An Effective Risk Computation Metric for Android Malware Detection

Mahmood Deypir*
Faculty of Computer and Information Technology, Shahid Sattari University of Science and Technology, Tehran, Iran
mdeypir@ssau.ac.ir
Ehsan Sharifi
Faculty of Computer and Information Technology, Shahid Sattari University of Science and Technology, Tehran, Iran
sharifi@ssau.ac.ir

Received: 15/May/2016 Revised: 05/Sep/2016 Accepted: 07/Sep/2016

Abstract

Android has been targeted by malware developers since it has emerged as widest used operating system for smartphones and mobile devices. Android security mainly relies on user decisions regarding to installing applications (apps) by approving their requested permissions. Therefore, a systematic user assistance mechanism for making appropriate decisions can significantly improve the security of Android based devices by preventing malicious apps installation. However, the criticality of permissions and the security risk values of apps are not well determined for users in order to make correct decisions. In this study, a new metric is introduced for effective risk computation of untrusted apps based on their required permissions. The metric leverages both frequency of permission usage in malwares and rarity of them in normal apps. Based on the proposed metric, an algorithm is developed and implemented for identifying critical permissions and effective risk computation. The proposed solution can be directly used by the mobile owners to make better decisions or by Android markets to filter out suspicious apps for further examination. Empirical evaluations on real malicious and normal app samples show that the proposed metric has high malware detection rate and is superior to recently proposed risk score measurements. Moreover, it has good performance on unseen apps in term of security risk computation.

Keywords: Mobile Device Security; Risk Computation; Android Malwares; Critical Permissions; Security Metric.

1. Introduction

Android becomes the most popular operating system for smartphones and tablets which made its users the largest target group for security threats. This operating system security architecture reduces the attack surface by restricting applications using permissions and sandboxing. Therefore, in order to perform malicious activities, e.g., stealing user's data, sending premium messages and making phone call, an attacker must deceive users to install a malicious app since other ways of intrusion are almost closed in Android. For installing an app, Android requires the user to grant privileges through the requested permissions. There are large number of applications (Apps) developed for this operating system which requires various permissions based on their functionalities. For an application, these permissions are displayed in the first screen of the installation program. The end user of an Android based mobile device must approve these permissions or discard to install the application. The privileges are remain unchanged until they are revoked from the app when the user issues the app removal process. Although, this security mechanism is very simple and straight forward for users, it causes many challenges. First, users usually does not spend much time for studying the permissions and think about their effects. Therefore, they tend to go forward and to complete the installation process. Moreover, an ordinary user does not have technical skills about the Android permissions and their impacts. Therefore, this security model is not effective regarding to security and privacy of end users in order to preserve their personal information from disclosure or to prevent monetary resource abuse by various type of potential malwares. Consequently, an Android malware e.g., spyware, Trojan, Adware, can deceive the users by introducing itself as a useful app and stole their personal or business data as well as using their mobile phone credit and monetary. There exists some research regarding to enhance the Android security model and its security risk communication mechanism. Using better and intuitive titles for permissions, categorization of permissions based on their effects, reducing the number of permissions by merging similar ones, utilizing user reviews about apps, using visual security indicators for risky apps, and etc. are some samples of these efforts [1-6]. Additionally, a number of statistical and mining models have so far been presented in order to measure the security risk of Android apps. The number of critical permissions and the number of critical permissions combinations requested by an app are simple examples of the statistical measures of security risk for apps [2]. Based on an effective security measure, it can be possible to compute the security risk of an app and fire a warning signal to the user if the computed risk exceeds a predetermined threshold. Moreover, the users can compare similar functionality apps in term of their risk scores. Furthermore, Android markets require an

* Corresponding Author

effective risk computation metric to identify suspicious apps among vast number of newly submitted apps by developers for further examination. The reason is detailed analysis and deterministic malware detection for each app is a very time consuming process and systematic filtering of low risk apps is an important requirement. However, our evaluations show that current measures and models of Android risk computation do not have acceptable performance. That is, they don't compute relative high risk values for known malwares and low risk quantities for benign apps to well recognize malicious apps from non-malicious ones. In this paper, a new security risk score measurement has been proposed which has better performance with respect to previously proposed ones. This risk score benefits from statistics of permission usages in known malicious and clean apps. However, it can be simply extended to other features of Android apps including static and dynamic ones. Moreover, we have attempted to give better definition of permission criticality to aim users for making the best decision for new apps installation. We have shown effectiveness of the proposed metric through extensive experiments on large number of real Android app samples including both malwares and goodwares. The paper is organized as follows. In the next section, some previous research works regarding to Android security and malware detection are reviewed. The problem statement is presented in Section 3. In Section 4, the new security risk score metric is introduced. In this section, our algorithm for risk computation by the proposed metric is also described. Extensive experimental evaluations of the proposed measure with respect to previously proposed ones are presented and illustrated in Section 5. These experiments have been performed using known malwares in the Android world and ordinary useful apps belong to Google App store. Finally, Section 6 concludes the paper.

