Advance Access publication on 14 November 2019
doi: 10.1093/comjnl/bxz109

New Security Risk Value Estimate Method for Android Applications

Latifa Er-rajy*, My Ahmed El Kiram and Mohamed El Ghazouani

Department of Computer Sciences, University Cadi Ayyad, Marrakesh, Morocco
**Corresponding author: errajy.latifa@gmail.com*

Nowadays, mobile applications are the devices' core, so their security is essential for the platform on which are installed. Therefore, to make this security strong, Android implements permission system and sandboxing mechanism to reduce the attacks contingency. Also, antivirus software is used to detect the already known malware based on their signature, but unfortunately, this kind of program cannot detect cloned or repackaged malware. Thus, using authorizations to estimate the security vulnerability will surely be very useful for avoiding applications that are more susceptible to be dangerous according to the risk values obtained. Consequently, this will provide systematic support that will make it easier for users to make appropriate decisions and therefore greatly improve the Android devices' security. In this article, additionally to the risk measures that have been already proposed in previous studies based on functionalities such as permissions and function calls, we add a set of mathematical metrics describing the case of susceptible and innocent applications already known. Our risk assessment exploits applications already known as malware and safe samples without any threat.

Keywords: components; risk; permissions; Android; method

Received 22 September 2018; Revised 25 June 2019; Editorial Decision 14 August 2019
Handling editor: Mark Ryan

1. INTRODUCTION

In Android system, the majority of functionalities and features can be installed by end users as applications (apps) developed by anonymous developers or downloaded from PlayStore and Android markets. These apps provide access to the different types of services available on the device and access to end-user sensitive, private and personal data, such as contact lists, geo-location, messages and social networks [1]. These types of accessibility make the privacy and security assurance more difficult. So, to combat the attacks that target Android, the permissions and sandboxing mechanisms are applied [2]. This means that if an application has intention to perform any malicious activities such as making phone calls, stealing user-sensitive data or simply sending premium messages, the attacker must deceive users in order to install a malicious application, because other previously existing intrusion ways are almost eliminated in Android recent versions [3]. At the installation time, an application must request permission to access system resources (such as location, Internet, cellular networks, etc.), and the user either accepts all the requested permissions or cancels the installation, since it is not possible to accept or deny access privileges selectively [4]. The permissions granted to the application are not changeable as they are revoked from it when the user starts its removal process. Even if this security mechanism seems to be very simple and easy, but it creates many challenges because many users simply accept these authorizations without considering their implications on their personal data [5] since they do not usually have much time to study and think about the side effects of these permissions, they are only interested in completing the application installing process quickly to use it. On the other hand, technical skills of an ordinary user are not enough to understand well the requested permissions impacts [1].

In addition, an innocent application and a malware may request similar permissions, which will obviously complicate the task of making the correct decision by the end users. For example, if an application has granted certain critical permissions such as Internet permission, this application can easily control communication with remote servers. And, if it has access to the camera as well, it can send user personal pictures to any server on the Internet. For all these reasons, the security offered by the Android model is weak and vulnerable. It is unable to preserve the personal information from disclosure or to prevent fraudulent use of monetary resources. In fact,

spyware, Trojan horses or adware can easily induce users by pretending to be a useful application for him and then hack banking applications on his device to steal his credit [6, 7].

To ensure an effective security, we propose in this article a new method to calculate the risk score that an application may cause to the user safety and send him a warning if the risk value obtained exceeds a predetermined value. This will allow the user to compare similar applications based on their risk scores. Android markets are in urgent need of an effective measure to calculate the risk rate that applications can present, in order to be able to identify and detect those that seem to be suspicious among the large number of applications newly submitted by developers. However, it will take a lot of time to carry out a detailed and thorough analysis for each new application in aiming to detect malware. Indeed, our studies [8–11] based on detailed evaluations reveal that Android risk calculation models use measures with vulnerabilities and unacceptable performances, which will undoubtedly force end users to lose confidence in Android applications. In fact, Android system never calculates high-risk values for an unknown malicious application. Actually, it has a low level of risk for this kind of application.

In this article, we present a new measurement that calculates the risk rate of Android applications much higher than those already proposed by other researchers. In fact, the existing risk measures are weak because they are mainly based on a single criterion. Nevertheless, our solution is based on a large set of well-known malicious applications. We proved our proposition effectiveness by performing many in-depth experiments on many actual applications samples selected from Android markets.

This article organization is as follows: In Section II, we present our related work. In Section III, we discuss Android security issue behind our solution. We present our proposed measure for calculating the risk rate in Section IV. Then, in Section V, we present the results obtained after our measure experimental evaluations applied to normal and malicious Android applications and their comparisons with those previously obtained by measures already proposed. Finally, in Section VI, we conclude our article.

2. RELATED WORK

If we look at Android system security, we find that there is a small number of researches that is interested in this topic. As we explained before, Android users have only two choices in front of them, whether they accept the permissions requested by the application they want to install to start its installation or deny the requested permissions and completely cancel the application installation process. The results obtained by these researches show that most users accept these permissions without paying attention to their impact on their personal data, which can be very dangerous if they fall in the hacker's hands. Of course, researchers have tried to find effective solutions to solve this problem. So, they come out by some propositions to improve the already existing Android security architecture [12–14]. For example, Chin *et al.* [11] proposed to change Android permissions categorizations by highlighting the security danger instead of focusing on the requested permissions. They therefore proposed a new method to accept the permissions required by the application to be installed.

Correspondingly, Kelley *et al.* [14] proposed to control access to high-level critical data that affects the user's privacy. Consequently, they proposed to replace the permission names with location information and contact lists in the first install page. Gates *et al.* [15] as well advised adapting the risk rate visualization to reduce the memory space size required for displaying permissions to the user and help him to make quick, correct and efficient decisions about the applications he wants to install on his device. In order to calculate these scores, Gates *et al.* [15] had based on varying set of permissions requested by each Android application. The majority of users prefer to have a brief summary of threat results or security display as graphical indicators since they are more convenient than textual information.

