Hindawi
Security and Communication Networks
Volume 2018, Article ID 8087303, 10 pages
https://doi.org/10.1155/2018/8087303

Research Article

RoughDroid: Operative Scheme for Functional Android Malware Detection

Khaled Riad [1,2] **and Lishan Ke** [3]

[1]*School of Computer Science, Guangzhou University, Guangzhou 510006, China*
[2]*Mathematics Department, Faculty of Science, Zagazig University, Zagazig 44519, Egypt*
[3]*College of Mathematics and Information Science, Guangzhou University, Guangzhou 510006, China*

Correspondence should be addressed to Lishan Ke; kelishan@gzhu.edu.cn

Received 11 June 2018; Accepted 6 August 2018; Published 20 September 2018

Academic Editor: Lianyong Qi

There are thousands of malicious applications that invade Google Play Store every day and seem to be legal applications. These malicious applications have the ability to link the malware referred to as Dresscode created for network hacking as well as scrolling information. Since Android smartphones are indispensable, there should be an efficient and also unusual protection. Therefore, Android smartphones usually continue to be safeguarded from novel malware. In this paper, we propose *RoughDroid*, a floppy analysis technique that can discover Android malware applications directly on the smartphone. *RoughDroid* is based on seven feature sets ($FS_1, FS_2, \ldots, FS_7$) from the XML *manifest* file of an Android application, plus three feature sets (FS_8, FS_9, and FS_{10}) from the Dex file. Those feature sets pass through the Rough Set algorithm to elastically classify the Android application as either benign or malicious. The experimental results mainly consider 20 most common malware families, plus three new malware families (*Grabos, TrojanDropper.Agent.BKY,* and *AsiaHitGroup*) that invade Google Play Store at 2017. According to the experimental results, *RoughDroid* has 95.6% detection performance for the malware families at 1% false-positive rate. Finally, *RoughDroid* is a lightweight approach for straightly examining downloaded applications on the smartphone.

1. Introduction

The world's most preferred mobile operating system currently is Android OS. Android surpasses Windows as the globe's most preferred OS, yet some Android applications have been discovered to privately swipe individual details from various other applications. Recently, the GadGet Hacks website stated that, after evaluating 110,150 Android applications over a duration of 3 years, the scientists located countless sets of applications that can possibly leakage delicate phone or individual information as well as permitting unapproved applications to hack the blessed information. With numerous thousands of applications in various markets, Android OS offers riches of capability to its customers. Smart devices running Android are progressively targeted by assaulters as well as contaminated with destructive software programs [1].

Google took down over 700,000 bad Android applications in 2017, that is, 70% more than in 2016 [2]. In addition to the existing malware families, three new Android malware families (*Grabos*, *TrojanDropper.Agent.BKY*, and *AsiaHitGroup*) invade Google Play Store at 2017 [3]. It appears that there is an urgent requirement for quitting the expansion of malware on Android markets and also smartphones. The Android platforms constantly attempt as well to supply numerous security solutions that stop the installment of malware applications, most significantly the Android authorization system. To carry out particular tasks on the Android device, such as capturing a picture, the application needs to clearly ask for consent from the individual throughout the setup procedure. Some customers thoughtlessly approve the installment agreement to unidentified applications without thoroughly reviewing it.

As a result, malicious software is hardly constricted from the Android permission program in training. Opening your Android phone or tablet as much as for applications and video games outside Google's protective walled yard likewise makes your device considerably a lot more at risk to malware. It is the cost you spend for a totally free software programs [4]. There

should be a method that is able to restrict these malicious applications.

1.1. Motivation. To the very best of our understanding, a huge body of research study has actually examined approaches for evaluating and also discovering Android malware applications before their setup. These approaches could be approximately classified right into techniques making use of dynamic as well as static evaluation. There are some techniques that could keep track of the habits of applications at run-time, such as TaintDroid [5], DroidRanger [6], and DroidScope [7] which are techniques that could check the actions of applications at run-time. Although run-time monitoring is really reliable in determining harmful task, it experiences a substantial cost and could not be straight used for mobile devices. On the other hand, static evaluation techniques, such as Stowaway [8] and RiskRanker [9], typically generate just a tiny run-time overhead. While these methods are scalable and also reliable, they mostly improve the handcrafted discovery patterns, which are commonly not readily available for new malware circumstances. This is behind our motivation to propose a new Android malware detection scheme that makes it possible to recognize malware straight on the smartphone throughout the setup process based on Rough Set algorithm.

1.2. Main Contributions. In this paper, we introduced *RoughDroid* that is a new broad floppy analysis malware detector on smart Android phones during the installation time by introducing robust feature extraction framework. The main contributions could be summed up as follows:

(i) **Effective Detection:** We introduce a novel scheme (*RoughDroid*) for combining floppy analysis and machine learning that is capable of identifying Android malware with high accuracy and few false alarms. Also, it is independent of manually crafted detection patterns.

(ii) **Various Features:** *RoughDroid* groups numerous features from the *manifest* file as well as *application's Dex code*. Those features are categorized into ten feature sets ($FS_1, FS_2, \ldots, FS_{10}$).

(iii) **Rough-Based Detection:** The proposed scheme considers the *adware* Android applications during the detection of malware applications. This is due to executing the detection process elastically using *Rough Set* algorithm that introduces flexible (not straight line) classification into benign and malware applications.

