utilize static analysis techniques implemented within

[1]https://en.wikipedia.org/wiki/Social_network

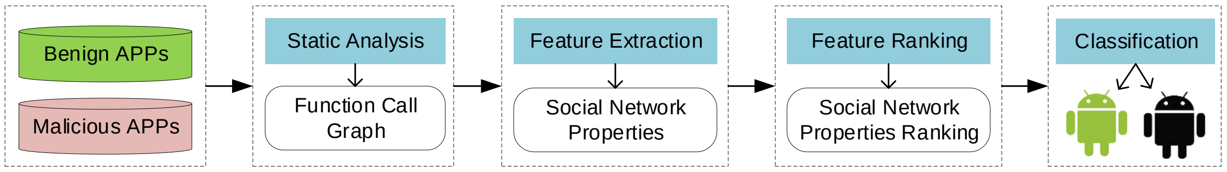

Fig. 1. Procedures of our empirical study

Androguard [22]. Androguard provides the necessary functionality to analyze the structure and components of an APK file, including building the function call graphs. Once we have obtained the function call graphs, we consider them as complex social networks and apply social network analysis techniques to uncover various properties in these graphs. By treating the call graphs as social networks, we can leverage existing methodologies from social network analysis to gain insights into the program's behavior. It's important to note that both call graphs and social networks are representations of dynamic behaviors using static graphs. In the case of function call graphs, they capture the program's behavior by representing the relationships between functions, where each edge of the call graph may signify multiple function invocations. We expect to discover more potential logical topological features in the program call graph through social network analysis. In order to conduct extensive and profound research on social properties, we end up extracting a total of 57 social network properties from a function call graph which cover almost all common social network properties. We divide all social network properties into three categories: whole graph topology properties, node properties, node cluster properties.

Whole graph topology properties are used to characterize the whole network from different aspects and some of the details are shown in TABLE I. For instance, the size of the network can be characterized by nodes, edges, diameter, and radius; the cohesiveness of the network can be described by density, average clustering coefficient, and component number; and the structure of the network can be represented by clique number, cycles number, and triads census. Moreover, centrality measures [23]–[27] are used to quantify the importance of a node in the network. However, in this paper, we focus on studying the properties of a whole network instead of individual nodes within the network. Therefore, we only select the average, maximal, and minimal centrality values among all the nodes within the network as node properties for each centrality measure. As for the node cluster properties, they reflect the topological characteristics of a social network which are suitable to be measured by triads census. According to the different edge relations between any three user nodes in a network, there are 16 different triads types in total [28]. We focus on the number of 16 triads types possibly presenting in a network. In other words, triads census includes a total of 16 social network properties, corresponding to the number of 16 different types of triads in one network (*i.e.,* 003, 012, 102, 201, 210, 300, 021C, 021D, 021U, 030C, 030T, 111D, 111U, 120C, 120D, and 120U in Figure 2). In function call graph, the triads census reflects the close relationship between function nodes, and reveals whether the program function calls

TABLE I
DESCRIPTIONS OF SOCIAL NETWORK PROPERTIES

Social Network Property	Description
Nodes	The number of nodes in a network.
Edges	The number of edges in a network.
Density	The density of the network.
Diameter	The maximum eccentricity of the network.
Radius	The minimum eccentricity of the network.
Eccentricity Number	The maximum distance from the node to other nodes in the network.
Periphery	The set of nodes with eccentricity equal to the diameter.
Degree Assortativity Coefficient	The measurement of the similarity of connections in the network with respect to the node degree.
Bridge Number	The number of the edges that don't belong to any cycle.
Degree Centrality	The centrality that for each node is the fraction of nodes it is connected to.
Katz Centrality	The centrality for the nodes based on the centrality of their neighbors.
Harmonic Centrality	The centrality that for each node u is the sum of the reciprocal of the shortest path distances from all other nodes to u.
Closeness Centrality	The centrality that for each node u is the reciprocal of the average shortest path distance to u over all n-1 reachable nodes.
Betweenness Centrality	The centrality that for each node is the sum of the fraction of all-pairs shortest paths that pass through the node.
Clique Number	The size of the largest clique in the graph.
Maximal Clique Number	The maximal clique number in the network.
Largest Clique Size	For each node is the largest size of the complete subgraph containing it.
Average Triangles Number	The average of the number of triangles that include a node as one vertex.
Transitivity	The fraction of all possible triangles present in the graph.
Average Clustering Coefficient	The local clustering of each node in G is the fraction of triangles that actually exist over all possible triangles in its neighborhood. The average clustering coefficient of a graph G is the mean of local clusterings.
Strongly Connected Components Number	The number of strongly connected components in the network.
Weakly Connected Components Number	The number of weakly connected components in the network.
Attracting Components Number	The number of attracting components in the network.
Algebraic Connectivity	The second smallest eigenvalue of its Laplacian matrix.
Cycles Number	The number of the minimal collection of cycles such that any cycle in the network can be written as a sum of cycles in the basis.
Simple Cycles Number	The number of the close path where no node appears twice.
Reciprocity	The ratio of the number of edges pointing in both directions to the total number of edges in the network.
Average Shortest Path Length	The average of the shortest path length.
Maximal Shortest Path Length	The maximum of the shortest path length.
Minimal Shortest Path Length	The minimum of the shortest path length.
Shortest Path Number	The number of shortest path.
Triads Census	The triads census is a count of how many of the 16 possible types of triads are present in a directed graph.

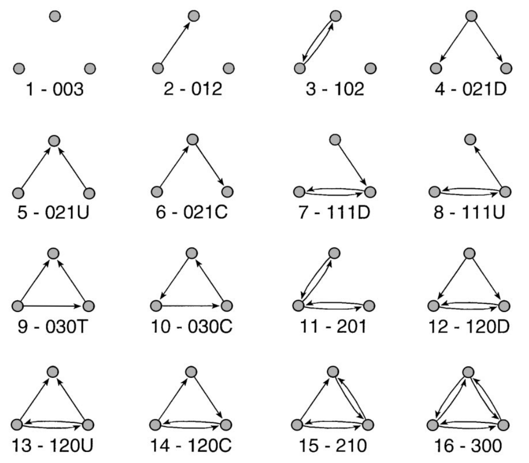

Fig. 2. Types of triads

are close, frequent and coherent.

Note that some social network properties (*e.g.,* clustering coefficient) are defined only for undirected graphs. Therefore, we first convert the function call graph as an undirected graph when extracting these properties.

C. Feature Ranking

The measurement of the correlation between a feature and class variables is called feature ranking in machine learning, whose purpose is to select the most informative features. Compared with machine learning models and algorithms, feature selection determines the upper limit of machine learning performance. Therefore, in this stage, we aim at ranking all social network property features and digging out which features are useful in characterizing mobile malicious behaviors. As a matter of fact, there have proposed certain feature importance measurement methods such as *T-test* and *Pearson Correlation*. Besides, some machine learning algorithms such as *Random Forest* and *Logistic Regression* can also score the training features by themselves. To make our ranking results more general and interpretable, we utilize the three most widely used feature ranking methods, namely *T-test*, *Normalized Mutual Information* (NMI), and *Maximal Information Coefficient* (MIC) to commence our feature ranking phase. We present the introduction of these three feature ranking methods as follows:

1) T-test: T-test utilizes the t-distribution theory to infer the probability of a difference occurring, then comparing whether the divergence between the two means is significant. That is, a T-test examines the t-statistic, the t-distribution values, and the degrees of freedom to determine the probability of discrepancy between two sets of data. Let S_1^2, S_2^2 be the sample variance and n_1, n_2 be the sample capacity, then the T-test is performed by

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{(n_1-1)S_1^2+(n_2-1)S_2^2}{n_1+n_2-2}(\frac{1}{n_1}+\frac{1}{n_2})}}$$

P-value is the probability or significance when the null hypothesis H_0 is true after statistical testing. The null hypothesis H_0 is rejected if the calculated p-value is less than a pre-determined threshold value α (*e.g.,* 0.5, 0.05), which is referred to as the level of significance.