2. Related Works

The user of a mobile phone participates in their device security by approving requested permissions of an app or decline the permissions which is equal to cancel the installation process. Research findings show that, most users discard checking permissions requested by an Android app. There are researchers trying to overcome this problem and thus enhance the Android security architecture [3-6]. However, Android security architecture requires a simple and straightforward for risk computation of new untrusted applications. Felt et al [3] proposed solutions like changing the categorizations of the Android permissions, emphasizing on the security risk instead of permissions, and a method of approving permissions. In [7] it is suggested that a high level critical information access regarding to the user privacy including personal data, location information, and contact list are displayed instead of the permission names in the first installation page. However, similar to permission list, this high level information might be bypassed by end users. In order to reduce the required space for displaying permissions and assisting the user for fast and effective decision making, in [1] visualizing summary risk and safety scores are suggested. These scores are quantities which can be computed based on various permissions requested by an app. It is shown that, for most users, displaying a summary of risk or safety scores by graphical indicators are more effective than textual information of the permissions in term of user notification. However, metric of risk or safety value computation for untrusted apps is not the main concern in [1]. Peng et al [8] introduce statistical measures and mining models to compute security risk scores and ranking apps based on the requested permissions. The approach can rank the applications in an Android app store like Google play based on their security risk values. Such a ranking aims the users to select more secure apps where there exist a number of apps with the same functionality and different security risk values. Moreover, similar definitions were introduced for the concept of security risk regarding to the list of permissions requested by apps. In [2], the work in reference [8] has been extended and a number of statistical and probabilistic generative risk scores for Android apps using permission usage patterns have been precisely described. All of the measures are defined based on the concept of critical permission which is defined as a permission which can access sensitive software and hardware mobile resources and its usage pattern in malicious apps. An Android malware usually abuses critical permissions and corresponding API functions within its code to perform a malicious activity. The proposed risk scores in [2] and [8] are generative and are mainly computed using benign apps permission usage information. However, for improving the performance, authors increase the impact of some critical permissions on the resulting risk score values. They manually selected nine critical permissions that can be misused by malwares but details of the approach for critical permission selection was not described. In fact, a systematic approach for recognizing critical permissions using information contained in previously known malicious and non-malicious apps is required. An automated system called RiskRanker was introduced in [9] to examine whether a particular app is risky in term of having dangerous behavior. While a mobile antivirus rely on known malware signatures in a reactive manner, RiskRanker system can proactively spot zero day Android malwares. Since deterministic detection of zero day malwares requires further analysis, the system can be used as a preprocessing step to sift through a large number of apps from an Android market by producing a prioritized list of suspicious apps based on their computed security risk. However, for risk computation of untrusted apps RiskRanker only relies on analysis of known malwares and does not take the information of known benign apps into account. Enck et al. [10] developed a system named Kirin which examines combinations of risky permissions to determine whether the permissions requested by an app satisfy a certain global safety policy.

In this system, permission combinations e.g., WRITE_SMS and SEND_SMS are manually specified. These combinations could be used in a malicious apps and therefore, are used to identify malwares. However, a systematic approach for identifying risky permissions or combination of them is required.

A number of approaches have been proposed in the literature to classify Android apps into malwares and benign apps [11-13]. The aim is to construct a mining model like naïve bayes, based on labeled apps augmented by some information regarding to static and dynamic behavior of malwares and clean apps in order to classify future malwares. However, in this context classification models usually suffer from significant misclassification error on unseen data since there is not a crisp boundary between malicious and non-malicious apps. Therefore, measuring amount of risk for newly unseen apps is preferable for decision making compared to deterministic malware detection by classification models. There is other category of researches which use static code analysis of decompiled apps to analysis malicious activates and behaviors within malwares. In this approach, permission to function mapping is performed as a preprocessing step to recognize which function calls are used and what is their ordering. For example, accessing contact list or storage and then sending a SMS is a malicious behavior used in some malwares. In this way, the extracted knowledges and patterns are used to distinguish malicious apps from ordinary applications [14-17]. Malware detection and risk score computation based on static source code analysis can be regarded as complementary method for permission analysis. However, it faces some challenges like code obfuscation and code writing techniques exploited by malware writers which prevent to extract suitable features for risk computation. Dynamic behavior analysis of the running Android apps is another method to detect malwares [18-21]. In this approach, an app is running in a testing environment to identify when and how a part of code is executed and which resources are misused. Both static and dynamic analysis are time consuming processes. Ordinary users and Android markets require fast approach of risk computation.

Permission based security analysis and malware detection are considered by a large number of researches. This is due to its simplicity, explainability, effectiveness, and faster analysis. Moreover, it can be augmented by static and dynamic analysis. A main drawback of this approach is unused permissions of apps since an app can request a permission without actually using it, i.e., over privileged Android apps. This offers opportunities to malware developers to gain access to otherwise inaccessible resources. However, this shortcoming can be overcome by static and dynamic analysis of source code and technique like the function to permission mapping in order to confirm permission usage and remove unused permissions. In [22], a certification technique, is proposed to identify over privileged application in the direction of better risk management assessment. In this technique both runtime information and static analysis are combined to profile mobile applications and identify if they are over privileged or follow the least privilege principle. Coarse grain nature of permission is another problem since granting a permission for an app is equal to allow it to call a couple of API functions. Fortunately, almost all security measures, analysis, and classification based on permissions can also be extended to work using function calls in order to obtain more detailed evaluations. Other challenges and arising issues regarding to Android based security analysis including, incompetent permission administration, insufficient permission documentation, over claim of permissions, permission escalation attack, and TOCTOU (Time of Check to Time of Use) attack were reviewed in [23] and existing countermeasures were addressed. These findings are useful for better risk estimation using requested permissions. Barbara et al [24] proposed an approach to evaluate security models based on permissions by using the self-organizing maps (SOM). They apply the approach on thousands of apps in order to analysis permission distributions. They showed that, how requesting permissions by apps is related to applications categorization. Analyzing decompiled source code of an Android app was used in [25] in order to detect data leak within the app. In [26] a security tool named MAST has been developed to identify high probable malware apps using static code and permission usage analysis. PScout [27] is another Android security tool developed for source code analysis to extract permission to function mapping. Applying this tool on the Android source code reveals that its permission system has a little redundancy and this property remains stable within newer versions of the operating system. DREBIN is a system which works based on detailed set of static features of apps including function call, permission list and hardware usage to recognize malware by an SVM based classifier [28]. Androgaurd is a reverse engineering tool to disassemble and to decompile Android apps. It is designed to analyze malicious and non-malicious Android apps [29]. Some malicious apps repackage malicious codes into benign apps and spread the resulting malwares for easily deceiving end users. Although, this method can be prevented by verifying digital signature of the original apps, some end user might be deceived. In [30], a mechanism named SCSdroid (System Call Sequence Droid) is devised which adopts the thread-grained system call sequences used by apps to extract the truly malicious common subsequences from the system call sequences to identify repackaged malicious apps without requiring the original benign applications. Static dataflow analysis of malwares and goodwares have been utilized in [31] to construct a k-nearest neighbor based classifier. In this classifier, dataflow related API-level features of malicious and non-malicious apps have been used as training samples for future malwares detection. Feizollah et. al in [32] review various types of features including static features, dynamic features, hybrid features and applications metadata which are used in the literature for

Android malware detection. Deterministic recognition of Android malwares encounters some challenges since the boundary between malwares and goodwares is not crisp. Therefore, it is preferred to compute security risk scores of apps instead of binary warning signal regarding to being malwares or goodwares. However, it requires to have effective risk score measurements for precise estimation of the risk value. In this study, we have proposed a new risk score measurement based on a decision making architecture to aim user and systems for making better decisions related to potential Android malwares.