(iv) **Lightweight Analysis:** For efficiency, we apply linear time analysis and learning techniques that enable detecting malware on the smartphone as well as analyzing large sets of applications in a reasonable time.

Finally, the experiments with 131,611 applications and 5,560 malware samples, in addition to 158 malware applications introducing three new malware families at 2017, demonstrate the efficacy of our method for directly checking downloaded applications on the smartphone.

1.3. Organization. The rest of this paper is organized as follows: Section 2 introduces our *RoughDroid* scheme with its ten feature sets and Rough Set detection algorithm. Section 3 presents the experimental evaluation of *RoughDroid* by comparing it with some popular detection schemes and ten of the most common antiviruses. Section 4 introduces the related work and smooth comparison between the currently proposed Android malware detection schemes and *RoughDroid*. This is followed by the conclusion in Section 5.

2. RoughDroid

In this paper, we present *RoughDroid*, a lightweight technique for discovering Android malware that presumes discovery patterns immediately. In addition, it allows recognizing malware straight on the smartphone. *RoughDroid* performs a broad floppy analysis, gathering as numerous features from an *application's code* as well as *manifest* as feasible. These features are organized in groups of strings (for instance, features API calls along with network speeches) and embedded within a combined vector space. As an example, an application sending out premium SMS messages is cast to a particular area in the vector room connected with the equivalent consent, intents, and also API calls. This geometric depiction allows *RoughDroid* to recognize mixes as well as patterns of features indicative for malware automatically, by utilizing machine learning techniques.

To this end, our technique utilizes a broad floppy analysis that extracts feature sets from various resources and examines these features in a meaningful vector space. This procedure is shown in Figure 1 and also described as follows:

(I) **Floppy Analysis:** *RoughDroid* floppily inspects a given Android application and various feature collections from the application's *manifest* and also disassembled *Dex code*. *RoughDroid* inspects the application's *manifest* and disassembled *Dex code* of a given Android Application in *Parallel Sweep* to reduce the time of analysis.

(II) **Constructing Vector Space:** The extracted feature sets are after that mapped to a joint vector space, where patterns and also mixes of the features could be evaluated geometrically.

(III) **Rough-Based Detection:** The embedding of the feature sets allows us to recognize malware utilizing effective strategies of machine learning (Rough Set algorithm).

2.1. Floppy Analysis. As the primary step, *RoughDroid* carries out a lightweight floppy analysis of an offered Android application. The floppy extraction of features should run in a constricted environment and in full prompt way. The customer might avoid the recurring procedure, if the evaluation takes so long time. Appropriately, it becomes vital to pick features that can be extracted effectively. We therefore focus on the *manifest* in addition to the disassembled *Dex code* of this application, which could be obtained with a parallel sweep within the application's content. To enable an extensible as well as common evaluation, we represent all extracted

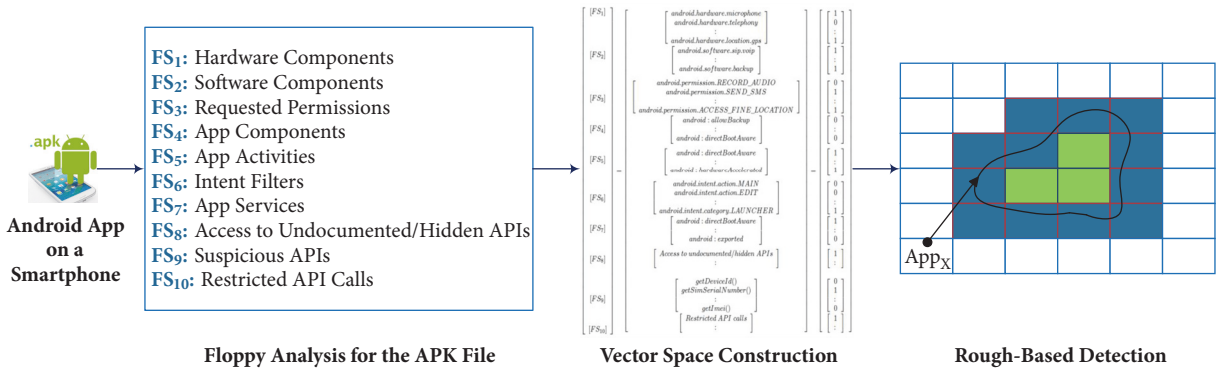

Figure 1: General overview for the proposed *RoughDroid* system model.

features as sets of strings, such as authorizations, intents, and also API calls. Specifically, we extract the adhering ten sets of features.

2.1.1. Manifest Feature Sets. It is an effective file in the Android system that defines the performance and also demands of an application to Android. AndroidManifest.xml could be located at the root of the project and has numerous various feature sets.

A simple XML *manifest* file generated for an Android application tested under *RoughDroid* is shown in Figure 2. The presented XML file declares seven different features sets ($FS_1, FS_2, FS_3, FS_4, FS_5, FS_6$, and FS_7) as follows:

FS_1 :***Hardware Components:*** It has the requested hardware features by an application. The figure indicates three requested hardware features (microphone, telephony, and location.gps). An application that has access to GPS and network modules is, for instance, able to collect fine location information and send it to an attacker over the network.

FS_2 :***Software Components:*** It indicates that the application utilizes or requires software features. The figure declares sip.voip that allows the application to use Session Initiation Protocol (SIP) services and do VOIP calls.