2) Normalized Mutual Information: Normalized Mutual Information aims to put the value of mutual information into [0, 1] to evaluate the quality of the algorithm. The formula plays as

$$NMI(X;Y) = 2\frac{I(X;Y)}{H(X)+H(Y)}$$

where I(X;Y) is the mutual information of the two variables and H(X), H(Y) are the information entropy respectively. Besides, let the joint distribution of two stochastic variables X, Y be P(X,Y) and the edge distributions be P(X), P(Y), then these statics are performed as

$$I(X;Y) = \sum_{x\in X}\sum_{y\in Y} P(X,Y)log\frac{P(X,Y)}{P(X)P(Y)}$$

$$H(X) = -\sum_{i\in X} P(X_i)logP(X_i)$$

In this paper, the NMI value of each social network property is between 0 and 1. NMI(X;Y)=0 indicates that there is no correlation between X and Y, while NMI(X;Y)=1 means that Y is completely inferable by knowing X. The bigger the value of NMI, the stronger the correlation is between X and Y.

3) Maximal Information Coefficient: Maximal Information Coefficient(MIC) applies mutual information to continuous random variables and measures the degree of association between two factors. Compared to Mutual Information, MIC is more accurate because of its universality, fairness, and symmetry. It discretizes the relationship between two variables in a two-dimensional space and uses a scatter plot to represent. Then it divide the space into a certain number of intervals in the x, y direction and check the position of the scatters in each square. MIC is performed as

$$MIC(X;Y) = max_{a*b<B}\frac{I(X;Y)}{log_2 min(a,b)}$$

where a,b are the numbers of divided grids in the x, y direction, which is essentially the grid distribution and B is the variable which is about 0.6 power of the amount of data. More generally, MIC has the following three important properties: 1) If the variable x,y presence function, then when the sample increases, MIC values towards 1; 2) If the variable x,y can have a parametric equation $c(t) = [x(t), y(t)]$ when the sample is increased, MIC inevitably tends to 1; 3)If the variable x,y independently of one another, then when the sample is increased,MIC values inevitablytends to 0. In this paper, MIC(X;Y) is nonnegative in [0,1]. MIC(X;Y)=0 indicates no correlation, while MIC(X;Y)=1 means Y is inferable from X.

For each social network property X, the correlation between X and class Y can be computed by the above-mentioned methods. These calculated values are used to evaluate the rankings of all social network properties.

D. Classification

From the above subsection, we can obtain the rankings of these 57 social network properties on differentiating Android malware from benign apps. Given the rankings of different social network properties, we then build classifiers by selecting different properties based on their rankings to perform malware detection. We totally choose four different learning algorithms: 1-Nearest Neighbor (1-NN), 3-Nearest Neighbor (3-NN), Random Forest, and Decision Tree for classification. These four classifiers are implemented by the scikit-learn python library [29]. For the Random Forest, we adopt the default parameters to commence our experiments [2]. Each model is trained by using feature vectors obtained from a training dataset and then performing classification on a testing dataset. All the experimental results are presented in Section III by performing 10-fold cross validations on our dataset. 10-fold cross-validation is a technique where the dataset is divided into 10 equally-sized folds. The models are then trained and tested 10 times, each time using a different fold as the testing set and the remaining nine folds as the training set. This approach ensures that every sample in the dataset is used for both training and testing, reducing the potential bias introduced by a single train-test split.

III. Experiments

In this section, we aim at answering the following research questions:

- *RQ1: How different are social network properties between function call graphs of benign apps and malicious apps?*
- *RQ2: Which social network properties are more useful in differentiating Android malware from benign apps?*
- *RQ3: What is the effectiveness of different social network properties in detecting Android malware?*

TABLE II
SUMMARY OF THE DATASET USED IN OUR EMPIRICAL STUDY

Category	#Apps	Average Size (MB)
Benign apps	75,000	3.45
Malicious apps	25,000	3.26
Total	100,000	3.40

A. Dataset and Metrics

In this paper, we conduct the first empirical study on Android malware characterization by social network analysis. AndroZoo [21] is a growing collection of Android Applications collected from several sources, including the official Google Play app market. Therefore, it can comprehensively represent the features of existing Android apps. We randomly download 100,000 APK files from AndroZoo as the dataset for our empirical study, and all of them have been analyzed by different AntiVirus products in VirusTotal [30]. We leverage the detection reports to filter and generate our dataset. An APK we consider as benign only if it's all reports show normal. For collecting malware samples, we download APK files that are reported as malicious by several AntiVirus. Because a program is very likely to be a false positive if only one AntiVirus product flags the program as malware. In order to ensure the feature learning of malicious apps and fit the fact that benign software is more than malicious software in the real world, our final dataset has 75,000 benign applications and 25,000 malicious applications as shown in Table II. All the APK files of our dataset are available in github[3], by this researchers can conduct reproducible study.

Since the Android malware detection is a binary classification task, we adopt widely used metrics to measure its performance. The descriptions of our used metrics are as follows:

- *True Positive* (TP): the number of samples correctly classified as malware.
- *True Negative* (TN): the number of samples correctly classified as benign samples.
- *False Positive* (FP): the number of samples incorrectly classified as malware.
- *False Negative* (FN): the number of samples incorrectly classified as benign samples.
- *Precision=TP/(TP+FP)*. The correct rate of detection.
- *Recall=TP/(TP+FN)*. The percentage of malware that are successfully detected.
- *F1=2∗Precision∗Recall/(Precision+Recall)*. A comprehensive metric of detection.

B. Social Network Properties Distribution

Our first study is to research the difference in social network properties between function call graphs of benign apps and malicious apps. Given an APK file, we first perform static analysis to obtain the function call graph and then dig out the 57 social network properties by in-deep social network analysis. After collecting the 57 social network properties of all 100,000 apps in our dataset, we investigate the differences in these properties between benign and malicious applications through statistical analysis.

As shown in Table III, we present the mean, median, interquartile range, and mode of these social network properties of benign and malicious applications, respectively. From the results in Table III, we can see that the mean, median, and interquartile range of some social network properties are similar between benign and malicious apps. For instance, the mean of *Diameter*, the median of *Periphery*, the interquartile range of *Average Katz Centrality* and so on. However, some social network properties differ greatly between benign and malicious applications, not only in node centrality-related properties but also in many others, especially for several types of triads. For example, the mean, median, and interquartile range of *021C* for benign apps are 7,123.8, 5,458, and 9,141.3 while are 9,104.2, 7,886, and 11,071 for malicious apps, respectively. This is mainly because malicious apps usually have simpler logical functions and structures than benign apps,

[2]More detailed information of parameters are available in the official website: https://scikit-learn.org/.

[3]https://github.com/johorun/SNADroid.

TABLE III
THE MEAN, MEDIAN, INTERQUARTILE RANGE, AND MODE OF 57 DIFFERENT SOCIAL NETWORK PROPERTIES IN BENIGN APPS AND MALICIOUS APPS