Similarly, Peng *et al.* [16] proposed using crawl models and statistical metrics to compute security risk scores basing on permissions requested by Android apps. The approach they proposed is to rank apps in a market similar to PlayStore based on their risk rate, which has encouraged users to select safer app among applications available with the same functionality with different levels of danger. Afterward, Gates *et al.* [9] had improved their previous solution by accurately describing many statistical scores risks using Android permissions to generate probabilities for Android apps. They also exploited all measurements obtained based on critical permissions, such as those allowing access to sensitive software and hardware resources and usually used by malware. Generally, Android malicious applications request critical permissions to exploit the API functions to perform malicious activity against the user. Correspondingly, Gates *et al.* [9] used the permissions requested by benign applications to calculate risk scores. However, both solutions have increased the facts of some critical permissions that influence the obtained risk rate values to improve their proposition performance. Indeed, Gates *et al.* manually selected a set of critical permission consisting of nine permissions that can be used by malware to perform dangerous actions. Similarly, Sarma *et al.* [10] proposed to use critical permissions requested by Android applications, but unlike Gates *et al.* [9], they choose to rely on the permissions that are rarely required by normal applications for the purpose of recognizing dangerous applications. As well, Grace *et al.* [8] implemented an automated system they called "RiskRanker" to check if an application has dangerous behavior or not. To develop this system, they exploited the malicious behaviors commonly used in malware belonging to the same time interval. RiskRanker was used as an initial step before putting the application on

the official Android market "PlayStore"'; RiskRanker has the ability to detect malware by producing a list to classify the suspicious application priority based on their risk value gotten.

Besides, Enck *et al.* [17] implemented a Java static analysis tool with an aim to decompile applications; due to this tool, they were able to evaluate a large set of applications and study their API usage. However, they only focused on studying applications that use an API call with an unimportant number of permissions. In another study, Felt *et al.* [12] manually sort a small collection of Android applications to check whether they are overprivileged or not. But, unfortunately, they were unable to separate between necessary and unnecessary permissions because of the limited documentation provided by Android. Moreover, Barrera *et al.* [18] collected 1100 free Android applications and analysed the requested permissions. They primarily focused on the permission system structure, and then they grouped applications together using a neural network and searched for patterns in these permissions. They noted that 62% of applications collected in December 2009 use the Internet permission.

Sharma and Gupta presented in [19] a novel approach they called RNPDroid for risk mitigation using the permissions analysis. Also, The article written by Razak *et al.* [20] has highlighted the significant findings of risk assessment and the risk zone for Android applications through their tool called EZADroid, which implements a permission-based application to determine the risk zone. In addition, Hammad *et al.* [21] have developed DelDroid, an automated system for determination of least privilege architecture in Android and its enforcement at runtime. A key contribution of DelDroid is the ability to limit the privileges granted to apps without modifying them. This tool utilizes static analysis techniques to extract the exact privileges each component needs.

Our proposed measure basically relied on Enck's and Arp's works. Enck *et al.* [22] introduced Kirin, a tool that relies on security rules to simultaneously prevent the installation of malware and allow legitimate app. These rules are developed by adapting requirements engineering techniques to mitigate malware from an analysis of applications, phone stakeholders and system interfaces. So, when a user wants to install an application, the Kirin-based software installer extracts the security configuration from the manifest file. Then, the Kirin security service evaluates this configuration. If it fails to pass all predefined security rules, the installer has two choices. The safest choice is to reject the request. The other choice is to use a user interface to replace the analysis results.

The risk rate identification of dangerous applications with Kirin included five steps:

(i) Identification of the assets: in this step, Kirin extracts requested features. For example, if an application requests the permission RECORD_AUDIO, then the asset here is the microphone input.
(ii) Identification of functional requirements.
(iii) Determination of goals and threats for security.
(iv) Definition of security requirements of the asset.
(v) Determination of the limits of the security mechanism.

Kirin studies in all those steps the permissions requested by an application in order to determine its nature.

We also use the requested permissions in our proposed measure to attribute a risk value to an application. In addition to this criterion, we use other indicators such as API calls, intents, function calls, etc.; similar to Enck *et al.*, we based on malware samples and similarity remoteness in our proposition.

Arp *et al.* [23] introduced a lightweight measure they called DREBIN, which can detect and identify malware directly on the device of the Android user. To extract the features of an application such as permissions, API calls and network addresses, DREBIN performs a broad static analysis. Then these features are integrated into a common vector space. The purpose of this geometric representation is to automatically identify combinations and characteristics indicative of malicious applications by exploiting machine learning techniques. DREBIN provides the user with an explanation for each detected application, as well as an overview of the samples of malicious applications already identified.

DREBIN is based on a complete and light representation of applications to determine the indications that characterize a malicious activity according to the following process:

(i) Wide static analysis: In this step, DREBIN extracts the set of features included in manifest and dex files of the application.
(ii) Integration in a vector space: In this second step, DREBIN takes the extracted feature sets and integrates them into a common vector space to geometrically analyse the patterns and the combinations of features related to malicious activities.
(iii) Detection based on learning techniques: In this step, the learning techniques such as linear support vector machines embedding are exploited in the identification of malicious applications.
(iv) Explanation: In this last step, DREBIN provides the user with an explanation about the detected application and the list of functionalities responsible for this detection.

In brief, in our proposed measure, we relied on the similarity remoteness proposed by Enck *et al.* and on the idea of Arp *et al.* to use extract features from an application and compare them to those that belong to malicious activities, in order to attribute a risk value to unknown applications. Besides the static analysis used by Arp *et al.*, we use a dynamic analysis in our measure to detect the application behavior.