FS_3 :***Requested Permissions:*** It is very important for Android security mechanisms. The figure mentions three dangerous permissions (RECORD_AUDIO, SEND_SMS, and ACCESS_FINE_LOCATION) that are granted to the application during the application's setup time by the user.

FS_4 :***App Components:*** It is a set of Boolean expressions that grant some services to the application, such as allowBackup and directBootAware.

FS_5 :***App Activities:*** It allows the application to execute a specific activity, such as directBootAware and hardwareAccelerated.

FS_6 :***Intent Filters:*** It specifies the types of intents that an activity, service, or broadcast receiver can respond to, such as action.MAIN and action.EDIT.

FS_7 :***App Services:*** It represents a service as one of the application's components, such as directBootAware and exported.

The information saved in AndroidManifest.xml file could be effectively obtained on the device by making use of the *Android Asset Packaging Tool* that allows us to extract out the previously mentioned sets of features.

2.1.2. Disassembled Code Feature Sets. We implement a lightweight disassembler, which takes as input the Dalvik Executable (*Dex*) and provides *RoughDroid* with the complete information about API calls and the data utilized in the application. The *Dex* file contains a set of class definitions and their associated adjunct data. Table 1 introduces a simple example for the *Dex* file that is enhanced bytecode for the Dalvik virtual machine. Every Android application has a unique classes.dex file, which references any type of approaches or courses utilized within an application. Basically, any type of task, things, or piece utilized within the codebase will certainly be changed right into bytes within a *Dex* file that could be run as an Android application.

We are mainly interested in the API calls and method calls, because they can be easily extracted from the *Dex* file of an application, as follows:

FS_8 :***Access to Undocumented/Hidden APIs:*** Applications could be limited from accessing APIs that are undocumented in the Android Software Development Kit (SDK). *RoughDroid* looks for the incident of these demands in the *Dex* file, in order to get a further understanding of the behavior of an application.

FS_9 :***Suspicious APIs:*** Requesting some delicate information or sources of the Android phone might result in destructive behavior. We are laying more importance to a set of such suspicious APIs:

(i) **Sensitive data (*IMEI* and *USIMnumbeleakage*) APIs**, where the Android requests are such as *getDeviceId()*, *getSimSerialNumber()*, and *getImei()*;

(ii) **Network communication APIs**, such as *setWifiEnabled()* and *execHttpRequest()*;

Table 1: A simple Dex file has code which is eventually performed by the Android Runtime.

(a)

No.	Dex File Magic		Checksum	SHA1-Hash Signature					File Size	Header Size
1	6465780A	30333800	7A44CBBB	FB4AE841	0286C06A	8DF19000	3C5DE024	D07326A2	E0010000	70000000
2	00000000	02000000	9C000000	01000000	AC000000	14010000	CC000000	E4000000	EC000000	07010000
3	00000000	01000000	01000000	00000000	00000000	FFFFFFFF	00000000	57010000	00000000	01000100
4	706C6963	6174696F	6E3B0023	4C636F6D	2F627567	736E6167	2F646578	6578616D	706C652F	42756773

(b)

No.	Endian Constant	Link Size & Offset	Map Section	String ID	Type ID	Photo ID	Field ID	Method ID	Class Def	Data Size
1	78563412	00000000	00000000	64010000	05000000	70000000	03000000	84000000	01000000	90000000
2	2C010000	2F010000	01000000	02000000	03000000	03000000	02000000	00000000	00000000	00000000
3	01000000	00000000	04000000	70100000	00000E00	063C696E	69743E00	194C616E	64726F69	642F6170
4	6E616741	70703B00	01560026	7E7E4438	7B226D69	6E2D6170	69223A32	362C2276	65727369	6F6E223A

(iii) **Location leakage APIs**, such as *getLastKnownLocation(), getLatitude(), getLongitude(),* and *requestLocationUpdates();*

(iv) **Sending and receiving SMS/MMS messages APIs**, such as *sendTextMessage(), SendBroadcast(),* and *sendDataMessage()*.

FS_{10} :***Restricted API calls:*** The Android authorization system limits accessibility to a collection of crucial API calls. Our approach looks for the event of these calls that represent a apart of the *Dex code*, in order to get a much deeper understanding of an App's capability.

$$V(App) = \begin{bmatrix} [FS_1] \\ [FS_2] \\ [FS_3] \\ [FS_4] \\ [FS_5] \\ [FS_6] \\ [FS_7] \\ [FS_8] \\ [FS_9] \\ [FS_{10}] \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} android.hardware.microphone \\ android.hardware.telephony \\ \vdots \\ android.hardware.location.gps \end{bmatrix} \\ \begin{bmatrix} android.software.sip.voip \\ \vdots \\ android.software.backup \end{bmatrix} \\ \begin{bmatrix} android.permission.RECORD\_AUDIO \\ android.permission.SEND\_SMS \\ \vdots \\ android.permission.ACCESS\_FINE\_LOCATION \end{bmatrix} \\ \begin{bmatrix} android : allowBackup \\ \vdots \\ android : directBootAware \end{bmatrix} \\ \begin{bmatrix} android : directBootAware \\ \vdots \\ android : hardwareAccelerated \end{bmatrix} \\ \begin{bmatrix} android.intent.action.MAIN \\ android.intent.action.EDIT \\ \vdots \\ android.intent.category.LAUNCHER \end{bmatrix} \\ \begin{bmatrix} android : directBootAware \\ \vdots \\ android : exported \end{bmatrix} \\ \begin{bmatrix} Access\ to\ undocumented/hidden\ APIs \\ \vdots \end{bmatrix} \\ \begin{bmatrix} getDeviceId() \\ getSimSerialNumber() \\ \vdots \\ getImei() \end{bmatrix} \\ \begin{bmatrix} Restricted\ API\ calls \\ \vdots \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 \\ \vdots \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 \\ \vdots \end{bmatrix} \end{bmatrix} \tag{1}$$