3. Problem Statement

As mentioned previously, users require a convenient method to detect malicious application and make a correct decisions. However, at any time, all malwares and their signature are not fixed and known, i.e., zero day malwares. Therefore, in order to fire a warning signal about using a suspicious application a risk score measurement is desirable. This measure can be exploited in a security tool or embedded in Android to warn a user about malicious apps. It can utilize different aspect of an app to compute its security risk value. These aspect include, permissions, function calls, static or dynamic behavior and etc. Android permissions show what might be called or used in an app. In order to perform malicious activities, a malware requires using critical permissions. Critical permissions are those that can give an app access to sensitive resources and information. Here, permission list of apps are utilized in order to compute their security risk. We assume that there is a set P containing $|P|$ permissions in a mobile operating system: $P = \{p_1, p_2, p_3, \dots, p_n\}$. A mobile application A can request a subset of P to perform its activities. We use a binary variable named x_{Ap} to represent the status of permission x_p in application A. In the other words, x_{Ap} can be set when the permission x_p is requested by an application A. Otherwise it is unset. The problem is to measure the security risk of an input application A using its requested permissions. This measurement requires a formulation and a model which can well exploit historical statistics about previously known malwares and useful apps. For example consider the Table (1) which contains information regarding to permissions requested by a number of known apps including both malwares (+) and goodwares(-).

Table 1. Information about some malwares(+) and useful apps(-)

ID	Permissions	Malware
1	INTERNET, READ_PROFILE	-
2	BATTERY_STATS, BLUETOOTH	-
3	BROADCAST_SMS,WRITE_SMS	+
4	INTERNET,INSTALL_PACKAGE, READ_SMS	+
5	READ_SMS, WRITE_EXTERNAL_STORAGE	-
6	BATERY_STATS, INTERNET	-
7	INSTALL_PACKAGE, READ_PROFILE	-
8	INTERNET, READ_SMS, BLUETOOTH	-

In this table, the second column shows the list of permissions really used in each app. For each app, the status or label of being malicious or useful are depicted in third column. A risk score of an unlabeled app is a value which can be computed based on the list of its permissions. The criticality of each permission is not pre-determined and changes over time. Since the permissions have different criticalities based on their historical usage or misusage, their contribution in computing risk score might be different from each other. A permission's criticality value can be related to its nature and amount of its usage in the previously known malware and goodwares.

An effective security risk score must compute higher values for malware samples than benign apps instances. The more relative risk score value for an untrusted app, the more potentiality of being malware is. In this study, the aim is to propose an effective, simple, and explainable security risk measurement. This measure of security can be used for user warning signal when they are going to install or use a suspicious application. Moreover, it can be used for apps prioritization based on their security risk or safety. Therefore, our aim is not to classify Android apps into malwares and goodwares but we are going to propose a security risk metric which is meaningful for both malicious and useful apps and can well distinguish malwares from goodwares by assigning higher risk values to malicious apps. Therefore, effectiveness of a risk measurement means having high detection rate for malwares within a set of unlabeled apps. Figure (1) illustrates the overall process of our decision making architecture based on the risk computation.

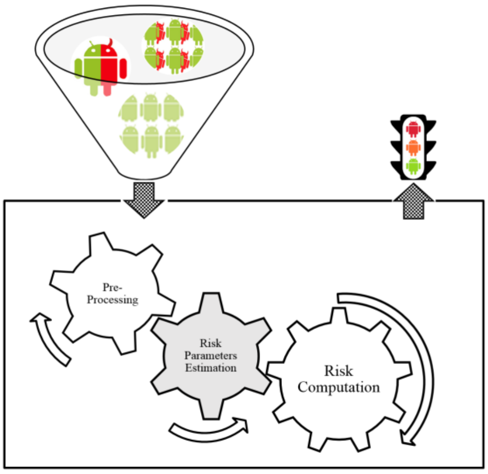


Fig. 1. Overall decision making architecture.

This process computes the risk of untrusted apps using analyzed previously known malicious and clean app samples. As shown in this figure, labeled malwares and benign apps are used to construct the model which consists of three main stage including data pre-processing, risk parameter estimation, and risk computation. The constructed model uses an effective measurement to compute risk of future input apps. In fact, the risk of an untrusted app or set of apps can be computed by the model. The computed risks can be seen as a guiding light for selecting low risk apps for usage or selecting high risk apps, i.e., potential malwares for further analysis which leads to identify Android malicious apps.

4. The Proposed Method

Our evaluation shows that previously proposed criterions for risk measurements of Android apps do not have good performance because they operate based on imprecise definition of the criticality for permissions. We require a simple risk score which precisely benefits from the underlying statistics of known malwares and benign apps and exploit their discrimination power of permissions for identifying new malwares. In order to analysis statistical properties of permissions in apps and defining an effective risk score measure, we have used thousands of normal and malicious Android apps. For each app sample, requested permissions can be extracted using the Android Manifest.xml file exist inside the *apk* package file. Before, gathering statistics, a preprocessing can be performed to remove duplicate apps, i.e., several different versions of the same app and removing useless permissions by permissions to function mapping within each app. We have numbered permissions based on their alphabetical order from 1 to |*P*| where *P* is the set of permissions in Android operating system. In order to obtain a better risk score metric based on permissions, 808 malwares and 71331 benign apps are analyzed. In this study, we have proposed a risk score measurement for effective risk computations of Android apps. As mentioned in [2], a good risk measurement has two main properties, high detection rate and high explainability.