3. ISSUE DESCRIPTION

Indicators such as Android permissions, function calls and exploit static and dynamic behaviors can be used in the risk

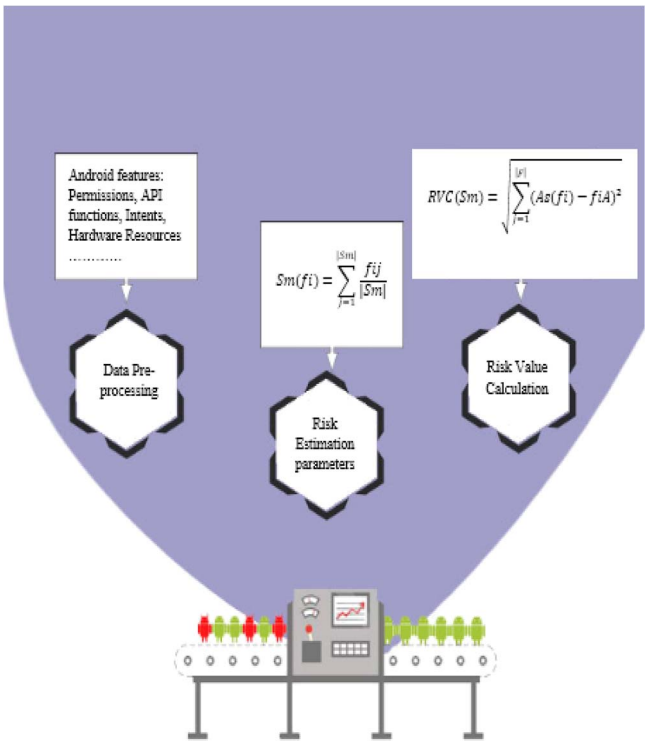

FIGURE 1. Our risk estimate model.

calculation. To apply this, we assume that we have a set f representing all the features of Android operating system where f is defined by the instance $fi = \{f_1, f_2, f_3, \ldots, f_n\}$.

There are two types of features extracted from an application:

- **Static:** This type of functionality includes permissions, intent filters, Java code, network address and hardware components.
- **Dynamic:** This type is about how an application will behave if it is installed on the user's devices.

Figure 1 illustrates how our method we called RVC (risk value calculation) calculates the risk values for an Android application. To get an optimal risk value, we used a set of malware and normal applications carefully selected and saved in a database.

Applications analysis goes into three main steps: preprocessing or extracting the application data, estimating the risk by using the extracted feature and obtaining final risk values.

Preprocessing is exploited to extract the application characteristics such as permissions, API function calls, intentions, material resources etc, since these characteristics can have a varied importance on the obtained risk value. To obtain a precise and effective security risk rate, it is necessary to use malicious application samples for calculating higher values. This will provide the highest relative risk value for an unsecured application, as well as the potential for this malicious application. So, when the user wants to install or just use an application that seems to be suspicious, our proposed security measure employs warning signals to invert the user. In fact, this measure makes it possible to establish the application priorities based on the security risks value found. Therefore, our main goal of this work is to provide a reliable and effective security measure to protect Android users by assigning high-risk rates to malicious apps to easily distinguish them. So, finding a high detection rate for malicious applications from a set of unknown applications is strongly needed to ensure the effectiveness of a risk measure.

4. OUR PROPOSITION

Our proposed measure RVC is involved once a user wants to install a new application on his device; our system extracted the various features contained in its APK file in the preprocessing phase. The main concept behind our solution is quite simple, but it guarantees a very interesting results production. Our measure relies on a database that contains a set of Android application representation with their extracted features and their final risk rate. Based on that information, we can get a more accurate representation and a better risk estimate. In our own view, the security risk is directly related to its remoteness from malicious applications when it comes to an unreliable application. Thus, there is a high probability that an application is likely to be malicious, when it is close to all malicious applications samples, because this means that it has the same behavior.

We consider the average behavior of the malicious applications available in our database as a point to represent a malicious behavior in the functions space based on the various malware such as viruses, Trojan horses that attack SMS on the user's device, fake banking applications etc. On the other hand, we calculated the remoteness for each unreliable application and compared it to the malicious and nonmalicious application centers we obtained. The space containing available malware and normal applications is shown in Figure 2.

In order to obtain the risk value for an unknown application, our system calculates the remoteness separately between the malicious and harmless application centers. Various remoteness measurements such as Manhattan, Euclidean, Hamming etc. can be used at this phase. However, Euclidean remoteness is exploited successfully in the classification and detection of malicious Android applications to ensure an effective security risk rate estimate [21–23]. In the case of an unknown application, it is very likely to be safe, if the remoteness to the normal application center is less than the remoteness to the center of the malicious applications. Then, by applying this condition, we used the following formula:

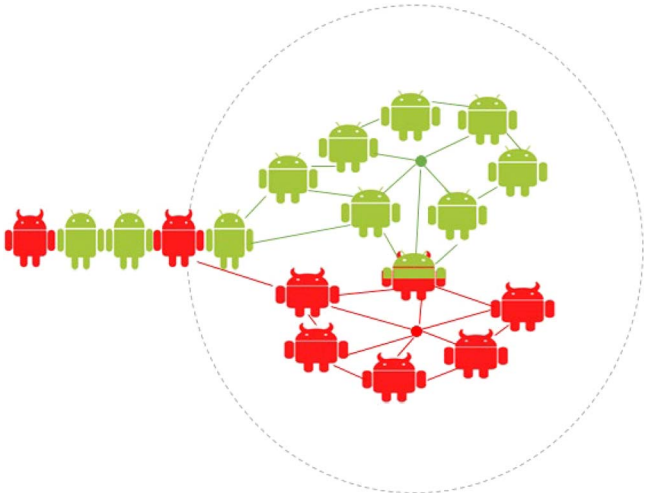

FIGURE 2. Remoteness calculation for an unknown application.

Each malicious application instance in our set is represented by a set of features common between malicious and normal applications. This is noted as $f = \{f_1, f_2, \ldots, f_n\}$, and $m = \{m_1, m_2, \ldots, m_n\}$ denotes the malicious application available sets exploited in the risk rate estimate.