2.2. Vector Space Construction. A harmful task is normally shown in particular patterns as well as mixes of the extracted features. As an example, a malware application sending the fine location of a smartphone might have the permission *android.permission.ACCESS_FINE_LOCATION* in FS_3 and the hardware feature *android.hardware.location.gps* in FS_1.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.IntentApp">
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <!-- FS1: Hardware Components -->
    <uses-feature android:name="android.hardware.microphone" />
    <uses-feature android:name="android.hardware.telephony" />
    <uses-feature android:name="android.hardware.location.gps" />

    <!-- FS2: Software Components -->
    <uses-feature android:name="android.software.sip.voip" />
    <uses-feature android:name="android.software.backup" />

    <!-- FS3: Requested Permissions -->
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <!-- FS4: App Components -->
    <application
        android:allowBackup="true"
        android:directBootAware="true">

        <!-- FS5: App Activities -->
        <activity
            android:directBootAware="true"
            android:hardwareAccelerated="true"/>

        <!-- FS6: Intent Filters -->
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <action android:name="android.intent.action.EDIT" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

        <!-- FS7: App Services -->
        <service android:name=".IntentService"
         android:directBootAware="true"
         android:exported="true">
        </service>
    </application>
</manifest>
```

Figure 2: A simple XML *manifest* example (AndroidManifest.xml) that declares seven feature sets for an Android application.

Preferably, we would like to create Boolean expressions that catch this reliance in between features as well as returning true if a malware is found.

We will need to place the extracted feature collections from an Android application $(FS_1, FS_2, \ldots, FS_{10})$ in a vector. In our experiments the vector space $(V(App))$ contains approximately 550,000 different extracted features. If the application (App) contains the feature (f), the vector space element for that feature is mapped to 1 $(V(App, f) = 1)$; otherwise, it is mapped to 0 $(V(App, f) = 0)$. A simple structure as an example of the vector space is shown in (1). Regardless of the measurement of the vector space, it is hence enough to just save the extracted features from an application for sparsely standing for the vector $V(App)$ by using either hash tables [10] or Bloom filters [11].

2.3. Rough-Based Detection. Rough Set based data analysis [12–14] starts after constructing the vector space (feature table), as depicted in Section 2.2. Each row represents a specific feature obtained from a certain feature set according to a specific Android application in our scheme. The Rough system has multiple entities and stages.

(i) **Feature Table:** It is a pair $FS = (Apps, F)$ where $Apps$ is a nonempty finite set of Android applications called the universe and F is a nonempty finite set of features such that $f : Apps \longrightarrow Vf$ for every $f \in F$. The set Vf is called the value set of f, and elements of $Apps$ are called Android applications.

(ii) **Decisions:** It is the feature table in the form $FS = (Apps, F \cup \{app_f\})$, where App_f (not a feature in F) is the decision feature. The features of F are called conditional features or simply conditions.

(iii) **Approximations:** Let $App_X \subseteq Apps$:

(a) **Lower Approximation:** It consists of all Android applications, which definitely belong to $\underline{R}(App_X) = \{App \in Apps \mid [App]R \subseteq App_X\}$.

(b) **Upper Approximation:** It contains all Android applications, which possibly belong to $\overline{R}(App_X) = \{App \in Apps \mid [App]R \cap App_X \neq \phi\}$.

(c) **Boundary Region:** The difference between the upper and lower approximations constitutes the boundary region of the Rough Set algorithm. Boundary positive and negative regions [15] are described as below.

$$\begin{aligned} BND_R(App_X) &= \left|\overline{R}(App_X X) - \underline{R}(App_X)\right| \\ POS_R(App_X) &= \underline{R}(App_X) \\ NEG_R(App_X) &= U - \overline{R}(App_X) \end{aligned} \tag{2}$$

An Android application of the negative region $NEG_R(App_X)$ does not belong to App_X, an application of the positive region $POS_R(App_X)$ belongs to App_X, and only one application of the boundary region $BND_R(App_X)$ belongs to App_X. Those approximation sets and regions are shown in Figure 3.