Social Network Properties	Mean		Median		Interquartile Range		Mode	
	Benign	Malware	Benign	Malware	Benign	Malware	Benign	Malware
Nodes Number	3701.6	3685.3	3120	3125	4752.3	4291.3	86	98
Edges Number	7271.1	8707.4	5914	7644	9317.0	10356.5	103	104
Density	0.0021	0.0022	0.0007	0.0008	0.0014	0.0012	0.0141	0.0109
Diameter	2.0861	2.0260	1.6529	1.8333	0.6690	0.6778	4.6667	3.1667
Radius	1.3607	1.3377	1.1893	1.2667	0.3719	0.3124	2	1.6667
Eccentricity Number	9.6263	9.7334	9.6582	9.9696	1.8166	1.5906	8.1512	4.4388
Periphery	3.4474	2.9491	2.3204	2.6172	0.6965	0.6472	5	5
Reciprocity	0.0011	0.0005	0.0006	0.0002	0.0017	0.0007	0	0
Algebraic Connectivity	0.0001	0.0047	0	0	0	0	0	0
Degree Assortativity Coefficient	-0.1429	-0.1166	-0.1084	-0.1113	0.0706	0.0516	-0.5885	-0.2030
Bridge Number	1621.6	1443.2	1310	1178	2128	1701.3	53	76
Average Degree Centrality	0.0042	0.0044	0.0013	0.0015	0.0028	0.0025	0.0282	0.0219
Maximal Degree Centrality	0.0891	0.0923	0.0699	0.0718	0.0219	0.0208	0.2588	0.2680
Minimal Degree Centrality	0.0014	0.0015	0.0003	0.0003	0.0007	0.0006	0.0118	0.0103
Average Katz Centrality	0.0253	0.0238	0.0152	0.0140	0.0167	0.0146	0.1069	0.1008
Maximal Katz Centrality	0.3042	0.2881	0.3076	0.2947	0.1433	0.1298	0.2122	0.1379
Minimal Katz Centrality	0.0214	0.0198	0.0124	0.0110	0.0136	0.0115	0.0947	0.0908
Average Harmonic Centrality	0.0032	0.0033	0.0014	0.0015	0.0024	0.0024	0.0201	0.0127
Maximal Harmonic Centrality	0.1186	0.1139	0.1109	0.1128	0.0322	0.0267	0.1471	0.0619
Minimal Harmonic Centrality	0	0	0	0	0	0	0	0
Average Closeness Centrality	0.0028	0.0029	0.0011	0.0012	0.0020	0.0020	0.0180	0.0121
Maximal Closeness Centrality	0.0907	0.0874	0.0818	0.0838	0.0291	0.0211	0.1420	0.0561
Minimal Closeness Centrality	0	0	0	0	0	0	0	0
Average Betweenness Centrality	1.97E-05	1.35E-05	2.06E-06	1.78E-06	6.61E-06	7.45E-06	2.35E-04	4.05E-05
Maximal Betweenness Centrality	9.97E-04	7.34E-04	4.27E-04	2.54E-04	5.75E-04	6.42E-04	8.82E-03	2.15E-03
Minimal Betweenness Centrality	0	0	0	0	0	0	0	0
Clique Number	11817.97	13570.3	9735	12090.5	15156	16307.5	189	209
Maximal Clique Number	6704.01	7959.7	5467.5	6932	8613.3	9476.8	103	97
Largest Clique Size	3.8770	4.0241	4	4	0	0	4	4
Average Triangles Number	0.5929	0.8503	0.5535	0.8347	0.3224	0.5112	0	0.2143
Transitivity	0.0103	0.0097	0.0102	0.0093	0.0058	0.0039	0	0.0054
Average Clustering Coefficient	0.0263	0.0298	0.0268	0.0298	0.0092	0.0089	0	0.0340
Strongly Connected Components Number	3688	3679	3100	3122	4729	4280	86	98
Weakly Connected Components Number	200.6	90.9	70	40	397	74	3	6
Attracting Components Number	1665.6	1756.5	1464	1647	1945	1883.3	60	75
Cycles Number	3765.5	5110.2	2823	4359.5	4516.3	6183.8	20	12
Simple Cycles Number	7440.2	79.1	8	4	38	14	0	0
Average Shortest Path Length	3.1308	2.8745	3.1895	2.9349	1.4420	0.7570	1.7024	1.2662
Maximal Shortest Path Length	11.451	11.178	13	11	5	4	14	11
Minimal Shortest Path Length	1	1	1	1	0	0	1	1
Shortest Path Number	3.71E+04	3.88E+04	2.79E+04	2.98E+04	4.88E+04	4.63E+04	205	139
003	2.56E+10	2.33E+10	5.04E+09	5.06E+09	3.41E+10	2.96E+10	9.44E+04	1.43E+05
012	4.28E+07	4.91E+07	1.79E+07	2.34E+07	6.41E+07	7.11E+07	7184	8561
102	2.91E+04	1.70E+04	4.93E+03	2.57E+03	5.23E+04	1.62E+04	0	0
021D	3.60E+04	5.47E+04	2.70E+04	4.75E+04	4.13E+04	5.97E+04	565	643
021U	1.65E+05	2.18E+05	7.29E+04	1.11E+05	1.99E+05	2.84E+05	95	25
021C	7123.8	9104.2	5458	7886	9141.3	11071	74	33
111D	10.593	9.725	2	1	21	12	0	0
111U	40.820	38.001	18	8	65	44	0	0
030T	804.78	1117.4	650	965	942	1344	0	7
030C	0.8134	0.6094	0	0	1	1	0	0
201	0.5178	0.8209	0	0	0	0	0	0
120D	0.5698	0.2125	0	0	1	0	0	0
120U	8.9381	4.8215	7	0	17	9	0	0
120C	0.4426	0.2581	0	0	1	0	0	0
210	0.0077	0.0086	0	0	0	0	0	0
300	0.0014	0.0006	0	0	0	0	0	0

with less complexity and nesting of function calls. Intuitively, malicious apps have more *102*, *120D*, *120U*, and *120C* types of triads while benign apps have more *210D*, *210U*, *210C*, *201* types of triads. Furthermore, the other graph properties such as the *Average Shortest Path Length* and the *Average Triangles Number* of malware are also much smaller than those of benign apps. The such difference indicates that the structure of call graphs of benign and malicious apps are different.

Additionally, as for *Minimal Harmonic Centrality*, *Minimal Closeness Centrality*, *Minimal Betweenness Centrality*, and *Minimal Shortest Path Length*, the mean, median, interquartile range, and mode of benign and malicious apps are the same. We also check all the values of these four properties from our dataset, and the result suggests that there is no difference in these four social network properties between benign and malicious apps. In conclusion, there are some big differences in several social network properties between benign and malicious apps, such as *Weakly Connected Component Number*, *Average Triangles Number*, some node centrality and triads properties. These social network properties all reveal the functional behavior complexity and logical characteristics of a program, so that we can utilize them to better characterize the difference between malware and benign applications.

C. *Social Network Properties Ranking*

In this part, we adopt three feature ranking algorithms (*i.e.,* T-test, normalized mutual information, and maximal information coefficient) to obtain the rankings of 57 social network properties on 100,000 Android apps. The ranking results for 57 social network properties are presented in Table IV. It is obvious that the rankings of 57 properties are different when we apply different feature ranking methods. For instance, the top 1 ranking is *Weakly Connected Component Number* when we apply the T-test to rank these properties, while is *Average Katz Centrality* and *Average Triangles Number* for normalized mutual information and maximal information coefficient, respectively. It is reasonable because the definitions and analysis processes of different feature ranking methods are different. However, the top-ranking properties computed by NMI and MIC share a similar set. There are 20 same properties among the top 21 ranking properties obtained by NMI and MIC, such as *Average Katz Centrality*, *003*, and *Degree Assortativity Coefficient*. The only different property among the top 21 ranking properties by normalized mutual information is *Maximal Closeness Centrality*, while is *Shortest Path Number* for maximal information coefficient. The such result suggests that the ranking results generated by NMI and MIC are consistent.

Although the ranking result of T-test is quite different from NMI and MIC, there are 4 same properties among the top 20 ranking properties generated by T-test, NMI, and MIC. They are *Average Triangles Number*, *Average Katz Centrality*, *Degree Assortativity Coefficient*, and *Maximal Harmonic Centrality*. This is mainly because compared with NMI and MIC, T-test is a statistical test, which is more inclined to capture the distribution and statistical characteristics between targets. When ranking the social property features of the global graph, T-test can capture the differences more directly and concisely, while NMI and MIC are more suitable for the association mining of complex features, which may introduce new noise and cause misjudgment. In the subsequent malware detection experiments, it is also proved that the features screened by T-test are more effective, and we will discuss this in Section III-D. In addition, no matter which feature ranking methods we use to generate the rankings, *Minimal Harmonic Centrality*, *Minimal Closeness Centrality*, *Minimal Betweenness Centrality*, and *Minimal Shortest Path Length* are almost the lowest. It happens because there is no difference in these four social network properties between benign and malicious applications.

In conclusion, *Average Triangles Number*, *Average Katz Centrality*, *Degree Assortativity Coefficient*, and *Maximal Harmonic Centrality* can be the most informative social network properties in differentiating malware from benign apps. *Minimal Harmonic Centrality*, *Minimal Closeness Centrality*, *Minimal Betweenness Centrality*, and *Minimal Shortest Path Length* are useless for detecting Android malware.