For each feature, the average value of malicious applications is noted by Sm (fi), Sn (fi) representing the average value of normal applications set. The two values are computed based on the two following equations:

$$\forall\, fi \in F\ Sm(fi) = \sum_{j=1}^{Sm} \frac{fij}{Sm} \quad (1)$$

$$\forall\, fi \in F\ Sn(fi) = \sum_{j=1}^{Sn} \frac{fi}{Sn} \quad (2)$$

Sm and Sn in the equations above represent, respectively, the size of malicious and normal applications we have; the feature description in the malicious application and in the normal applications is noted with f_{ij} and f_i, respectively. According to these formulations, we can calculate the malicious applications averages by exploiting the remoteness methods to obtain the risk rates. In other words, as shown in Figure 2, the centers are calculated for points reflection of normal and malicious applications samples. Then, we calculate the risk rate of an unknown application x by using the following equation:

$$\mathrm{RVC}\,(\mathrm{x}) = \frac{\sqrt[2]{\sum_{\mathrm{j}=1}^{|\mathrm{F}|} (\mathrm{Xsm} - \mathrm{fjx})\,2}}{\sqrt[2]{\sum_{\mathrm{i}=1}^{|\mathrm{F}|} (\mathrm{Xsn} - \mathrm{fix})\,2}} \quad (3)$$

RVC (x) in this formulation represents the risk rate calculated based on the instance of an unknown application x. Indeed, by exploiting the equation above, we compute in an isolated way the average points by applying Formulations (1) and (2) for all available malicious applications.

In addition, we get all the application x remoteness and the average of all its entities. The Formulation (3) is used to calculate the estimated risk value for the considered application x. This equation constituted of the two average points obtained for all the types of available applications. The risk value estimate for application x is gotten by comparing the malicious applications average behavior available in our samples with the application x remoteness and inversely related to the normal applications average behavior compared to the application x remoteness.

The data measured in this remoteness are adequate on the same scale. The pseudo-code of the main steps for calculating parameters and measuring the risk rate for a given application list Lx is illustrated by Algorithm 1. This list can contain one application when a user wants to install an application or all the already installed applications on his device.

The symbols used in Algorithm 1 are represented in Table 1. The Algorithm 1 is used to calculate the average value of the normal and malicious applications by applying Formulations (1) and (2), which provide all the available features, the malicious applications set Sm and the normal applications set Sn represent the Algorithm 1 inputs. This algorithm execution helps us to get the average of all applications.

Lines 3 to 11 calculate the average value for all features separately. The normal applications average behavior is measured by lines 3 to 6 and for malicious applications by lines 7 to 10. Line 14 returns these values. The risk value RVC is calculated for each application x belonging to Lx by executing lines 15 to 21 and obtained by the line 24.

The security and privacy of the end user can be seriously damaged by the various malicious applications existing in the Android market, such as some Trojans that exploit the device monetary resources to send SMS or make advertising call.

TABLE 1. Variable used in Algorithm 1 and their meaning.

Variable	Meaning
Sn	is the normal application samples.
Sm	refers to all malicious applications samples.
F	is the applications feature set.
Lx	represents the unknown application list.
f_i	is the feature ith in the list of available features.
f_{ij}	means either the ith functionality in the normal application samples set or the jth functionality in the malicious application samples set.
RVC(x)	is the function that calculates the risk rate for an unknown application x.
X	is the unknown application
M_1, M_2	temporary variables

Algorithm RVCcompter (F, Sn, Sm, Lx, X_{Sn},X_{Sm})

```
1   Begin
2       for all fi ∈ F do
3           for Appn j ∈ Sn do
4               M1 = M1+fij;
5           end for;
6           XSn(fi) =M1/Sn;
7           for Appm j ∈ Sm do
8               M2 = M2+fij;
9           end for;
10          XSm(fi) =M2/Sm;
11      end for;
12      X=[XSn,XSm]
13      return X;
14      M1=0; M2=0;
15      for all Appx∈ Lx do
16          for all fi ∈ F do
17              M1=(XSm(fi) -fix)²
18              M2=(XSn(fi) -fix)²
19          end for;
20          RVC(x)= ²√M1 /²√M2;
21      end for;
22      return RVC;
23  End;
24
25
```

ALGORITHM 1. RVC calculation.

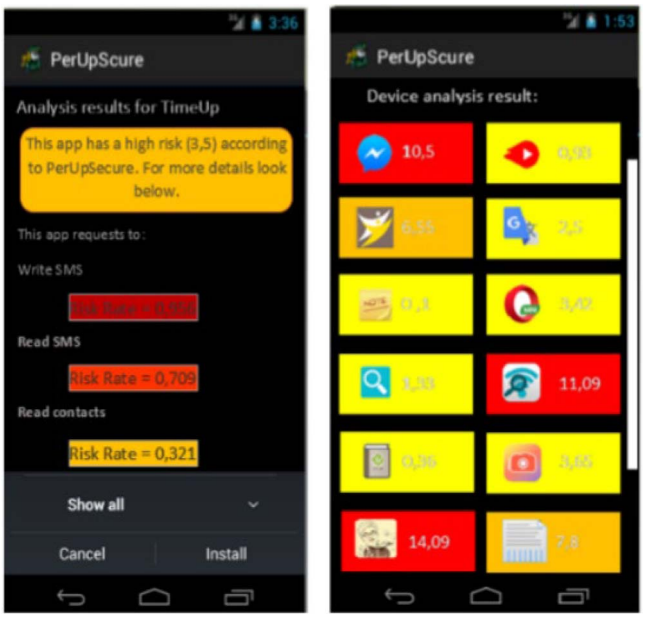

FIGURE 3. PerUpSecure Screenshots.

In addition, there are many malicious applications that aim to disclose and steal the user personal information without his permission or to install new malicious applications in the background without his knowledge. This example shows that all malicious applications have different risk values depending on their malicious actions. The method proposed by our system allows us to take into consideration the available malware samples behaviors to obtain a more accurate risk estimate.

The most dangerous samples may be subject to additional criteria when the average value is obtained for malicious applications. Furthermore, features with different risk values such as the ones that require permissions, like SEND_SMS, INSTALL_PACKAGES and DELETE_PACKAGES, are saved in the list of the most dangerous applications [24–26]. In another example, there are malicious applications that have certain function calls allowing them to access resources requiring monetary information or private and personal information from the device owner. Twenty-six permissions have been identified as critical permissions by Sarma *et al.* [10]. Besides, Yuan *et al.* [27] improved malicious applications detection using 64 API calls, and they exploited 18 dynamic behaviors. By applying these results, we can also guarantee a better risk estimate and adding an additional entity criterion to our proposed model. However, without applying this improvement, the empirical evaluation results of our system show that applied measures provide very good risk values and a very efficient malware detection rate.