In the devised measurement, we have designate a new formulation to assign higher risk values to permissions which have higher usage in malwares and very lower usage in benign apps. The idea is quite simple but produces interesting results. That is, the security risk of a permission is directly related to its usage in malware and inversely proportional to its usage in non-malicious apps. Given estimated risk values of permissions, one can compute risk of an Android app based on its permission list. We name our risk metric as *RF*(Rarity and Frequency based risk metric). Since the proposed measurement computes the risk values of permissions according to simple statistics of known malwares and useful Android apps, it has good explainability. Users can be effectively informed regarding to danger about approving risky permissions. They can made reasonable decision based on total risk score of an app which can be simply computed using security risks of its requested permissions. Moreover, Android markets can use the devised metric to handle the large number of daily submitted apps for security analysis by filtering out top most risky apps and examining them using time consuming and deterministic malware detection methods.

A. RF Metric

As mentioned previously, we require a simple risk score which precisely benefits from the underlying statistics of known malwares and benign apps. We leverage permission statistics of both malwares and goodwares to devise an effective risk metric. Permissions which are used frequently in malicious apps and rarely required by normal apps must have more impact on risk score measurement. For each permission, frequency of usage in malwares or rarity of usage in goodwares are not solely symptoms of having high risk. The reason is, it might also have high usage in both malwares and goodwares. On the other hand, requesting a permission might be rarely occurred in both normal and malicious apps. Therefore, an effective risk metric must take both of rarity in normal apps and frequency in malwares into account. In the proposed metric, for each Android permission, its frequency in both benign apps and malwares are considered. Based on this idea we have designed *RF* metric for computing security risk of apps according to the following equation:

$$s \quad RF(x_i) = \sum_{p=1}^{|P|} x_{ip}.(\tfrac{N}{C_{pb}+\varepsilon}).(\tfrac{C_{pm}}{M}). \qquad (1)$$

In the above equation, |P| and x_{ip} are total number of permissions and status of p*th* permission in app x_i, respectively. Moreover, C_{pm} and *M* are usage count of p*th* permission in available malicious apps and total number of malwares, respectively. Finally, *N* and C_{pb} are total number of training benign app samples and the count of permission usage in the set of these samples, respectively. *ε* is a very smaller value used to prevent infinite or undefined numbers where the permission is not used by any analyzed normal apps. In this formulation, C_{pb} is computed as follow:

$$C_{pb} = \sum_{i=1}^{N} x_{ip}. \qquad (2)$$

In the above equation, x_{ip} =1 if i*th* app x_i, uses p*th* permission and x_{ip}=0 otherwise. Similarly C_{pb} is computed according to equation (3):

$$C_{pm} = \sum_{i=1}^{M} x_{ip}. \qquad (3)$$

In formulation (1), the higher the score, the more risky the application is. In fact, for an app x_i, formulation (1) is the summation of risks for used permissions in the app. Therefore, *RF* metric can be also defined for each permission x_p as:

$$RF(x_p) = (\tfrac{N}{C_{pb}+\varepsilon}).(\tfrac{C_{pm}}{M}). \qquad (4)$$

The symbols are defined similar to the previous equations. We named this risk score measurement, Rarity and Frequency based risk score measurement (*RF*) since for risk score computation, it takes the impact of both rarity of permissions in benign apps, i.e., the first component of the equation and frequency of them in malicious apps, i.e., the second component of the formulation, into account. As can be inferred from formulation (4), a permission which is used more frequently in normal apps, its impact on risk computation of apps is reduced. On the other hand, a critical permission is frequently requested by malwares. For a permission, as the frequency in malwares and rarity in clean apps is large it is more critical and more risky.

For 20 top most obtained risky permissions, Table (2) represents their rank based on *RF* metric, their weights in malwares and goodwares, and their *RF* risk values. The permissions are sorted in descending order of their RF weights. RF values are not normalized and are computed according to equation (4).

Table 2. Information of top most critical permissions based on *RF* weights

Rank Based on RF	Permission Name	Usage in malwares	Usage in benign apps	RF Metric
1	WRITE_APN_SETTINGS	0.324	0.003	108
2	INSTALL_PACKAGES	0.218	0.003	72.667
3	DELETE_PACKAGES	0.062	0.001	62
4	WRITE_SMS	0.562	0.016	35.125
5	READ_SMS	0.679	0.023	29.522
6	DISABLE_KEYGUARD	0.29	0.014	20.714
7	READ_LOGS	0.269	0.013	20.692
8	RESTART_PACKAGES	0.348	0.022	15.818
9	WRITE_CONTACTS	0.417	0.031	13.452
10	MOUNT_UNMOUNT_ FILESYSTEMS	0.104	0.008	13
11	RECEIVE_SMS	0.46	0.036	12.778
12	CHANGE_WIFI_STATE	0.262	0.021	12.476
13	SEND_SMS	0.489	0.043	11.372
14	RECEIVE_BOOT_COMP LETED	0.566	0.059	9.5932
15	ACCESS_WIFI_STATE	0.671	0.076	8.8289
16	ACCESS_LOCATION_ EXTRA_COMMANDS	0.126	0.016	7.875
17	CALL_PHONE	0.415	0.069	6.0145
18	READ_CONTACTS	0.392	0.085	4.6118
19	READ_PHONE_STATE	0.931	0.222	4.1937
20	ACCESS_NETWORK_ STATE	0.808	0.2941	2.7474

As would be seen in the experimental section, the relative values of estimated risks are considered to compute and to compare the risks of apps. As shown in Table (2), a permission with a relative high usage weight in malwares, might have a lower RF weight and thus low rank with respect to the other permissions. For example in this list READ_PHONE_STATE, has most usage in malwares but it has nineteenth rank regarding to RF risk value. On the other hand, for a permission, the rarity of usage in benign apps solely does not determine amount of risk value since it might be also rarely requested by malwares. For instance, in Table (2), DELETE_PACKAGES is rarer than WRITE_APN_SETTINGS and INSTALL_PACKAGES but it is less risky than these permissions. Based on the proposed risk score measurement we have re-defined the criticality concept of permissions in Android platform.

Criticality of a permission: It is a relative and variable property which directly proportional to its usage in the current malware samples, and inversely proportional to its normal usage in benign apps.