D. Malware Detection Effectiveness

In this part, we focus on evaluating the effectiveness of social network properties on Android malware detection. We use a custom Android malware detection system named *SNADroid* for experiment evaluation. In particular, we commence our evaluations from the following three aspects:

- *Malware detection by individual social network property: We use individual social network property to detect malware.*
- *Malware detection by different sets of social network properties: We select different sets of properties to detect malware according to their rankings obtained by three feature ranking methods (i.e., T-test, NMI, and MIC) and four machine learning classifiers (i.e., 1NN, 3NN, Random Forest, and Decision Tree).*
- *In-depth analysis: Through manual analysis and combined with experimental results, we further explain why some properties can greatly improve the detection effect of malware while others cannot.*

1) Malware detection by individual social network property: To validate the capability of different social network properties on detecting Android malware, we totally conduct 57 experiments by using individual property to train classifiers and detect malware. Table V presents the results, which include the f-measure and ranking by adopting different classifiers (*i.e.,* 1NN, 3NN, Random Forest, and Decision Tree). The last column of Table V shows the average value of the four f-measures generated by four different classifiers. From the results in Table V, we can observe that the *Average Triangles Number* is the most informative social network property among these 57 properties on Android malware detection no matter what machine learning algorithms are used to train classifiers. The f-measure can be up to 65% when we select Decision Tree to train a model for classification. Social network properties with the top 3 average f-measures are *Average Triangles Number*, *021D*, and *Average Shortest Path Length*, which are the same as the top 3 ranking results generated by the maximal information coefficient. Moreover, as shown in Table IV, there are 4 same properties (*i.e., Average Triangles Number*, *Average Katz Centrality*, *Degree Assortativity Coefficient*, and *Maximal Harmonic Centrality*) among the top 20 ranking properties generated by T-test, normalized mutual information, and maximal information coefficient. The average f-measures of these four properties are 61.05%, 50.75%, 49.75%, and 48.74%, respectively. Similarly, the f-measure of 4 useless social network properties (*i.e., Minimal Harmonic Centrality*, *Minimal Closeness Centrality*, *Minimal Betweenness Centrality*, and *Minimal Shortest Path Length*) are both 0%. Such result demonstrates that they have no ability to detect Android malware. In one word, through the ranking results in Table IV and f-measure results in Table V, it's observed that the *Average Triangles Number* is the most impactful social network property to detect Android malware among our 57 selected properties.

2) Malware detection by different sets of social network properties: From the above subsections, we can obtain the rankings of these 57 social network properties on Android app classification. To validate the effectiveness and evaluate the capability of different social network properties for malware detection, we apply four machine learning algorithms (*i.e.,* 1NN, 3NN, Random Forest, and Decision Tree) to construct classifiers. The classification results are presented in Figure 3.

From Figure 3, we observe that the f-measure and accuracy differ according to the selected machine learning algorithms.

TABLE IV
57 SOCIAL NETWORK PROPERTIES RANKED BY T-TEST, NORMALIZED MUTUAL INFORMATION, AND MAXIMAL INFORMATION COEFFICIENT

Rank	T-test		Normalized Mutual Information		Maximal Information Coefficient	
	Score	Social Network Properties	Score	Social Network Properties	Score	Social Network Properties
1	92.1346	Weakly Connected Components Number	0.2276	Average Katz Centrality	0.2777	Average Triangles Number
2	82.2572	Reciprocity	0.2276	003	0.2525	Average Shortest Path Length
3	80.2028	Average Triangles Number	0.2275	Degree Assortativity Coefficient	0.2393	021D
4	60.8373	021D	0.2275	Maximal Harmonic Centrality	0.2299	Maximal Betweenness Centrality
5	59.0988	120U	0.2275	012	0.2223	Average Clustering Coefficient
6	47.9220	030T	0.2273	Average Harmonic Centrality	0.2222	Density
7	47.4446	Average Katz Centrality	0.2273	Minimal Katz Centrality	0.2170	Eccentricity Number
8	47.1155	102	0.2272	Average Shortest Path Length	0.2150	Transitivity
9	46.2216	Degree Assortativity Coefficient	0.2271	Maximal Clique Number	0.2148	Maximal Clique Number
10	45.6354	Cycles Number	0.2265	Eccentricity Number	0.2131	Average Degree Centrality
11	41.9794	120D	0.2262	Transitivity	0.2125	Average Closeness Centrality
12	41.8129	Average Shortest Path Length	0.2262	Maximal Degree Centrality	0.2092	012
13	38.3122	021C	0.2256	Average Degree Centrality	0.2088	Degree Assortativity Coefficient
14	33.7793	120C	0.2255	Average Clustering Coefficient	0.2081	Minimal Katz Centrality
15	32.3855	Largest Clique Size	0.2251	Average Closeness Centrality	0.2078	Average Harmonic Centrality
16	29.9640	Edges Number	0.2237	Density	0.2074	003
17	28.5950	Maximal Shortest Path Length	0.2211	Average Triangles Number	0.2051	Average Katz Centrality
18	24.8178	Maximal Harmonic Centrality	0.2111	021U	0.2042	Shortest Path Number
19	23.7565	Clique Number	0.2055	Maximal Closeness Centrality	0.2041	Maximal Degree Centrality
20	21.8198	Bridge Number	0.2003	Maximal Betweenness Centrality	0.2004	021U
21	21.2898	030C	0.1969	021D	0.1936	Maximal Harmonic Centrality
22	20.5907	021U	0.1885	Shortest Path Number	0.1933	021C
23	20.4283	Transitivity	0.1428	Clique Number	0.1925	Maximal Closeness Centrality
24	19.3343	Maximal Betweenness Centrality	0.1392	Reciprocity	0.1900	Maximal Shortest Path Length
25	17.5675	Maximal Degree Centrality	0.1344	102	0.1900	Clique Number
26	14.7496	012	0.1302	021C	0.1875	Cycles Number
27	13.1881	Maximal Clique Number	0.1231	Edges Number	0.1852	Edges Number
28	11.6695	Average Betweenness Centrality	0.1206	Maximal Shortest Path Length	0.1720	Periphery
29	11.3592	Diameter	0.1061	Cycles Number	0.1716	030T
30	10.6431	Attracting Components Number	0.0969	Periphery	0.1712	Diameter
31	9.3745	Minimal Katz Centrality	0.0935	Diameter	0.1669	Radius
32	9.2361	Radius	0.0857	Minimal Degree Centrality	0.1647	Minimal Degree Centrality
33	9.2324	Maximal Katz Centrality	0.0853	Nodes Number	0.1636	Nodes Number
34	8.6244	003	0.0842	Strongly Connected Components Number	0.1623	Strongly Connected Components Number
35	7.9743	Algebraic Connectivity	0.0756	Radius	0.1574	Bridge Number
36	7.6083	Average Harmonic Centrality	0.0701	030T	0.1560	Attracting Components Number
37	7.4903	Periphery	0.0654	Attracting Components Number	0.1469	Reciprocity
38	7.2986	Eccentricity Number	0.0643	Bridge Number	0.1347	Average Betweenness Centrality
39	6.9116	Maximal Closeness Centrality	0.0595	111U	0.1270	102
40	6.1165	Shortest Path Number	0.0548	Average Betweenness Centrality	0.1212	Weakly Connected Components Number
41	5.3164	111U	0.0470	Weakly Connected Components Number	0.1175	111U
42	4.1140	111D	0.0424	111D	0.0757	111D
43	4.0607	201	0.0371	Maximal Katz Centrality	0.0667	Simple Cycles Number
44	3.4272	Density	0.0345	120U	0.0661	Maximal Katz Centrality
45	3.4272	Average Clustering Coefficient	0.0330	Simple Cycles Number	0.0584	120U
46	3.2890	300	0.0307	120D	0.0279	120D
47	3.2572	Average Closeness Centrality	0.0259	120C	0.0249	120C
48	3.0424	Average Degree Centrality	0.0177	030C	0.0209	030C
49	2.0233	Simple Cycles Number	0.0173	201	0.0166	201
50	1.3608	Minimal Degree Centrality	0.0127	Algebraic Connectivity	0.0129	Largest Clique Size
51	0.8126	Nodes Number	0.0120	Largest Clique Size	0.0022	Algebraic Connectivity
52	0.6316	210	0.0013	300	0.0001	210
53	0.4446	Strongly Connected Components Number	0.0007	210	0.0001	300
54	0.0000	Minimal Betweenness Centrality	0.0000	Minimal Betweenness Centrality	0.0000	Minimal Betweenness Centrality
55	0.0000	Minimal Closeness Centrality	0.0000	Minimal Closeness Centrality	0.0000	Minimal Closeness Centrality
56	0.0000	Minimal Harmonic Centrality	0.0000	Minimal Harmonic Centrality	0.0000	Minimal Harmonic Centrality
57	0.0000	Minimal Shortest Path Length	0.0000	Minimal Shortest Path Length	0.0000	Minimal Shortest Path Length