Table 2 shows an example of the most five important features extracted from three malicious applications from our predefined dataset.

Starting from Android version 6.0, users are not obliged to accept the permissions requested by an application to install it, so they can now grant permissions when running it. Android 6.0 also provides users with advanced features and control of their applications by allowing them to select the permission granted to an application through a configuration interface [28]. For example, a user may refuse to give an application permission to view his contact list or read his SMS and at the same time allows it to access to the location of his device. For this case, in the implementation of our tool named PerUpSecure, our method RVC measures the risk rate of the applications installed on the user's device as shown in Figure 3. Also, our tool helps the user to know if the application really needs the requested permissions or not.

5. EXPERIMENTAL EVALUATION

For our experimental evaluation, we used the tools Eclipse and MATLAB to provide the necessary codes to implement and evaluate the measures proposed by our system. These tools allowed us to get the risk score by evaluating and comparing two sets of malicious and normal applications.

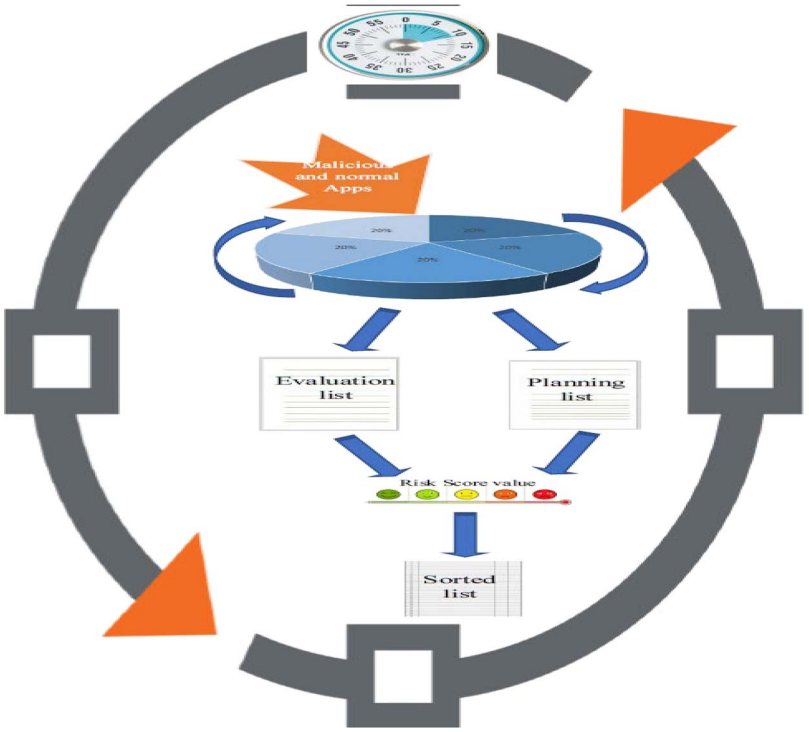

FIGURE 4. Overview of our evaluation process.

TABLE 2. Examples of top five features extracted from three malware

Malware	Feature	Notation	Description	Value	RVC
FakeInstaller	sendSMS	f_3	A suspicious API call	0.76	16.5
	READ_PHONE_STATE	f_6	A dangerous permission	2.1	
	android.hardware.telephony	f_9	A hardware component	0.89	
	SEND_SMS	f_2	A dangerous permission	2.87	
	sendTextMessage	f_4	A restricted API call	2.45	
GingerMaster	READ_PHONE_STATE	f_3	A dangerous permission	2.1	13.43
	system/bin/su	f_1	A suspicious API call	0.57	
	getSubscriberId	f_5	A suspicious API call	0.48	
	USER_PRESENT	f_1	A filtered intent	0.33	
	HttpPost	f_8	A suspicious API call	0.63	
GoldDream	android.provider. Telephony.SMS RECEIVED	f_2	A filtered intent	0.53	12.31
	DELETE_PACKAGES	f_5	A dangerous permission	1.94	
	sendSMS	f_8	A suspicious API call	0.76	
	lebar.gicp.net	f_3	A network address	1.96	
	getSubscriberId	f_4	A suspicious API call	0.82	

We first dealt with all the malicious applications collected on the web by Gates *et al.* [9]. In addition, we evaluated all the existing applications on the Arp *et al.* [23] official website. Their applications set was a mix of malicious applications and innocent applications from different sources. We have exploited these applications to evaluate and compare our proposed approach with those proposed previously.

Therefore, to obtain a precise and effective risk rate and thus guarantee a better detection rate, it is necessary to obtain lower values for normal applications. Indeed, we applied a detection rate evaluation to our measure RVC, and then we compared it

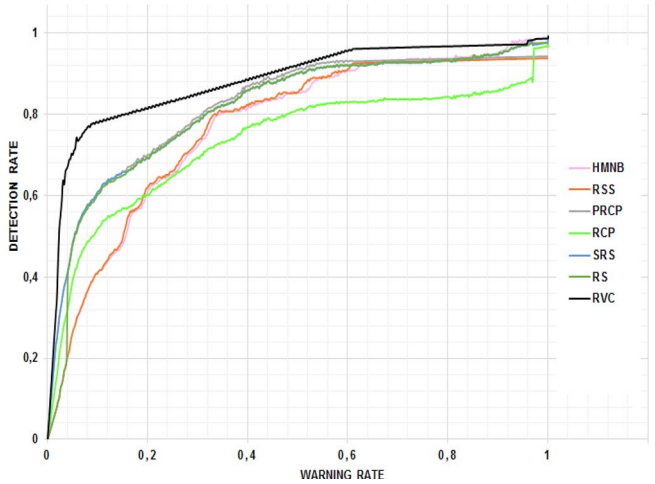

FIGURE 5. All compared measure results.

with that previously proposed by the approaches illustrated in Table 3.

Figure 4 shows how our overall assessment process works. So, as we can conclude from this figure, to evaluate our proposed risk measure, we have created a list containing all the malicious and the normal applications we previously analysed.