(d) **Approximation Accuracy:** The roughness precision of any subset $App_X \subseteq Apps$ with regard to $R \subseteq F$, represented as $\alpha_R(App_X)$, is quantified by $\alpha_R(App_X) = |\underline{R}(App_X)/\overline{R}(App_X)|$, where $|App_X|$ represents cardinality of App_X. For an empty set ϕ, we define $\alpha_R(\phi) = 1$. It is worth noting that $0 \leq \alpha_R(App_X) \leq 1$. If $\alpha_R(App_X) = 1$, the set App_X is crisp with respect to R. If $\alpha_R(App_X) < 1$, App_X is tough with reference to R.

3. Evaluation

After providing *RoughDroid* thoroughly, we currently continue to an empirical assessment of its efficiency. In order to do so, we first describe the used dataset and then run some experiments to evaluate the detection performance.

3.1. Considered Data Sets. Our experiments are executed based on a dataset of genuine Android applications and also actual malware. We are utilizing Drebin dataset [1], it comprises 131,611 software samples collected from other tools, including Google Play Store, both Chinese and Russian Markets, and also Android sites. Additionally, it Includes 5,560 malware applications, from 179 distinct malware families, for example, *FakeInstaller, DroidKungFu, Plankton, Opfake, GingerMaster, BaseBridge, Iconosys, Kmin, FakeDoc, Geinimi, Adrd, DroidDream, Linux/Lotoor, GoldDream, MobileTx, FakeRun, SendPay, Gappusin, Imlog,* and *SMSreg*.

Table 2: Statistical measures for *RoughDroid* and four other schemes.

Scheme	Area	Std. Error	Asymptotic Prob.	95% LCL	95% UCL
RCP [16]	0.52123	0.08965	5.63972E-5	0.43983	0.62718
Peng et al. [17]	0.08182	0.85211	4.941212E-5	0.51643	0.73156
SigPID	0.91364	0.07131	4.72643E-4	0.57846	0.87653
Drebin [1]	0.93521	0.06344	3.66257E-4	0.66972	0.91842
RoughDroid	0.95633	0.04577	2.48984E-6	0.79892	0.97833

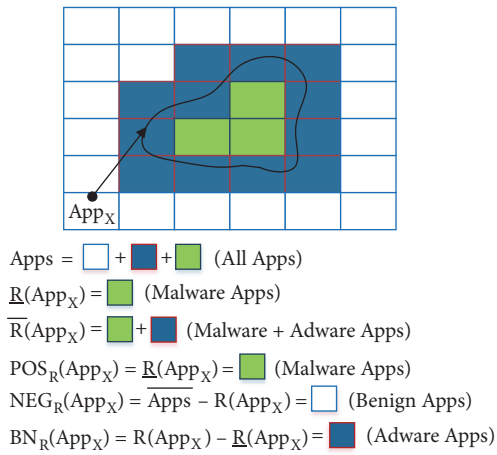

Figure 3: The approximations and regions of an Android applications' set App_X using Rough Set algorithm.

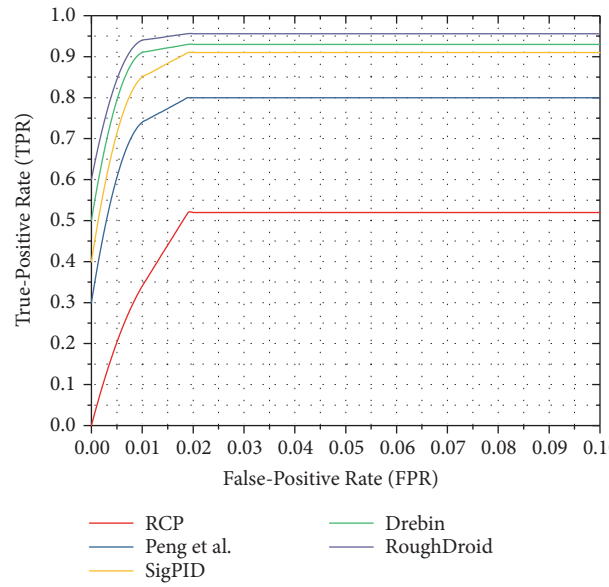

Figure 4: *RoughDroid* performance analysis ROC curve comparison with three approaches.

In addition, we have also considered 158 Android applications introducing three new malware families (*Grabos*, *TrojanDropper.Agent.BKY*, and *AsiaHitGroup*) that invade Google Play Store at 2017. It should be mentioned that the adware applications are considered in our dataset.

3.2. Performance Analysis. Our *RoughDroid* does not need initial training in advance, which is one of its basic advantages. *RoughDroid* analyzes each application in a broad floppy way and grab a great collection of features categorized in ten feature sets ($FS_1, FS_2, \ldots, FS_{10}$). It should be mentioned that the results are obtained from the average of 25 trials using the same environmental conditions. We introduce our analysis based on comparing *RoughDroid's* results with the results obtained from related approaches and ten popular antivirus scanners, finally employing *RoughDroid* to find the detection rate for the most popular malware families.

3.2.1. RoughDroid and Related Approaches. We initially contrast the efficiency of *RoughDroid* versus associated static methods for the discovery of Android malware. Specifically, we think about Drebin [1], RCP [16], Peng et al. [17], and SigPID [18]. The outcomes of these experiments are displayed in Figure 4 as ROC curve. *RoughDroid* outperforms the four previously mentioned approaches by detecting 95.6% of the malware applications at a false-positive rate (FPR) equal to 1%.

Also, according to the statistical measures introduced in Table 2, the *Asymptotic Probability* of the five schemes is much smaller than 0.05; thus we can conclude that all schemes are effective. In addition, the area under the curve of *RoughDroid* is 0.95633, which is closer to 1.0; hence, *RoughDroid* is the best scheme in successfully detecting the malware of an Android application. The excellent efficiency of *RoughDroid* arises from the various feature sets that are used to design the malicious activity of an application.

3.2.2. RoughDroid and Popular AV Scanners. *RoughDroid* reveals a much better efficiency compared to related approaches ([1, 16–18]). We likewise contrast it with ten picked antivirus scanners on the considered dataset. It should be mentioned that we consider *FPR* = 1%, which we assume to be adequately low enough for practical scenarios.

Experimental results are displayed in Table 3. The best antivirus detects over 90% of malware applications. Our *RoughDroid* also provides best performance with detection rate of 95.6%.