For instance, the accuracy of 1NN is 81.9% when we choose the ranking first social network property (*i.e., Average Triangles Number*) obtained by maximal information coefficient to detect malware while is 82.8%, 80.5%, and 82.5% for 3NN, Random Forest, and Decision Tree, respectively. In addition, we can achieve better performance when we select Decision Tree to build a classifier and use it to detect malware. For example, the accuracy can be up to 91.6% when the classification algorithm is Decision Tree, while is 90.0%, 88.5%, and 88.9% for 1NN, 3NN, and Random Forest. Moreover, Figure 3 also presents that the detection effectiveness can be different when we select different ranked social network properties as features to detect malware. This happens because that the ranking of a social network property can be different when we apply different feature ranking methods. In fact, the detection performance will both be better as the number of top-ranking properties increases regardless of which feature ranking methods are used. However, the effectiveness in detecting malware will decrease slightly as more social network properties are added. It is normal because the top-ranked properties have a stronger ability to differentiate between malware and benign applications. Correspondingly, lower-ranking properties are less efficient for malware detection, and can even lead to some false positives when adding these properties into classifiers. For instance, the accuracy of 1NN is 69.1% when we select the top one property ranked by T-test to detect malware while is 88.0% when the number of selected properties increased to the top 10. Unfortunately, it drops to 83.7% when we employ

TABLE V
THE F-MEASURE AND CORRESPONDING RANKINGS OF 57 SOCIAL NETWORK PROPERTIES GENERATED BY 1NN, 3NN, RANDOM FOREST, AND DECISION TREE

Social Network Property	1NN		3NN		Random Forest		Decision Tree		Average	
	F1	Ranking	F1	Ranking	F1	Ranking	F1	Ranking	F1	Ranking
Average Triangles Number	0.6420	1	0.6389	1	0.5119	1	0.6493	1	0.6105	1
021D	0.5981	14	0.6029	4	0.3806	2	0.6174	3	0.5498	2
Average Shortest Path Length	0.6216	3	0.6049	3	0.3234	6	0.6240	2	0.5435	3
Maximal Degree Centrality	0.6106	4	0.5887	5	0.2848	9	0.6011	7	0.5213	4
Maximal Clique Number	0.6097	5	0.5856	6	0.2712	14	0.6054	6	0.5180	5
Eccentricity Number	0.6023	10	0.5855	7	0.2745	12	0.6093	5	0.5179	6
Average Harmonic Centrality	0.6048	8	0.5777	10	0.2742	13	0.5838	14	0.5101	7
021U	0.5760	22	0.5708	16	0.2915	7	0.5934	10	0.5079	8
Average Katz Centrality	0.6050	7	0.5825	8	0.2574	19	0.5851	13	0.5075	9
003	0.6074	6	0.5767	12	0.2286	28	0.6110	4	0.5059	10
Transitivity	0.5967	15	0.5778	9	0.2758	11	0.5729	17	0.5058	11
Minimal Katz Centrality	0.6016	11	0.5770	11	0.2640	16	0.5739	16	0.5041	12
012	0.5940	19	0.5664	22	0.2569	20	0.5989	8	0.5040	13
Maximal Closeness Centrality	0.5852	21	0.5680	19	0.2546	21	0.5918	11	0.4999	14
Degree Assortativity Coefficient	0.5954	17	0.5673	20	0.2296	27	0.5978	9	0.4975	15
Shortest Path Number	0.5601	23	0.5703	17	0.2514	23	0.5802	15	0.4905	16
Maximal Harmonic Centrality	0.5953	18	0.5664	21	0.2025	35	0.5852	12	0.4874	17
Average Clustering Coefficient	0.6025	9	0.5757	13	0.2186	31	0.4924	24	0.4723	18
Maximal Betweenness Centrality	0.6266	2	0.6073	2	0.3301	5	0.2992	38	0.4658	19
Average Degree Centrality	0.5992	13	0.5744	14	0.1896	38	0.4937	23	0.4642	20
021C	0.5040	25	0.5356	24	0.2693	15	0.5474	18	0.4641	21
Cycles Number	0.4804	28	0.5075	28	0.3343	3	0.5271	22	0.4623	22
Edges Number	0.4983	26	0.5227	26	0.2905	8	0.5366	20	0.4620	23
Clique Number	0.5155	24	0.5339	25	0.2415	25	0.5470	19	0.4595	24
Maximal Shortest Path Length	0.4933	27	0.5197	27	0.2804	10	0.5316	21	0.4562	25
Average Closeness Centrality	0.5963	16	0.5696	18	0.1926	37	0.4655	27	0.4560	26
Density	0.6006	12	0.5719	15	0.2067	34	0.4323	32	0.4529	27
030T	0.4317	35	0.4381	35	0.3302	4	0.4564	29	0.4141	28
Nodes Number	0.4412	34	0.4772	29	0.2286	29	0.4888	25	0.4090	29
Strongly Connected Components Number	0.4424	33	0.4662	31	0.2267	30	0.4823	26	0.4044	30
Diameter	0.4534	30	0.4617	32	0.2403	26	0.4565	28	0.4030	31
Periphery	0.4516	31	0.4531	33	0.2103	32	0.4420	30	0.3893	32
Radius	0.4039	39	0.4350	36	0.2518	22	0.4056	34	0.3741	33
Attracting Components Number	0.4262	36	0.4329	37	0.1834	39	0.4380	31	0.3701	34
Minimal Degree Centrality	0.4486	32	0.4709	30	0.2068	33	0.3534	36	0.3699	35
Bridge Number	0.4112	38	0.4311	38	0.2014	36	0.4291	33	0.3682	36
102	0.4657	29	0.4495	34	0.1290	43	0.3630	35	0.3518	37
Reciprocity	0.3766	41	0.3843	40	0.2604	18	0.3206	37	0.3355	38
Average Betweenness Centrality	0.5903	20	0.5594	23	0.1238	44	0.0498	46	0.3308	39
Weakly Connected Components Number	0.3595	43	0.3571	42	0.1760	41	0.2921	39	0.2962	40
111U	0.3163	44	0.3064	45	0.2607	17	0.2908	40	0.2936	41
Maximal Katz Centrality	0.2483	46	0.3307	44	0.2442	24	0.2443	41	0.2669	42
120U	0.4178	37	0.3844	39	0.0744	46	0.0751	45	0.2379	43
Simple Cycles Number	0.2759	45	0.3423	43	0.1640	42	0.1385	43	0.2302	44
111D	0.2292	47	0.2441	46	0.1771	40	0.1808	42	0.2078	45
201	0.3759	42	0.0908	48	0.0886	45	0.0890	44	0.1611	46
030C	0.0604	49	0.3840	41	0.0000	51	0.0000	51	0.1111	47
120D	0.3973	40	0.0021	51	0.0015	49	0.0017	49	0.1007	48
Largest Clique Size	0.1781	48	0.1436	47	0.0000	52	0.0000	52	0.0804	49
120C	0.0194	50	0.0102	49	0.0109	47	0.0122	47	0.0132	50
Algebraic Connectivity	0.0058	51	0.0058	50	0.0057	48	0.0058	48	0.0058	51
210	0.0014	52	0.0012	52	0.0004	50	0.0002	50	0.0008	52
300	0.0002	53	0.0000	53	0.0000	53	0.0000	53	0.0001	53
Minimal Harmonic Centrality	0.0000	54	0.0000	54	0.0000	54	0.0000	54	0.0000	54
Minimal Closeness Centrality	0.0000	55	0.0000	55	0.0000	55	0.0000	55	0.0000	55
Minimal Betweenness Centrality	0.0000	56	0.0000	56	0.0000	56	0.0000	56	0.0000	56
Minimal Shortest Path Length	0.0000	57	0.0000	57	0.0000	57	0.0000	57	0.0000	57

all the social network properties to conduct malware detection.