There are some important points regarding the above definition. First, the criticality is a relative property. That is, we cannot categorize permission into two separate sets, i.e., critical and not critical. In the other words, the permission can be compared together based on their criticality or risk value. This value can be estimated using a metric like *RF* in equation (4). The second point is regarding to the variable nature of the criticality. That is, based on permission usage pattern for current malwares and useful apps development, the criticality of permissions and number of critical permissions might be changed over time. It is obvious that the amount of risk for a permission is not fixed and must be periodically recomputed or updated due to developing new malwares and thus new permissions usage patterns. Finally, the last point is about approach for accessing critical resources and sensitive data through permissions by malicious apps. Malware developers are not interested in using some permissions to perform malicious activities due to some reasons despite critical resources and private data access through the permissions. For example, based on our analyses which is partly shown in Table (2), permissions related to using Bluetooth capabilities, i.e., BLUETOOTH and BLUETOOTH_ADMIN are not used frequently in malicious apps and have RF values very close to zero. This might be due to restrictions of using such capabilities.

B. The Algorithm

In this section, the pseudo code of algorithms for computing risk of permissions and apps based on the proposed *RF* metric are described. In these algorithms, it is supposed that preprocessing is performed on all app samples including malwares, benign apps, and untrusted input apps. The preprocessing consist of permission extraction, removing unused permissions, and removing duplicate apps, i.e., various versions of distinct apps, and etc. Figure (2) depict pseudo code of the algorithm for computing *RF* metric for the permissions based on training normal and malicious app samples. Algorithm for prioritizing a set of apps based on their security risk value is shown in Figure (3). This algorithm gets three parameters named *SP*, *SB*, and *SM* which are the set of Android permissions, set of benign app samples, and set of malwares, respectively. In line 1, the number of normal and malicious apps are obtained. In lines 2 through 14 for all permissions, the *RF* metric is computed. For each permission, in lines 3 through 7 counts of the permission usage in normal apps is accumulated in C_{pb} variable. Similarly, using lines 8 through 12 similar counting and accumulation is performed for malwares using C_{pm} variable. According to equation (4), in line 13, for each permission x_p, C_{pb} and C_{pm} are used to compute risk value of the permission based on the rarity value of the permission in normal apps and its frequency value in malicious apps, respectively. Finally, in line 15 a list containing computed risk values of all permissions is returned. These values are used for computing risk values of input apps which is described by the next algorithm.

```
Algorithm RFCompute(SP, SB, SM)
Begin
    N = |SB|; M = |SM|;
    for each permission $x_p \in SP$ do
        for each sample $s \in SB$ do
            If $x_p$ is requested by s then
                $C_{pb} = C_{pb} + 1$;
            end if;
        end for;
        for each sample $s \in SM$ do
            If $x_p$ is requested by s then
                $C_{pm} = C_{pm} + 1$;
            End if;
        End for;
        $RF(x_p) = (\frac{N}{C_{pb}+\varepsilon}) \times (\frac{C_{pm}}{M})$
    End for;
    return RF;
End;
```

Fig. 2. Risk computation for set of permissions

The overall structure of *RFCompute* algorithm consists of an outer loop and two inner loops. The number of rounds for outer loop is equal to |P| which is the number of permissions in *SP* set. First and second inner loops have |*SB*| and |*SM*| numbers of iterations, respectively which are the sizes of benign set and malware set, respectively. Usually the number of analyzed benign apps are greater than the number of malicious apps as you can see in our analysis and experimentation. Therefore, the complexity of *RFCompute* algorithm is calculated as follow:

$$O(RFCompute)=O(|P|\times(|SB|+|SM|))=O(|P|\times|SB|+|P|\times|SM|)= O(|P|\times|SB|) \quad (5)$$

Therefore, as the number of analyzed app is increased, a more time is required for risk computation of Android permission. However, the process of risk computation is performed once and risk value of each future app can be computed using obtained risk of permissions.

Risk computation can be performed for an individual apps. However, risk values of Android apps are also meaningful where untrusted apps are compared together based on their risk or where high risk apps must be identified. For example, when a user wants to compare some same functionalities untrusted apps to select lowest one or when top most risky apps must be selected for further examination to identify zero day malwares in an Android market or in a user device. In this situations, a prioritized list of available untrusted apps is desirable. Figure (3) briefly describes risk computations based on *RF* metric for a list of preprocessed input apps in order to prioritize them according to their risks. In this algorithm, *SP*, *SA*, and *RF*, respectively, are set of permissions, set of untrusted input apps, and the list of computed *RF* risk values of the permissions as illustrated in the previous algorithm. In lines 1 through 3, risk of each input app is computed. In line 4 apps are sorted based their risk and finally the sorted list of apps as well as their risk values are returned in line 5. The sorting order can be either in descending or ascending order based on the application of risk computation.

```
Algorithm RiskPrioritization(SP, SA, RF)
Begin
    1. for each app $x_i \in SA$ do
    2.     $RF(x_i) = \sum_{x_{ip} \in SP} (x_{ip} \times RF(x_{ip}))$;
    3.     end for;
    4. SA= Sort(SA, RF);
// Sort input apps based on their RF risks  in descending order
    5. return SA;
End;
```

Fig. 3. Apps prioritization based on RF metric

C. An Exampl

For better describing the overall process based on the proposed metric, after preprocessing of malicious and clean app samples, consider the following example. In this toy example which is designed similar to a real situation, our approach for computing risk of the permissions and any app *A* is explained.

Example: Suppose that, there is a set of labeled apps including both malwares and useful apps according to Table (1). Here, it is not important how these apps were labeled.

In order to compute security risk score of unknown apps, the risk values of all permissions must be computed. For all Android permissions, statistics regarding to their rarities in goodwares and their frequencies in malwares must be computed to obtain risk score of future apps. Suppose that based on the above example, we have an unlabeled Android app A which requires INSTALL_PACKAGES, INTERNET, READ_SMS, and BLUETOOTH permissions. For these permissions, the value of rarity and frequency are computed. For the first permission, it is requested by one benign and one malicious apps. These values for the second permission are 3 and 1, respectively. READ_SMS is requested by 2 benign and one malicious apps, respectively. Finally, BLUETOOTH is requested only by two normal apps. Based on obtained values of rarity in benign apps and frequency in malwares, the security risk of this app according to equation (1) is estimated as:

RF(A)=RF(INSTALL_PACKAGES)+RF(INTERNET)+RF(READ_SMS)+RF(BLUETOOTH)= (6/1×1/2)+ (6/3×1/2)+ (6/2×1/2)+ (6/2×0/2)= 3+1+1.5+0 =5.5 (6)

As can be inferred from the above computation, in this example, BLUETOOTH permissions don't have any contributions in the resulting value since it was not used by any malware. The computed risk value can be used to prioritize several apps based on their risks. For an app, having a security risk essentially is not a reason for being malicious but it is a warning signal for the user or can be used as a pre-processing step for more detailed analysis. Risk scores of apps are also relative values and can aid users to select low risk apps. That is, having more than one app with the same functionality and various security risk scores, selecting lowest risk app is a more preferable decision.