3.2.3. Detecting Malware Families. When evaluating the detection efficiency of an approach, the equilibrium of malware family members in the dataset is very important. If the number of applications of a particular malware family members is little great compared to various other families, the detection result might mostly depend on these families.

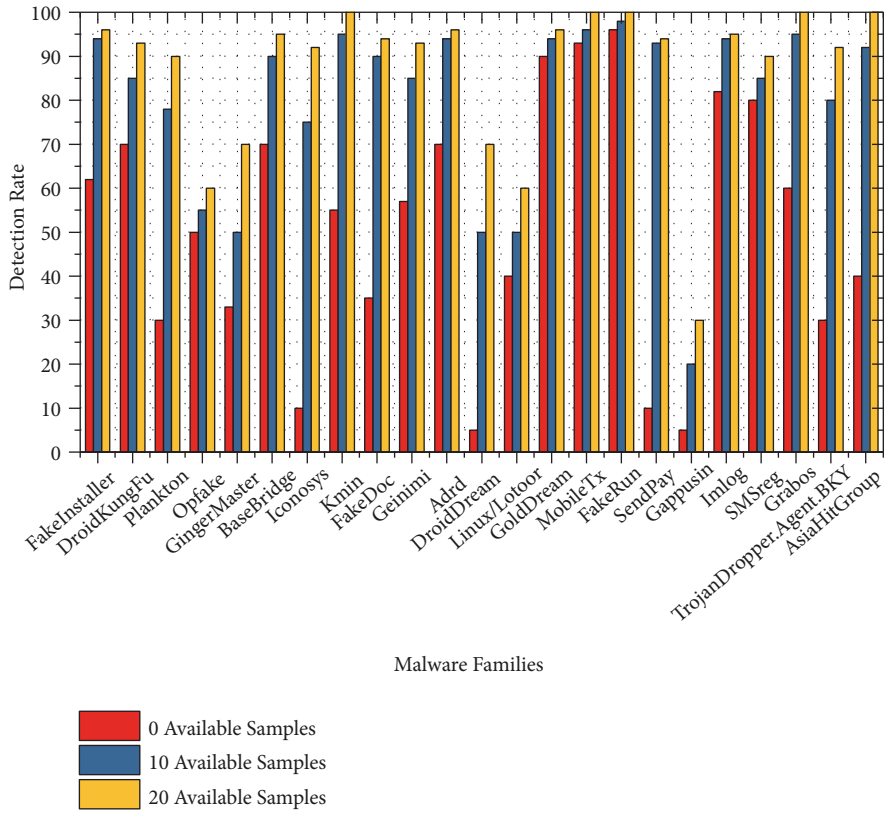

Figure 5: *RoughDroid* detection rate for 23 malware families based on three different number of applications for each family.

Table 3: Detection rates of *RoughDroid* in comparison with Drebin, SigPID, and ten anti-virus scanners.

	RoughDroid	Drebin [1]	SigPID [18]	Anti-V1	Anti-V2	Anti-V3	Anti-V4	Anti-V5	Anti-V6	Anti-V7	Anti-V8	Anti-V9	Anti-V10
Detection Rate	95.60%	93.90%	91.22%	96.41%	93.71%	84.66%	84.54%	78.38%	64.16%	48.50%	48.34%	9.84%	3.99%

An unreal solution to this problem is to make use of the same number of applications for each malware family.

We are laying more stress on 20 (*FakeInstaller, DroidKungFu, Plankton, Opfake, GingerMaster, BaseBridge, Iconosys, Kmin, FakeDoc, Geinimi, Adrd, DroidDream, Linux/Lotoor, GoldDream, MobileTx, FakeRun, SendPay, Gappusin, Imlog,* and *SMSreg*) top common malware families, plus the new three (*Grabos, TrojanDropper.Agent.BKY*, and *AsiaHitGroup*) malware families that invade Google Play Store at 2017.

We perform three more experiments, by restricting the variety of applications for a certain family in the test set. In the first experiment, we offer no applications of the family. In the second experiment, we place 10 arbitrarily picked applications of the family back right into the test set. Finally, in the third experiment, we use 20 arbitrarily picked applications of the family back right into the test set. The consequences of these three experiments are shown in Figure 5. *RoughDroid* can reliably detect all households with a typical precision of 95.6% at $FPR = 1\%$. The figure also shows that five (*Kmin, MobileTx, FakeRun, Grabos,* and *AsiaHitGroup*) families are perfectly detected.

4. Related Work and Discussion

To the best of our knowledge, Android malware detection and classification have a wide research area in the last decade. It has three basic categories, based on the detection technique, that is, static analysis, dynamic analysis, and machine learning analysis. Several methods have been proposed for statically analyzing an Android application, such as [8, 9, 18, 19]. Also, there are some contributions based on dynamic analysis, such as [5–7, 20, 21]. Regarding realizing the data placement considering both the energy consumption in private cloud and the cost for renting the public cloud services, the authors in [22] have proposed a cost- and energy-aware data placement method, for privacy-aware applications over big data in hybrid cloud.

Furthermore, the detection techniques [16, 17, 23, 24] are based on machine learning. The authors in [25] propose a new bio-key production algorithm called FVHS, which unites the benefits of the biometrics authentication and user-key authentication. Also, in [26] the authors suggest a new scheme named FREDP (File Remotely keyed Encryption and Data Protection). This strategy entails interaction between

one of the clouds that are personal and a terminal. The authors in [27] propose a new identity-based blind signature scheme based on number theorem research unit lattice.

The authors in [28, 29] are proposing a new access control for cloud infrastructure as a service. Also, a trust based access control model is proposed in [30]. In addition, cryptographic access control scheme is introduced in [31]. Also, the authors in [32] propose a new space metric optimization pushed deep-learning frame for age-invariant facial recognition. A complete review for Blockchain and intrusion detection is available in [33]. Reference [34] introduced JFCGuard for detecting juice filming charging attack and [35] enhanced network capacity. A privacy-preserving scheme based on location is introduced in [36].

Due to the sparsity of big rating data in E-commerce, both similar friends and similar product items may be absent from the user-product purchase network, which leads to a big challenge to the recommendation of appropriate product items to the target user. The authors in [37] propose a structural balance theory-based recommendation scheme. Also, protecting users' privacy is challenging when IBM releases its own data to Amazon. In addition, the recommendation efficiency and scalability are often low when the user-service quality data of Amazon and IBM are updated frequently. Thus, the authors in [38] have proposed a privacy-preserving and scalable service recommendation approach based on distributed locality-sensitive hashing.