3) In-depth analysis: In this part, we further analyze the previous experimental results to clarify the causes of the difference in feature distribution between benign and malicious apps. From Figure 3, we can find that all experimental results tend to be the best when the top 15-20 features are selected, which is also in line with the fact that the top 20 features of different measurement methods in Table IV are largely consistent. In addition, the features screened by T-test have better performance than NMI and MIC. For a clearer illustration, we draw a Venn diagram in Figure 4 for the top 25 features sorted by the three features. It can be seen that the number of intersection features between T-test and other methods is 15, which is also the number of features with the highest classification performance. However, the number of intersections between NMI, MIC and other methods is 25. Obviously, the classification effect of the experiment has declined at this time. Therefore, we can conclude that the optimal and minimum social property feature set is the intersection of T-test and feature sets of other methods, and the set size is 15. Therefore, we can screen out the most effective 15 social properties and divide them into three categories: whole graph topology properties, node properties, and node cluster properties (shown in Table VI).

a) Whole graph topology properties: This kind of properties describe the differences of benign and malicious apps in

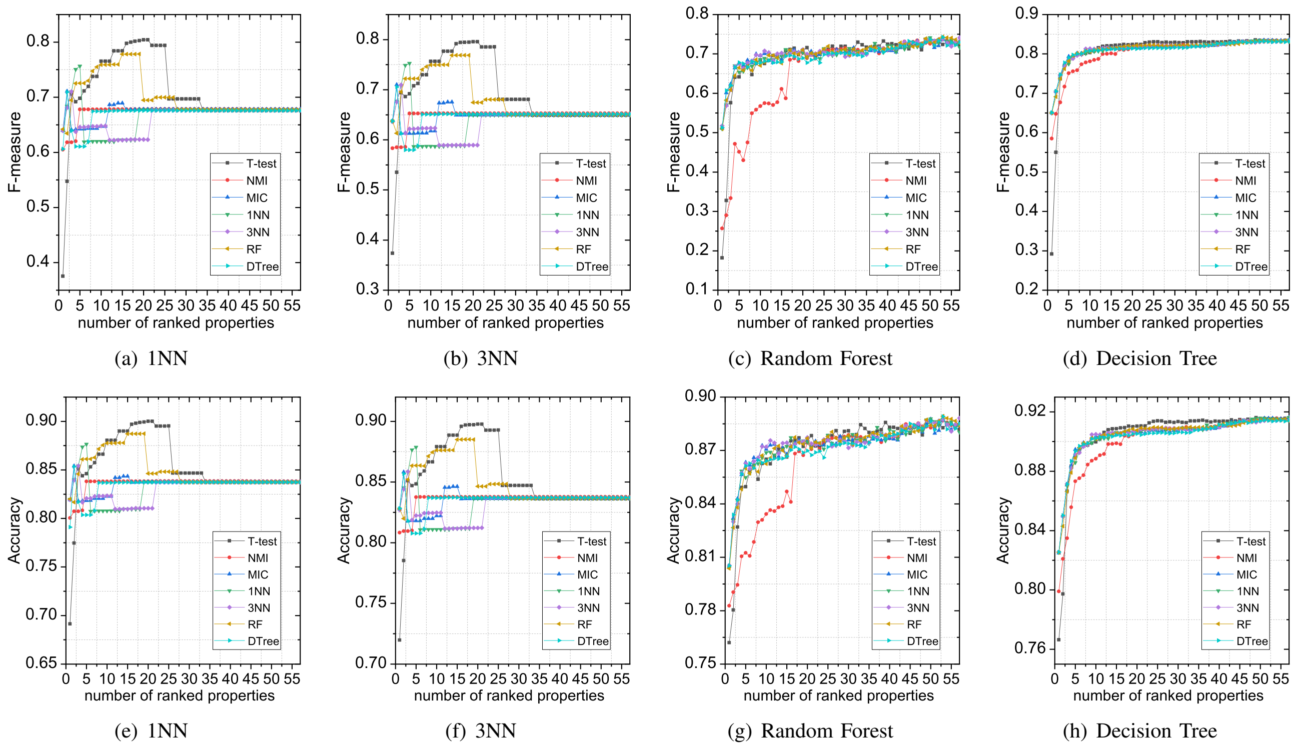

Fig. 3. The f-measure and accuracy of four classifiers when increasing the number of social network properties according to their rankings obtained by T-test, NMI, MIC, 1NN, 3NN, Random Forest, and Decision Tree, respectively

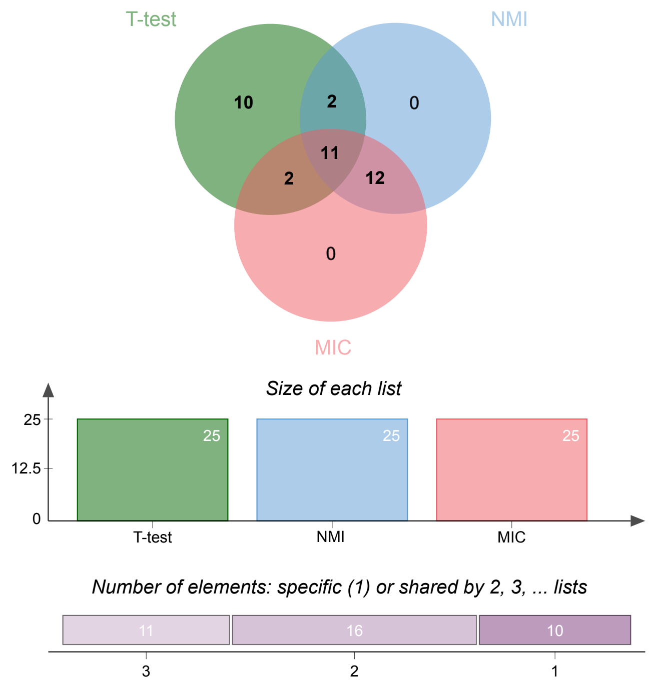

Fig. 4. The Venn diagrams of the top 25 effective features obtained by different feature ranking methods.

TABLE VI
THE MOST EFFECTIVE 15 SOCIAL PROPERTIES IN GRAPH-BASED MALWARE DETECTION

Category	Social Property
whole graph topology	Average Triangles Number, Average Shortest Path Length, Transitivity Edges Number, Clique Number, Maximal Shortest Path Length
node	Maximal Degree Centrality, Average Katz Centrality, Degree Assortativity Coefficient, Maximal Harmonic Centrality, Maximal Betweenness Centrality
node cluster	021D, 021U, 003, 012, 021C

a whole graph scale. As a result of the differences in benign and malicious apps functionality, the overall characteristics of the function call graph may also be different. For example, as shown in Table III, malware and benign apps have roughly the same *Nodes Number*, while the *Clique Number* of benign apps is smaller. This means that the maximum clique in benign apps call graphs has fewer vertices, that is, the calling relationship between functions is more sparse and difficult to form cliques in benign apps. The same is true for the *Average Triangles Number*, benign apps are much smaller than malware in this property. This indicates that malware is more likely to own triangle call loops, while benign software has deeper and more sparse function calls, making it relatively difficult to form triangle loops.