5. Experimental Evaluation

In order to evaluate the proposed risk score measurements, required codes are developed using Matlab 2013. We have obtained publically available preprocessed malwares and goodwares datasets as well as source codes of some previous approaches belong to authors of reference [2] from the web[1]. For useful ordinary apps, Market 2011 and Market 2012 are used which we named them as Benign 2011 and Benign 2012, respectively, since they contains non-malicious apps of Google app store at year 2011 and 2012 A.D. These dataset contain permission information of 71331 and 136534 useful apps, respectively. Both malwares and benign apps datasets have 122 columns which are alphabetically ordered permissions of apps in recent versions of Android operating system. Table (3) summarizes characteristics of the used datasets for our evaluations.

Table 3. Android apps datasets specifications for evaluation and comparisons

Dataset Name	Number of Apps	Brief Description
Benign2011	71331	Useful apps of Google App store in 2011 A.D
Benign2012	136534	Useful apps of Google App store in 2012 A.D
Malwares	808	A number of known Android malwares

In order to compare the proposed measurement against previously proposed ones, our proposed *RF* and couple of previously proposed risk score measurements have been evaluated. Table (4) summarizes all of these metrics. Some of them are statistical and others are probabilistic mining models. The interested readers are referred to [2], [8], and [23] for more details.

Table (4): Summarization of previous risk scores

Risk Metric	Meaning
RCP	Rare Critical Permission
RPCP	Rare Pairs of Critical Permissions
RS	Rarity based risk Score
RSS	Rarity based risk Score with Scaling
BNB	Basic Naive Bayes model
PNB	Naive Bayes with informative Priors
MNB	Mixture of Naive Bayes models
HMNB	Hierarchical Mixture of Naive Bayes models
Kirin	Certain combinations of dangerous permissions

In the experimentation, the main concern is detection rate. That is, detecting malwares by assigning relative higher risk to them. Using Benign 2011 and Malware datasets, the detection rates are computed with respect to a range of warning rates from 0 to 1.

A. ROC Curves

Figure (4) shows resulting ROC curves of all metrics where horizontal and vertical axes are warning rate and detection rate, respectively. The only exception is Kirin method which contains fixed rules and does not require warning rate parameter. For this method, instead of ROC curve, its fixed detection rate value is depicted by a single point. In order to evaluate a risk metric, we have placed all malwares and ordinary apps in the same list and sort them in descending order of computed security risk values of the metric. The more malwares placed in the top of sorted list, the stronger security risk score is. For evaluation, we use 10-fold cross validation approach. For this purpose, Benign 2011 and malwares are placed in the same list and at each fold, both models are made using 90 percent of the list. Using each model separately, the ordered remaining 10 percent of the list is obtained. For various percentage values, top most security risk score apps are selected from the ordered list. Subsequently, for each model's ordered list, it is determined that what percent of malwares are contained in the selected apps. In this setting, the percentage of selection from each ordered list and determined percentage of malwares are named warning rate and detection rate, respectively. In the other words, number of false positives and true positives are directly proportional to warning and detection rates, respectively. Although, the ranges of computed risk scores of the compared measurements are different, based on this approach, they can be fairly compared together since there is not any absolute warning rate threshold. It is obvious that, as a risk measurement is stronger, a larger number of malwares are resided on the top of the ordered list and thus the measurement has more detection rate. Additionally, a stronger risk score measurement has high detection rate in smaller warning rate e.g., 1%, 5% since in this experimental setting smaller warning rate is equal to smaller fraction of top most risk score apps. In the other words, the end user expects that the top high risk score apps are malicious not normal.

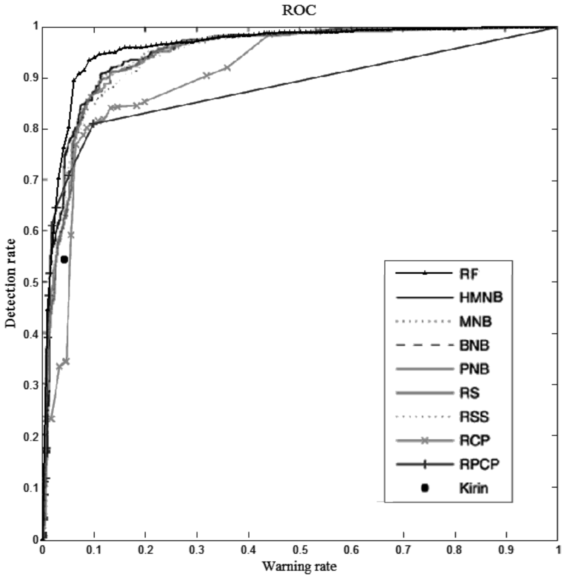


Fig. 4. Detection rate for various warning rates.