Based on deep learning, [39] proposed a novel finger vein recognition algorithm. For social networks, [40] introduced a measure for social influence. An early detection scheme for IP traffic is introduced in [41]. A new instant encrypted transmission is proposed in [42]. Based on trusted routing, a sensitive analysis of attack-pattern is proposed in [43]. Finally, [44] presents a new scheme M-SSE that achieves both forward and backward security based on a multicloud technique.

5. Conclusion

This paper introduced *RoughDroid* that is a new broad floppy analysis malware detector on smart Android phones during the installation time by introducing robust feature extraction framework. *RoughDroid* performs a broad floppy analysis, gathering numerous features from an application's *Dex code* as well as *manifest* file. It is based on ten feature sets ($FS_1, FS_2, \ldots, FS_{10}$). It then uses the Rough Set algorithm to check the behavior of an Android application. The experimental results showed that *RoughDroid* is detecting 95.6% of the malware applications at a $FPR = 1\%$, which means that *RoughDroid* outperforms the well-known detection approaches (Drebin [1], RCP [16], Peng et al. [17], and SigPID [18]). Also, *RoughDroid* is compared with the ten most popular antivirus scanners and proved efficiency in practical scenarios. Finally, *RoughDroid* is able to perfectly detect five (*Kmin, MobileTx, FakeRun, Grabos,* and *AsiaHitGroup*) malware families.

Data Availability

The data used to support the findings of this study are available from the authors upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by Guangzhou Scholars Project for Universities of Guangzhou (No. 1201561613). Also, this work was supported by the Egyptian Ministry of Higher Education, the Arab Republic of Egypt.

References

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: effective and explainable detection of android malware in your pocket," in *Proceedings of the NDSS Symposium 2014*, February 2014.

[2] Android Developers Blog, *How We Fought Bad Apps and Malicious Developers in 2017*, January 2018.

[3] SC Media US, *Three M Android Malware Families Invade Google Play Store*, November 2017.

[4] Amazon's App Store Compromises Android Security, *It's dangerous to go alone outside Google's protective walled garden, but it's the price you pay for free software*, 2017.

[5] W. Enck, P. Gilbert, B.-G. Chun et al., "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, pp. 393–407, February 2010.

[6] 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 *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS '12)*, 2012.

[7] L.-K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing os and dalvik semantic views for dynamic android malware analysis," in *Proceedings of the USENIX Security Symposium*, 2012.

[8] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 627–638, ACM, New York, NY, USA, 2011.

[9] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 281–294, ACM, New York, NY, USA, June 2012.

[10] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1989.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] S. Rissino and G. Lambert-Torres, "Rough set theory –fundamental concepts, principals, data extraction, and applications," in *Data Mining and Knowledge Discovery in Real Life Applications*, pp. 35–59, INTECH, January 2009.

[13] Z. Pawlak, J. Grzymala-Busse, R. Slowinski, and W. Ziarko, "Rough sets," *Communications of the ACM*, vol. 38, no. 11, pp. 88–95, 1995.

[14] L. Polkowski, S. Tsumoto, and T. Y. Lin, Eds., *Rough Set Methods and Applications: New Developments in Knowledge Discovery*

in Information Systems, Physica-Verlag GmbH, Heidelberg, Germany, 2000.

[15] S. Hirano and S. Tsumoto, "Rough representation of a region of interest in medical images," *International Journal of Approximate Reasoning*, vol. 40, no. 1-2, pp. 23–34, 2005.

[16] 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 (SACMAT '12)*, pp. 13–22, ACM, New York, NY, USA, June 2012.

[17] H. Peng, C. Gates, B. Sarma et al., "Using probabilistic generative models for ranking risks of Android apps," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '12)*, pp. 241–252, ACM, New York, NY, USA, October 2012.

[18] 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*, vol. PP, no. 99, pp. 1-1, 2018.

[19] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 235–245, ACM, New York, NY, USA, November 2009.

[20] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proceedings of the European Workshop on System Security (EUROSEC '13)*, April 2013.