This list is sorted in descending order according to the risk values estimated separately for each of its applications by exploiting all available risk indicators. The increase in the risk value importance obtained is mainly related to the number of malicious applications at the list top. Thus, in the sorted list and for various percentage values, the most dangerous applications are selected for the purpose of evaluating our system performance. Figure 5 demonstrates the results obtained for our RVC and for the risk measures shown in Table 3. Each sorted list is selected and identified based on the percentage of the malicious apps. These two actions are called warning rate and detection rate, respectively. The risk values set obtained from comparative measurements is different; at that point, we need to make a fair comparison between them since there is no absolute warning rate threshold value for our evaluation process, and it is also independent of the risk value set.

So, it is obvious that the detection rate is higher with the existence of a very large number of malicious applications placed at the top of the sorted list. So, if we get for example very low warning rate (1% or 5%), it means that our measurement is very strong concerning the calculation of the risk score. Indeed, our detection rate is very high because a large number of malwares exist in the smallest applications set with the highest risk rate. In other words, there is a high probability that the malicious applications have the highest risk score, but they may be normal in some cases. Previous studies [9, 16] have used the same experimental framework.

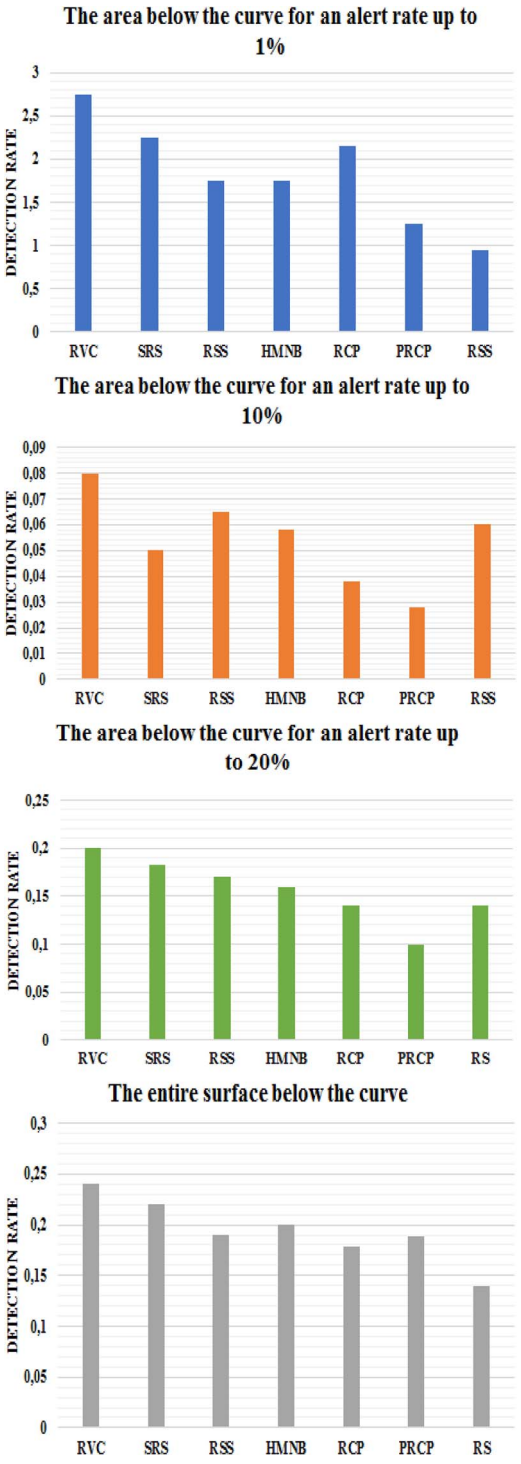

FIGURE 6. Comparison of the area below the curve until several warning rate.

This evaluation is used to make a comparison between malicious applications and normal applications in order to test the execution speed of the risk calculation measures and the high-risk value assignment to these applications.

TABLE 3. Compared measures and their descriptions.

Measures indices	Meaning	Description
RCP	Rarity critical permission	This system proposed by Sarma *et al.* [10] supports risk signals based on permissions rarely requested by Android applications. Indeed, to detect a malicious application, they considered the risk signals that are built using the permissions required by benign and malicious applications.
PRCP	Pairs rarity critical permission	In this signal, Sarma *et al.* [10] used a linear combination of #RCP (x) and #RPCP (y) to calculate a risk score, and then they chose a threshold θ to determine if the signal should classify an application as risky.
RS	Rarity-based risk score	This measurement proposed by Peng *et al.* [16] is characterized by monotony and can provide information on why the risk is high for a specific application and how a developer can reduce that risk. It identifies most of the current high-risk malicious applications. This method allows very critical permissions and less critical permissions to affect the overall score in a way that is easy to understand, making it more intuitive and difficult to escape from other models.
RSS	Risk score based on scarcity with scaling	This method proposed by Peng *et al.* [16] is simply the RS method, but it is strictly based on the fraction of apps requesting specific authorizations. For RSS, the scarcity of permissions is the primary indicator that helps to generate a warning for an application. However, instead of considering only the rarest permissions, this method accumulates the risks for all the requested permissions by the application.
HMNB	Hierarchical mixing of naive Bayes models	This extension of the influence model developed by Gates *et al.* [9] integrates categorical information to hide classes and allows sharing permission information between categories. This model was characterized by the use of applications in all categories and simultaneously by the difference between categories. It produces a mix pattern for each category.
SRS	Score risk based on similarity	This measure proposed by Enck *et al.* [22] detects malware using the similarity distance. It uses this distance to create and evaluate the permissions requested by mobile apps.

The main problem we encountered at the experiment time was to assign higher risk values to the malicious applications with the aim to calculate the detection rate. In our first evaluation, we measured the detection rates based on a warning rate interval between 0 and 1. Figure 5 represents Receiver Operating Characteristic (ROC) curves obtained for all the measures results including ours. Warning rate is presented by the horizontal axis, and the vertical axis shows the detection rate. As shown in the figure, our results are clearly superior to other approaches, especially that the warning rates we obtained are the lowest. Generally, all measurements converge to the full detection rate as the warning rate increases and performance remoteness decreases. However, when the false-positive number is lower, it is more desirable for the user that the warning rate is lower.