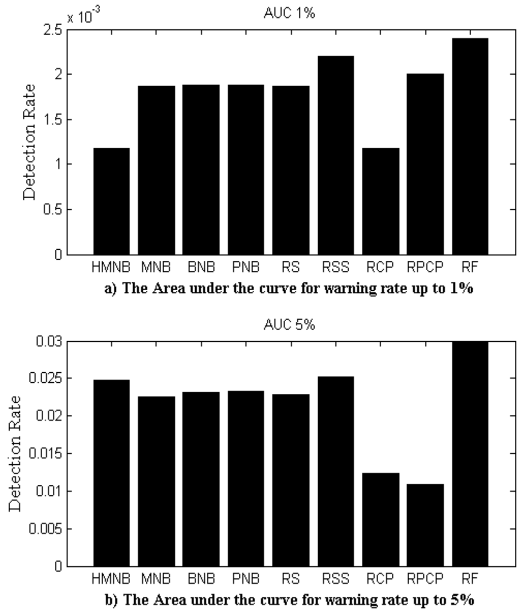
As can be seen from Figure (4), the proposed metric is superior to the other approaches especially for smaller warning rates. As warning rate increases, the performance gaps are reduced and all metrics converge to full detection rate. However, smaller warning rate is more desirable for user where number of false positives are smaller. Moreover, area under the curve for *RF* metric is close to one which shows the effectiveness of the proposed risk

[1] https://github.com/hao-peng/AppRiskPred

score measurement. Therefore, obtained results confirms the superiority of *RF* in term of assigning relative higher risk values to malicious apps than non-malicious ones. The reason is, *RF* considers both rarity of permissions in normal apps and frequency of misused ones in malicious apps.

B. Area Under The Curves (AUC)

For better illustration of this experiment, Area Under Curve *(AUC)* of *ROC* curves are computed for various risk scores metrics since some *ROC* curves are very close to each other especially in larger warning rate values. The *AUC* is computed up for small warning rate values of *ROC* curves. The results is plotted in Figure (5).a and Figure (5).b for 1 percent and 5 percent of warning rates, respectively. Similar results are obtained for other values. As shown in this figure, the proposed *RF* measure has better performance than other metrics. In fact, *RF* is significantly better than other metrics especially for small warning rates where users are interested in. The reason is the better distinguishing power of *RF* which can better differentiate malwares from goodwares. Moreover, the proposed *RF* metric utilizes permission usages statistics of both malwares and goodwares together while the other risk scores mainly focuses on malware or goodware permission usage patterns or manually take the impact of malware statistics into account.



a) The Area under the curve for warning rate up to 1%

b) The Area under the curve for warning rate up to 5%

Fig. 5. Comparison of Area Under Curve (AUC) up to 1% and 5% warning rates.

For example, *RSS* which has closest detection rate to our proposed *RF* metric, considers only the rarity of permissions in benign apps augmented by scaling factors to increase the impact of some manually selected critical permissions. This measurement takes the weights of rare permissions into account and exploit it to compute estimated value of risk. However, a permission may be rarely used in both malicious and non-malicious apps.

C. Performance on Unseen Data

In order to evaluate generalization of the proposed risk measurement, we must apply it on unseen apps. For this purpose, we repeat the above experiment using Benign 2012 and malwares. That is, we obtain usage statistics of permission using Benign 2011 and test it on Benign 2012. In the other words, we use whole set of Benign 2011 for training and whole set of Benign 2012 for test. In training and testing phases, *RF* values of permissions are computed according to usage statistics of the permissions in Benign 2011. Subsequently, detection rates of various warning rates for Benign 2012 and Benign 2011 are computed and resulting ROC curves are obtained and shown in Figure (6). As can be seen from this figure, the metric has high performance for seen and unseen apps. However, for unseen apps, detection rate is slightly degrades. This is due to change in permission usage patterns in newly developed apps which leads to change in risk of permissions and apps. Therefore, in order to obtain better estimation of security risk, usage statistics of permissions must be periodically updated since the criticality values of permissions are not fixed.

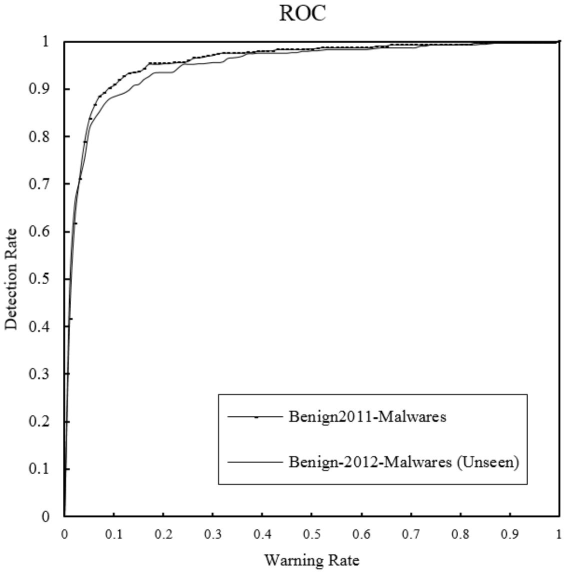


Fig 6. Performance of the proposed risk metric for seen and unseen apps

In fact, permission usage pattern of Android apps is changed over time since new apps with various services and capabilities and thus new permission requirements are introduced in the world. On the other hand, malware developers use new techniques to entice users for malicious apps installation which also leads to change in permission usage pattern.

6. Discussion and Conclusion

In this study, a new risk score metric namely *RF* is devised which has better detection rate with respect to other measurements due to precise identification of the critical permissions. Empirical evaluations on real

Android apps show that *RF* computes relative high risk values for known malwares rather than ordinary apps since it can well differentiate between permissions in term of their usage in malwares and clean apps. As a result, *RF* has high detection rate in comparison to previous risk score measurement. Moreover, the proposed measurement is highly explainable since it can be computed for an app by simply summation of the risk values of critical permissions requested by that app. Risk values of the permissions can be pre-computed using available known malwares and goodwares. An overview on top most critical permissions listed in Table (2) obtained by the proposed metric shows that these permissions are examples of those ones that an app can perform malicious activities by granting a subset of them. In this study, all analyzed malicious apps are categorized into the same category named malwares. However, by using larger and categorized malware datasets we can compute risk scores more precisely. In the other words, exploiting prior knowledge of malware types including Trojan, Adware, Spyware and etc. could enhances the obtained performance since various malware types have different impacts and thus various security risk values. For example, an Adware can be less dangerous than a spyware. Computing *RF* for pair of permissions can further improve the performance of devised approach and thus obtaining better estimation of security risk values. Although the proposed approach is based on permission analysis it can be extended to or completed using other features like Android function calls and dynamic running flow analysis which contain more detailed information.

References

[1] Gates, C. S., Chen, J., Li, N., & Proctor, R. W. (2014). Effective risk communication for android apps. Dependable and Secure Computing, IEEE Transactions on, 11(3), 252-265.