In our analysis, we find that the distribution of the other whole graph topology properties also conforms to the intuitive features of benign and malicious apps, and these whole graph topology properties are an important reference for graph-based malware detection.

b) Node properties: These properties measure the position and importance of a single node in the graph. Using individual node social centrality to handle sensitive functions of call graphs for malware detection has been shown to be effective [4], [5]. However, researchers lack an understanding of the overall function centrality distribution between benign and malicious apps. At the scale of the whole graph, *Degree*, *Katz*, *Harmonic*, *Betweenness* and their derived properties are still valid features to distinguish between malware and benign applications. This can help researchers more directly obtain general analysis results when the sensitive functions are unknown.

c) Node cluster properties: As far as we know, the current work of graph-based malware detection focuses on the single node or global call graph properties. Through social network analysis of triads, we observe that the distribution of the clusters formed by three functions and their calling relationships are also very different in malware and benign software. Thus, we name this kind of characteristics as *node cluster property*.

For example, in malware, the proportion of simple triads types (*i.e.,* 021D, 021U, 021C) is much higher than that of complex types (*i.e.,* 120D, 120U, 120C), and the situation is reversed in benign apps (although the absolute number of triads in benign apps is smaller than that in malware). To a certain extent, this reflects the difference between benign and malicious apps in terms of function calls. We suggest that researchers pay attention to the consideration of cluster node distribution properties in future malware analysis work, especially 003, 012, 021D, 021U, and 021C cluster types.

E. Time Overhead

In addition to effectiveness, another significant factor affecting the practicality of malware detection is the runtime overhead. In this section, we evaluate the running overhead of our work and use the Mean time to detect (MTTD) of each sample as the evaluation metric. The experiment is being repeated 10 times to obtain MTTD. In general, the running time overhead consists of three stages: preprocessing time, feature processing time, and classification time.

The preprocessing stage mainly includes using Androguard for program static analysis and program function call graph construction. At this stage, the MTTD of each sample in our dataset is 0.31 seconds.

The feature processing stage is mainly to conduct social network analysis on the program function call graph and obtain various social network properties. At this stage, the analysis time will vary depending on the different social properties and the analysis time overhead corresponding to each social property is shown in the Table VII. We can see that the most time-consuming social properties are those related to node centrality, such as Maximal Harmonic Centrality, Average Katz Centrality, etc. This is because compared with other simple global statistical features, these features require calculating the centrality of all graph nodes one by one before obtaining the statistical results.

The classification stage is mainly the process of the classifier classifying the feature vector. We take *SNADroid* as an example to give reference results. Since *SNADroid* has fewer feature dimensions (up to 57 dimensions), its classification speed is extremely fast. Regardless of whether KNN or random forest is used, the average classification time for each sample is less than 0.001 seconds.

Overall, our work is lightweight, and even with high time complexity social network node properties, it can run several times faster than other works [4].

TABLE VII
THE FEATURE PROCESSING TIME OVERHEAD OF EACH SOCIAL PROPERTY

Social Network Property	**Feature Processing (s)**
Nodes Number	<0.001
Edges Number	<0.001
Density	<0.001
Diameter	<0.001
Radius	<0.001
Eccentricity Number	<0.001
Periphery	<0.001
Reciprocity	<0.001
Algebraic Connectivity	<0.001
Degree Assortativity Coefficient	<0.001
Bridge Number	<0.001
Average Degree Centrality	0.014
Maximal Degree Centrality	0.014
Minimal Degree Centrality	0.014
Average Katz Centrality	0.746
Maximal Katz Centrality	0.746
Minimal Katz Centrality	0.746
Average Harmonic Centrality	0.793
Maximal Harmonic Centrality	0.793
Minimal Harmonic Centrality	0.793
Average Closeness Centrality	0.501
Maximal Closeness Centrality	0.501
Minimal Closeness Centrality	0.501
Average Betweenness Centrality	0.619
Maximal Betweenness Centrality	0.619
Minimal Betweenness Centrality	0.619
Clique Number	0.003
Maximal Clique Number	0.003
Largest Clique Size	0.003
Average Triangles Number	0.07
Transitivity	<0.001
Average Clustering Coefficient	0.017
Strongly Connected Components Number	0.009
Weakly Connected Components Number	0.007
Attracting Components Number	0.007
Cycles Number	0.002
Simple Cycles Number	0.002
Average Shortest Path Length	0.011
Maximal Shortest Path Length	0.011
Minimal Shortest Path Length	0.011
Shortest Path Number	0.011
003	0.005
012	0.005
102	0.005
021D	0.005
021U	0.005
021C	0.005
111D	0.005
111U	0.005
030T	0.005
030C	0.005
201	0.005
120D	0.005
120U	0.005
120C	0.005
210	0.005
300	0.005

IV. RELATED WORK

Android Malware Detection. At present, many Android malware detection works that rely on syntactic features [8], [9], [11], [31]–[34] or program semantics [4]–[7], [10], [12]–[17],

[20], [35]–[49] have been proposed. Wang *et al.* [9] employ three feature ranking methods to sort individual permissions by their risk and use the top risky permissions detecting malware. Unfortunately, this method is less efficient in detecting Android malware as a result of missing program behavior semantics. *Drebin* [11] processing apps for detailed features with extensive static analysis. However, it only focuses on the existence of some specific strings (*i.e.,* restricted API calls name) and thus it may be bypassed by obfuscation because of ignoring program behavior information. Zhu *et al.* [48] comprehensively consider permission, API calls, and system events, but their accuracy rate is not high, only 88%. Zhao *et al.* [47] try to mine the correlation of sensitive API calls with malware, but their processing is too simple to deal with complex situations. Zhang *et al.* [46] extract the API calls in operands of malware and abstract them to their package names. They focus on top-level abstraction, thus the granularity is too coarse. Also for API calls, Shen *et al.* [15] characterize API calls by constructing a complex information flow analysis structure, but this method is highly manual.

Graph-based Methods. *DroidSIFT* [13] extracts contextual API dependency graphs in a weighted manner to address the static analysis-based malware deformation problem. *Apposcopy* [14] leverages static taint analysis to generate a new program representation called *Inter-Component Call Graph* for malware detection. Nevertheless, *DroidSIFT* [13] and *Apposcopy* [14] are both time-intensive, taking 175 and 258 seconds to analyze a program, respectively. *MaMaDroid* [35] extracts abstract function call sequences from the call graph and builds a behavioral model to characterize malware. One flaw of this method is that it is easily circumvented by the specially defined packages of attackers which are similar to the official packages of Java, Android, and Google [50], and another limitation is that it requires a huge amount of memory while analyzing due to its large cumbersome features. [35]. Gao *et al.* [16] mine the local call relationship of the malware FCG for detection by graphlet sampling, however, this method lacks the overall attention and its effect is mediocre. Frenklach *et al.* [20] believe that the key to classifying application behavior lies in their common reusable building blocks, thus they propose a static Android application analysis method based on Application Similarity Graph (ASG) combined with neural networks. Ou *et al.* [17] propose a novel static sensitive subgraph-based feature for Android malware detection, named S3Featrue. Specifically, they develop a sensitive function call graph (SFCG) by extending a function call graph through tagging sensitive nodes on it to represent Android applications with high-level characteristics.

Social Network Analysis on Malware Detection. Several works [4], [5], [7], [51], [52] have been presented to detect malware using social network analysis. Alasmary *et al.* [51] first abstracts Android and IoT malware samples into Control Flow Graph (CFG) to represent the semantics of each sample. Then some social network properties such as components and average closeness centrality are extracted from the CFG to build a comparative model for detecting new IoT malware. Jang *et al.* [53] design a novel system to detect Windows malware based on the analysis of some social network properties such as the average distance obtained from a system call graph. *MalScan* [4] detects Android malware by treating the function call graph of a program as a social network and adopting node centrality analysis to obtain the sensitive API centrality in the global call graph. Similarly, *IntDroid* [5] treats function call graphs of apps as social networks and applies social network-based centrality analysis to unearth the central nodes within call graphs. After obtaining the central nodes, the average intimacies between sensitive API calls and central nodes are computed to represent the semantic features of the graphs. To detect covert malware, Wu *et al.* [7] analyze the homophily of call graphs with the social network. However, the above works only focus on a single social network property. Our work is the first attempt to examine the ability of different social network properties on Android malware characterization, and present a set of the most effective social network properties for graph-based malware analysis. We will introduce the advantages and effects of our work in the discussion section.

V. Discussion

In this section, we discuss the application value, limitations and future trends of our work.