Thus, our RVC curve is very close to the one that represents the effectiveness of all the proposed risk score measures. This proves that the RVC assigns higher risk values to malicious applications compared to nonmalicious applications since RVC exploits full information gotten from previous normal and malicious instances. Also, RVC has higher generalization properties because it also uses the average behavior of these applications, which drives all unknown malicious applications to higher levels of detection rate. To better illustrate this experiment, for all the studied methods, the X axis of Figure 6 illustrates the curves obtained for the different warning rate.

We measure the area under our curve with examples of common warning rate values, such as warning rates of (1%, 10% and 20%) or otherwise noted as 0.01, 0.1 and 0.2 since the majority of Android users should be interested in calculating risk rates. Indeed, we calculated in the first zone the curves for the interval between 0 and 0.1. Figure 6 illustrates all the ROC curves and the results obtained; this figure proves that the detection rate of our measurement is the best for all warning rate values, which means that our proposition in comparison with others gives in the all cases better results.

Our proposition exploits examples of previous malicious applications, while other risk scores are based primarily on the use patterns of permissions requested by malicious applications

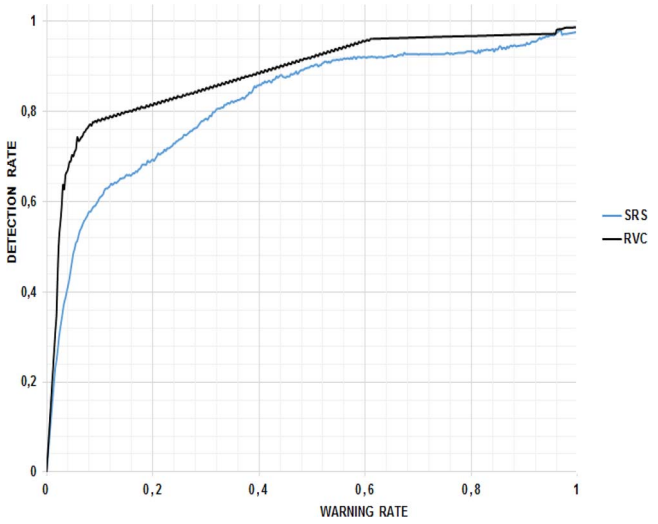

FIGURE 7. Comparison between RVC and SRS.

during installation or take into consideration only the statistics impact on malicious applications. For example, the closest to our proposed RVC curve is SRS proposed by Enck *et al.* [22], its detection rate based only on the permissions scarcity in normal applications. These applications were augmented by using scale factors to improve the impact of certain critical permissions that were manually selected. On the other hand, the information extracted from all normal and malware applications is routinely processed by our proposed RVC to obtain a more realistic risk estimate.

In the second experiment, we evaluated the RVC performance obtained for the Arp *et al.* set and compared it with the SRS risk score for the same set. We based on the assessments made by Arp *et al.* [23] to perform this test and to select permissions and API calls that directly influence the identification and detection of all of malicious Android applications families. We choose SRS for comparison because its performance is the closest to our proposed method. In this comparison, we used the same setting from the first experiment.

Figure 7 illustrates the complete ROC curve we got. So, according to the figure obtained, we deduce that our proposed measure RVC for estimating Android applications risk is more efficient than the SRS, the measure proposed by Enck *et al.* [22] to detect malware using similarity distance, especially when it comes to the lowest warning rates. Our measure performance is also more stable on different warning rates. In addition, although SRS has the closest detection rate to our proposed RVC, it focuses only on the permissions scarcity requested by normal applications with the use of scale factors for increasing the influence of certain critical permissions manually selected from malicious applications.

So, it is possible that a permission or function is rarely used or reflects similar scarcity values in normal or malicious applications. Also, detailed statistics of permissions and functions in malware and normal applications should be considered systematically in order to assign lower risk scores for normal applications and higher risk values for malicious applications. On the other hand, our RVC is characterized by a complete and comprehensive representation of malicious applications and normal applications that allow it to more accurately and efficiently estimate the risk rates of unknown applications. However, to improve the proposed RVC performance, we used a test evaluation between the detection rates obtained by our method and the others. In fact, for the 95% confidence interval, the test result obtained shows the existence of a real offset negligible in the detection rates averages compared to the real values.

6. CONCLUSION

In this article, we propose a new method for classifying and filtering Android apps based on their risk rate we called RVC. To evaluate our proposition effectiveness and performance, we performed a set of assessments on the actual Android applications. The results obtained show that our method is capable of getting relatively high-risk rates for unknown malicious applications as compared to normal or ordinary applications. Indeed, our RVC has a higher detection rate if we compare it with other risk methods. Likewise, our method proposed in this article distinguishes itself with its simplicity, because it is enough to calculate the remoteness between the previous applications (normal and malicious) and an unknown application to estimate its risk rate. In this article, we have classified in the same category all the malicious applications analysed. Moreover, we used a database that contains the malicious applications that have become famous for being the most important and ranked dangerous applications, which allowed us to obtain more precise risk values.

REFERENCES

[1] Struse, E., Seifert, J., Ullenbeck, S., Rukzio, E. and Wolf, C. (2012) PermissionWatcher: Creating User Awareness of Application Permissions in Mobile Systems. In *Proc. AmI 12*, Pisa, Italy, November 13–15, pp. 65–80. Springer, Berlin, Heidelberg.

[2] Neuner, S., van der Veen, V., Lindorfer, M., Huber, M., Merzdovnik, G., Weippl, E. and Mulazzani, M. (2014) Enter Sandbox: Android Sandbox Comparison. In *Proc. MoST 14*, San Jose, CA, USA, May 17, pp. 45–55. IEEE, Washington, DC, USA.

[3] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M. and Rajarajan, M. (2014) Android security: a survey of issues, malware penetration, and defenses. *IEEE Commun. Surveys Tuts.*, 17, 998–1022.