[2] Gates, C. S., Li, N., Peng, H., Sarma, B., Qi, Y., Potharaju, R., & Molloy, I. (2014). Generating summary risk scores for mobile applications. Dependable and Secure Computing, IEEE Transactions on, 11(3), 238-251.

[3] Chin, E., Felt, A. P., Sekar, V., & Wagner, D. (2012, July). Measuring user confidence in smartphone security and privacy. In Proceedings of the Eighth Symposium on Usable Privacy and Security (p. 1). ACM.

[4] Felt, A. P., Greenwood, K., & Wagner, D. (2011, June). The effectiveness of application permissions. In Proceedings of the 2nd USENIX conference on Web application development (pp. 7-7).

[5] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. Tech. Rep. UCB/EECS-2012-26, UC Berkeley.

[6] Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N., & Wetherall, D. (2012). A conundrum of permissions: installing applications on an android smartphone. In Financial Cryptography and Data Security (pp. 68-79). Springer Berlin Heidelberg.

[7] Kelley, P. G., Cranor, L. F., & Sadeh, N. (2013, April). Privacy as part of the app decision-making process. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 3393-3402). ACM.

[8] Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R.,& Molloy, I. (2012, October). Using probabilistic generative models for ranking risks of android apps. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 241-252). ACM.

[9] Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012, June). Riskranker: scalable and accurate zero-day android malware detection. In Proceedings of the 10th international conference on Mobile systems, applications, and services (pp. 281-294). ACM.

[10] Enck, W., Ongtang, M., & McDaniel, P. (2009, November). On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 235-245). ACM.

[11] Jang, J. W., Kang, H., Woo, J., Mohaisen, A., & Kim, H. K. (2016). Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information. Computers & Security.

[12] Sarma, B. P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., & Molloy, I. (2012, June). Android permissions: a perspective combining risks and benefits. In Proceedings of the 17th ACM symposium on Access Control Models and Technologies (pp. 13-22). ACM.

[13] Cen, L., Gates, C., Si, L., & Li, N. (2015). A probabilistic discriminative model for android malware detection with decompiled source code, in Dependable and Secure Computing, IEEE Transactions on, vol.12, no.4, (pp.400-412). IEEE.

[14] Desnos, A. (2012, January). Android: Static analysis using similarity distance. In System Science (HICSS), 2012 45th Hawaii International Conference on (pp. 5394-5403). IEEE.

[15] Schmidt, A. D., Bye, R., Schmidt, H. G., Clausen, J., Kiraz, O., Yüksel, K., & Albayrak, S. (2009, June). Static analysis of executables for collaborative malware detection on android. In Communications, 2009. ICC'09. IEEE International Conference on (pp. 1-5). IEEE.

[16] Zhou, Y., Wang, Z., Zhou, W., & Jiang, X. (2012, February). Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In NDSS.

[17] Aafer, Y., Du, W., & Yin, H. (2013). DroidAPIMiner: Mining API-level features for robust malware detection in android. In Security and Privacy in Communication Networks (pp. 86-103). Springer International Publishing.

[18] Christodorescu, M., Jha, S., & Kruegel, C. (2008, February). Mining specifications of malicious behavior. In Proceedings of the 1st India software engineering conference (pp. 5-14). ACM.

[19] Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and classification of malware behavior. In Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 108-125). Springer Berlin Heidelberg. [17]

[20] Shabtai, A., & Elovici, Y. (2010). Applying behavioral detection on android-based devices. In Mobile Wireless Middleware, Operating Systems, and Applications (pp. 235-249). Springer Berlin Heidelberg.

[21] Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011, October). Crowdroid: behavior-based malware detection system for android. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 15-26). ACM.

[22] Geneiatakis, D., Fovino, I. N., Kounelis, I., & Stirparo, P. (2015). A Permission verification approach for android mobile applications. Computers & Security, 49, 192-205.

[23] Fang, Z., Han, W., & Li, Y. (2014). Permission based android security: Issues and countermeasures. computers & security, 43, 205-218.

[24] Barrera, D., Kayacik, H. G., van Oorschot, P. C., & Somayaji, A. (2010, October). 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 (pp. 73-84). ACM.

[25] Enck, W., Octeau, D., McDaniel, P., & Chaudhuri, S. (2011, August). A Study of Android Application Security. In USENIX security symposium (Vol. 2, p. 2).

[26] Chakradeo, S., Reaves, B., Traynor, P., & Enck, W. (2013, April). Mast: triage for market-scale mobile malware analysis. In Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks (pp. 13-24). ACM.

[27] Au, K. W. Y., Zhou, Y. F., Huang, Z., & Lie, D. (2012, October). Pscout: analyzing the android permission specification. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 217-228). ACM.

[28] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014, February). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS.

[29] Desnos, A. (2013). Androguard-Reverse engineering, Malware and goodware analysis of Android applications. URL code. google. com/p/androguard.

[30] Lin, Y. D., Lai, Y. C., Chen, C. H., & Tsai, H. C. (2013). Identifying android malicious repackaged applications by thread-grained system call sequences. computers & security, 39, 340-350.

[31] Wu, S., Wang, P., Li, X., & Zhang, Y. (2016). Effective detection of android malware based on the usage of data flow APIs and machine learning. Information and Software Technology, 75, 17-25.

[32] Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. Digital Investigation, 13, 22-37.

Mahmood Deypir received his Ph.D. in 2011 and M.Sc. in 2006, both from Shiraz University. He is currently assistant professor in the Computer and Information Technology department at Shahid Sattari University of Science and Technology. His research interests include Data Mining and Cyberspace Security. He has published a number of papers in ISI journals and international conferences.

Ehsan Sharifi received the B.Sc. degree with honors in software engineering from the Shahid Sattari University in 2003 and the M.Sc. degree in software engineering from the PNU University of Tehran, in 2012. He is currently a PhD candidate in software engineering at the Amirkabir University of Technology of Tehran. In 2004, he joined the Department of Computer Engineering and Information Technology of Shahid Sattari University. His current research interests include the Software Quality, Software Modeling, Ontology Engineering, Network Security, and Fuzzy Systems. He has published numerous papers in leading academic journals and conference.