[21] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: automatic security analysis of smartphone applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY '13)*, pp. 209–220, ACM, New York, NY, USA, February 2013.

[22] X. Xu, X. Zhao, F. Ruan et al., "Data placement for privacy-aware applications over big data in hybrid clouds," *Security and Communication Networks*, vol. 2017, 2017.

[23] D. Barrera, H. G. Kayacik, P. C. Van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)*, pp. 73–84, ACM, New York, NY, USA, October 2010.

[24] 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 *Proceedings of the Seventh Asia Joint Conference on Information Security (Asia JCIS '12)*, pp. 62–69, Tokyo, Japan, August 2012.

[25] Z. Wu, L. Tian, P. Li, T. Wu, M. Jiang, and C. Wu, "Generating stable biometric keys for flexible cloud computing authentication using finger vein," *Information Sciences*, vol. 433-434, pp. 431–447, 2018.

[26] L. Yang, Z. Han, Z. Huang, and J. Ma, "A remotely keyed file encryption scheme under mobile cloud computing," *Journal of Network and Computer Applications*, vol. 106, pp. 90–99, 2018.

[27] H. Zhu, Y.-a. Tan, L. Zhu, X. Wang, Q. Zhang, and Y. Li, "An identity-based anti-quantum privacy-preserving blind authentication in wireless sensor networks," *Sensors*, vol. 18, no. 5, p. 1663, 2018.

[28] K. Riad, Z. Yan, H. Hu, and G.-J. Ahn, "AR-ABAC: A New Attribute Based Access Control Model Supporting Attribute-Rules for Cloud Computing," in *Proceedings of the 1st IEEE International Conference on Collaboration and Internet Computing, CIC 2015*, pp. 28–35, China, October 2015.

[29] K. Riad, "Blacklisting and forgiving coarse-grained access control for cloud computing," *International Journal of Security and Its Applications*, vol. 10, no. 11, pp. 187–200, 2016.

[30] K. Riad and Z. Yan, "Multi-factor synthesis decision-making for trust-based access control on cloud," *International Journal of Cooperative Information Systems*, vol. 26, no. 4, pp. 1–33, 2017.

[31] K. Riad, "Revocation basis and proofs access control for cloud storage multi-authority systems," in *Proceedings of the 3rd International Conference on Artificial Intelligence and Pattern Recognition, AIPR 2016*, pp. 118–127, Poland, September 2016.

[32] Y. Li, G. Wang, L. Nie, Q. Wang, and W. Tan, "Distance metric optimization driven convolutional neural network for age invariant face recognition," *Pattern Recognition*, vol. 75, pp. 51–62, 2018.

[33] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: A review," *IEEE Access*, vol. 6, pp. 10179–10188, 2018.

[34] W. Meng, L. Jiang, Y. Wang, J. Li, J. Zhang, and Y. Xiang, "Jfcguard: Detecting juice filming charging attack via processor usage analysis on smartphones," *Computers & Security*, 2017.

[35] J. Cai, Y. Wang, Y. Liu, J.-Z. Luo, W. Wei, and X. Xu, "Enhancing network capacity by weakening community structure in scale-free network," *Future Generation Computer Systems*, 2017.

[36] T. Peng, Q. Liu, D. Meng, and G. Wang, "Collaborative trajectory privacy preserving scheme in location-based services," *Information Sciences*, vol. 387, pp. 165–179, 2017.

[37] L. Qi, X. Xu, X. Zhang et al., "Structural balance theory-based e-commerce recommendation over big rating data," *IEEE Transactions on Big Data*, p. 1, 2017.

[38] L. Qi, X. Zhang, W. Dou, and Q. Ni, "A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2616–2624, 2017.

[39] Y. Liu, J. Ling, Z. Liu, J. Shen, and C. Gao, "Finger vein secure biometric template generation based on deep learning," *Soft Computing*, vol. 22, no. 7, pp. 2257–2265, 2018.

[40] S. Peng, A. Yang, L. Cao, S. Yu, and D. Xie, "Social influence modeling using information theory in mobile social networks," *Information Sciences*, vol. 379, pp. 146–159, 2017.

[41] Z. Chen, L. Peng, C. Gao, B. Yang, Y. Chen, and J. Li, "Flexible neural trees based early stage identification for IP traffic," *Soft Computing*, vol. 21, no. 8, pp. 2035–2046, 2017.

[42] C. Wang, J. Shen, Q. Liu, Y. Ren, and T. Li, "A novel security scheme based on instant encrypted transmission for internet of things," *Security and Communication Networks*, vol. 2018, Article ID 3680851, 7 pages, 2018.

[43] R. H. Jhaveri, N. M. Patel, Y. Zhong, and A. K. Sangaiah, "Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile Ad-Hoc networks in industrial IoT," *IEEE Access*, vol. 6, pp. 20085–20103, 2018.

[44] C. Gao, S. Lv, Y. Wei, Z. Wang, Z. Liu, and X. Cheng, "M-SSE: An effective searchable symmetric encryption with enhanced security for mobile devices," *IEEE Access*, pp. 1-1, 2018.