[4] Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M.A. and Paxson, V. (2016) An Analysis of the Privacy and Security Risks of Android VPN Permission-Enabled Apps. In *Proc. IMC 16*, Santa Monica, California, USA, November 14–16, pp. 349–364. ACM, New York, NY, USA.

[5] Xing, L., Pan, X., Wang, R., Yuan, K. and Wang, X. (2014) Upgrading Your Android, Elevating My Malware: Privilege Escalation through Mobile OS Updating. In *Proc. 2014 IEEE Symposium on Security and Privacy*, May 18–21, San Jose, California, pp. 393–408. IEEE, Washington, DC, USA.

[6] Fragkaki, E., Bauer, L., Jia, L. and Swasey, D. (2012) Modeling and Enhancing Android's Permission System. In *Proc. ESORICS 2012*, Pisa, Italy, September 10–12, pp. 1–18. Springer, Berlin, Heidelberg.

[7] Zhou, Y. and Jiang, X. (2012) Dissecting Android Malware: Characterization and Evolution. In *Proc. S&P 2012*, May 22, San Francisco Bay Area, California, pp. 95–109. IEEE, Washington, DC, USA.

[8] Grace, M., Zhou, Y., Zhang, Q., Zou, S. and Jiang, X. (2012) Riskranker: Scalable and Accurate Zero-Day Android Malware Detection. In *Proc. MobiSys12*, Low Wood Bay, Lake District, UK, June 25–29, pp. 281–294. ACM, New York, NY, USA.

[9] Gates, C.S., Li, N., Peng, H., Sarma, B., Qi, Y., Potharaju, R., Cristina, N. and Molloy, I. (2014) Generating summary risk scores for mobile applications. *IEEE Trans. Depend. Sec. Comput.*, 11, 238–251.

[10] Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C. and Molloy, I. (2012) Android Permissions: A Perspective Combining Risks and Benefits. In *Proc. SACMAT 12*, Newark, New Jersey, USA, June 20–22, pp. 13–22. ACM, New York, NY, USA.

[11] Chin, E., Felt, A.P., Sekar, V. and Wagner, D. (2012) Measuring User Confidence in Smartphone Security and Privacy. In *Proc. SOUPS12*, Washington, DC, July 11–13, pp. 1–16. ACM, New York, NY, USA.

[12] Felt, A., Greenwood, K. and Wagner, D. (2011) The Effectiveness of Application Permissions. In *Proc. WebApps 11*, Portland, OR, June 15–16, pp. 75–86. USENIX Association Berkeley, CA, USA.

[13] Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D. (2012) Android Permissions: User Attention, Comprehension, and Behavior. In *Proc. SOUPS12*, Washington, DC, July 11–13, pp. 1–16. ACM, New York, NY, USA.

[14] Kelley, P.G., Cranor, L.F. and Sadeh, N. (2013) Privacy as Part of the App Decision-Making Process. In *Proc. SIGCHI 13*, Paris, France, April 27−May 02, pp. 3393–3402. ACM, New York, NY, USA.

[15] Gates, C.S., Chen, J., Li, N. and Proctor, R.W. (2014) Effective risk communication for android apps. *IEEE Trans. Depend. Sec. Comput.*, 11, 252–265.

[16] Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Cristina, N. and Molloy, I. (2012) Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proc. CCS 12*, Raleigh, North Carolina, USA, October 16–18, pp. 241–252. ACM, New York, NY, USA.

[17] Enck, W., Octeau, D., Mcdaniel, P. and Chaudhuri, S. (2012) A Study of Android Application Security. In *Proc. SEC'12*, San Francisco, CA, August 08–12, pp. 1–38. USENIX Association Berkeley, CA, USA.

[18] Barrera, D., Kayacik, H.G., Van Oorschot, P.C. and Somayaji, A. (2010) A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. In *Proc. CCS 10*, Chicago, Illinois, USA, October 04–08, pp. 73–84. ACM, New York, NY, USA.

[19] Sharma, K. and Gupta, B.B. (2018) Mitigation and risk factor analysis of android applications. *Comput. Electr. Eng.*, 71, 416–430.

[20] Ab Razak, M.F., Anuar, N.B., Salleh, R., Firdaus, A., Faiz, M. and Alamri, H.S. (2019) "Less give more": evaluate and zoning android applications. *Measurement*, 133, 396–411.

[21] Hammad, M., Bagheri, H. and Malek, S. (2019) DelDroid: an automated approach for determination and enforcement of least-privilege architecture in android. *J. Syst. Softw.*, 149, 83–100.

[22] Enck, W., Ongtang, M. and McDaniel, P. (2009) On Lightweight Mobile Phone Application Certification. In *Proc. CCS09*, Chicago, Illinois, USA, November 09–13, pp. 235–245. ACM, New York, NY, USA.

[23] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C. (2014) DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. NDSS14*, San Diego, California, February 23–26, pp. 23–26. Internet Society, Virginie, États-Unis.

[24] Jiao, H., Li, X., Zhang, L., Xu, G. and Feng, Z. (2014) Hybrid Detection Using Permission Analysis for Android Malware. In *Proc. SecureComm14*, Beijing, China, September 24–26, pp. 541–545. Springer, Cham, Switzerland.

[25] Yerima, S.Y., Sezer, S. and Muttik, I. (2015) Android Malware Detection: An Eigenspace Analysis Approach. In *Proc. SAI15*, London, UK, July 28–30, pp. 1236–1242. IEEE Computer Society Security and Privacy Workshops, Washington, DC, USA.

[26] Cui, S., Sun, G., Bin, S. and Zhou, X. (2016) An android malware detection system based on cloud computing. *Chem. Eng. Trans.*, 51, 691–696.

[27] Yuan, Z., Lu, Y., Wang, Z. and Xue, Y. (2014) Droid-Sec: Deep Learning in Android Malware Detection. In *Proc. SIGCOMM14*, Chicago, Illinois, USA, August 17–22, pp. 371–372. ACM, New York, NY, USA.

[28] Zhauniarovich, Y. and Gadyatskaya, O. (2016) Small Changes, Big Changes: An Updated View on the Android Permission System. In *Proc. RAID16*, Paris, France, September 19–21, pp. 346–367. Springer, Cham, Switzerland.