Application Value. In our work, we extract 57 social network properties of function call graphs on 100,000 Android apps and rank them by performing three feature ranking methods. We focus on researching the ability of different social network properties on distinguishing malware from benign apps and provide theoretical guidance for other graph-based malware analysis work. To highlight the advantage of our study, we conduct a guide experiment on *MalScan*. *MalScan* only uses social network node centrality properties as features for malware detection and achieves better performance than other state-of-the-art work (such as *PerDroid* and *MaMaDroid*) [4]. Based on Table VI, we directly expand the feature vector of *MalScan* by 15 dimensions, and the performance of MalScan is improved by 0.3%-2.1% on different datasets. This demonstrates that our work is an effective guide for graph-based work using social network properties.

Limitations. As our research object is the global function call graph, some unimportant node information in graph may introduce noise. In addition, in real scenarios, malicious samples are often packed to hinder static analysis. This is indeed a common limitation of all static analysis based-work and may affect the analysis results.

Future Trends. In future work, we plan to use some social network properties to conduct local analysis of sensitive functions. Moreover, we will consider introducing an automatic unpacking module to reduce the negative impact of software packing. Furthermore, we intend to measure graph properties in more metrics, and adopt more feature ranking methods to discover more impactful properties on Android malware characterization.

VI. Conclusion

In this paper, we present the first empirical study to explore the ability of 57 social network properties on differentiating malware from benign apps. To excavate the most informative properties to detect malware, we apply three feature ranking methods to generate the rankings of these 57 social network properties. Study results indicate that the *Average Triangles Number* is the most impactful property on Android malware characterization among 57 social network properties. Moreover, we also demonstrate the capability of these properties on Android malware detection and distill the 15 most effective features as guidelines for graph-based malware analysis.

ACKNOWLEDGEMENT

We would like to thank the reviewers for their insightful comments, which have improved the quality of this paper. This work is supported by the Hubei Key Project under Grant No. 2023BAA024.

REFERENCES

[1] H. Jeong, S. P. Mason, A.-L. Barabási, and Z. N. Oltvai, "Lethality and centrality in protein networks," *Nature*, 2001.

[2] R. Guimera, S. Mossa, A. Turtschi, and L. N. Amaral, "The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles," *Proceedings of the National Academy of Sciences*, vol. 102, no. 22, pp. 7794–7799, 2005.

[3] K. Faust, "Centrality in affiliation networks," *Social Networks*, 1997.

[4] Y. Wu, X. Li, D. Zou, Y. Wei, X. Zhang, and H. Jin, "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)*, 2019.

[5] D. Zou, Y. Wu, S. Yang, A. Chauhan, W. Yang, J. Zhong, S. Dou, and H. Jin, "Intdroid: Android malware detection based on api intimacy analysis," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–32, 2021.

[6] H. M. Kim, H. M. Song, J. W. Seo, and H. K. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018, pp. 1–8.

[7] Y. Wu, D. Zou, W. Yang, X. Li, and H. Jin, "Homdroid: detecting android covert malware by social-network homophily analysis," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 216–229.

[8] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of android apps," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*, 2012.

[9] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, 2014.

[10] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, 2014.

[11] 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.

[12] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of android malware using embedded call graphs," in *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, 2013.

[13] M. Zhang, Y. Duan, 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 (CCS'14)*, 2014.

[14] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, 2014.

[15] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1231–1245, 2018.

[16] T. Gao, W. Peng, D. Sisodia, T. K. Saha, F. Li, and M. Al Hasan, "Android malware detection via graphlet sampling," *IEEE Transactions on Mobile Computing*, vol. 18, no. 12, pp. 2754–2767, 2018.

[17] F. Ou and J. Xu, "S3feature: A static sensitive subgraph-based feature for android malware detection," *Computers & Security*, vol. 112, p. 102513, 2022.

[18] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the {Google-Play} scale," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 659–674.

[19] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*, 2014.

[20] T. Frenklach, D. Cohen, A. Shabtai, and R. Puzis, "Android malware detection via an app similarity graph," *Computers & Security*, vol. 109, p. 102386, 2021.

[21] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th Working Conference on Mining Software Repositories (MSR'16)*, 2016.

[22] A. Desnos and G. Gueguen, "Androguard: Reverse engineering and pentesting for android applications." https://github.com/androguard/androguard, 2011.

[23] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, 1978.

[24] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, 1953.

[25] M. Marchiori and V. Latora, "Harmony in the small-world," *Physica A: Statistical Mechanics and its Applications*, 2000.

[26] M. Piraveenan, M. Prokopenko, and L. Hossain, "Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks," *PloS one*, 2013.

[27] M. R. Faghani and U. T. Nguyen, "A study of xss worm propagation and detection mechanisms in online social networks," *IEEE Transactions on Information Forensics and Security*, 2013.

[28] V. Batagelj and A. Mrvar, "A subquadratic triad census algorithm for large sparse networks with small maximum degree," *Social networks*, 2001.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[30] H. Sistemas, "Virustotal - free online virus, malware and url scanner," https://www.virustotal.com/, 2022.

[31] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *Proceedings of the 9th International Conference on Security and Privacy in Communication Systems (SecureComm'13)*, 2013.

[32] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies (ACMT'12)*, 2012.

[33] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: detecting malicious apps in official and alternative android markets." in *NDSS*, vol. 25, no. 4, 2012, pp. 50–52.

[34] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, 2018.

[35] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *Proceedings of the 2017 Annual Symposium on Network and Distributed System Security (NDSS'17)*, 2017.

[36] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, 2015.

[37] J. Allen, M. Landen, S. Chaba, Y. Ji, S. P. H. Chung, and W. Lee, "Improving accuracy of android malware detection with lightweight contextual awareness," in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC'18)*, 2018.

[38] W. Yang, M. Prasad, and T. Xie, "Enmobile: Entity-based characterization and analysis of mobile malware," in *Proceedings of the 40th International Conference on Software Engineering (ICSE'18)*, 2018.

[39] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, 2015.

[40] A. Machiry, N. Redini, E. Gustafson, Y. Fratantonio, Y. R. Choe, C. Kruegel, and G. Vigna, “Using loops for malware classification resilient to feature-unaware perturbations,” in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC’18)*, 2018.

[41] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “Madam: Effective and efficient behavior-based android malware detection and prevention,” *IEEE Transactions on Dependable and Secure Computing*, 2018.

[42] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, “A multi-view context-aware approach to android malware detection and malicious code localization,” *Empirical Software Engineering*, 2018.

[43] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand, “Automated synthesis of semantic malware signatures using maximum satisfiability,” in *Proceedings of the 2017 Annual Symposium on Network and Distributed System Security (NDSS’17)*, 2017.

[44] J. Garcia, M. Hammad, and S. Malek, “Lightweight, obfuscation-resilient detection and family identification of android malware,” *ACM Transactions on Software Engineering and Methodology*, 2018.

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

[46] P. Zhang, S. Cheng, S. Lou, and F. Jiang, “A novel android malware detection approach using operand sequences,” in *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*. IEEE, 2018, pp. 1–5.

[47] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, “Quick and accurate android malware detection based on sensitive apis,” in *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2018, pp. 143–148.

[48] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, “Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model,” *Neurocomputing*, vol. 272, pp. 638–646, 2018.

[49] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, “Towards multimodal learning for android malware detection,” in *2019 International conference on computing, networking and communications (ICNC)*. IEEE, 2019, pp. 432–436.

[50] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, “Android hiv: A study of repackaging malware for evading machine-learning detection,” *IEEE Transactions on Information Forensics and Security*, 2019.

[51] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, D. Nyang, and A. Mohaisen, “Analyzing, comparing, and detecting emerging malware: A graph-based approach,” *arXiv e-prints*, pp. arXiv–1902, 2019.

[52] Y. Wu, S. Dou, D. Zou, W. Yang, W. Qiang, and H. Jin, “Contrastive learning for robust android malware familial classification,” *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–14, 2022.

[53] J.-w. Jang, J. Woo, J. Yun, and H. K. Kim, “Mal-netminer: malware classification based on social network analysis of call graph,” in *Proceedings of the 23rd International Conference on World Wide Web (WWW’14)*, 2014.