A Skewness-Based Framework for Mobile App Permission Recommendation and Risk Evaluation

Keman Huang[1,2], Jinjing Han[1,2], Shizhan Chen[1,2,3(✉)], and Zhiyong Feng[1,2,3(✉)]

[1] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
{keman.huang, hanjinjing, shizhan, zyfeng}@tju.edu.cn
[2] School of Computer Science and Technology, Tianjin University, Tianjin, China
[3] School of Computer Software, Tianjin University, Tianjin, China

Abstract. Mobile ecosystem has penetrated into people's daily life over these years and most web services are now using mobile application for service consumption. Permission system has been developed to protect the sensitive and valuable information stored in mobile. However, due to the complexity of permission framework, the *permission over-privilege problem* has become a serious problem bringing huge risk for the mobile ecosystem. Therefore, in this paper, we present a skewness-based framework for permission recommendation and risk evaluation, intending to facilitate the permission configuration and identify the risk applications. Specially, the topic model Latent Dirichlet Allocation is presented to build the mapping between app's functionality and permission. Then a two-phase skewness-based filtering strategy is developed and combined with the collaborative filtering framework to remove the abnormal applications and permissions. Finally, the high risk permissions for each application are identified based on the difference between the malicious applications and popular applications. The experiments based on the Apps from Google Play shows that comparing with the state-of-the-art; our approach can effectively remove the abnormal applications and permissions, identify the unexpected and risk permissions, as well as generate the recommended permission configurations with better performance to reduce the permission over-privilege problem.

Keywords: Permission over-privilege · Skewness-based filtering · Unexpected permissions · Risk permissions · Permission recommendation

1 Introduction

With the rapid development of mobile internet technology, mobile terminals and mobile applications have penetrated into people's daily life. The rise of the mobile app ecosystem has drastically changed the way software and services are produced and consumed [1]. More and more web services are now used by end-users through mobile

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 252–266, 2016.
DOI: 10.1007/978-3-319-46295-0_16

apps. As a new and widely accepted approach for service delivery and consumption, it is important and necessary to carry out related researches in the mobile service domain. Actually, by the end of December 2015, the number of available applications on Google Play has been more than 1.87 million. Since more and more sensitive and valuable information are stored in the mobile phone, attacking mobile system to get potential benefits is becoming more and more attractive [2]. In order to protect the mobile system, the permission system has been developed to implement security mechanism. If the application needs to use some protected resources, it must request the relevant permissions. However, due to the complexity of the permission framework, the dialogs based permission framework for end users have been proved invalid [3, 4]: most users just touch the accept button to approve permission request, no matter at installation or during runtime because of risk underestimate [5] while many apps will request unnecessary permissions which brings risk for the use of the apps [6]. Consequently, the *permission over-privilege problem* becomes an important concern for both academia and industry.

Many efforts have been strived to improve the performance of the permission dialogs [7, 8] but fail to lead to the desired effect [4]. Some approaches [6, 9] turn to use the source code analysis to identify the precise set of permissions, while the technology used by malware such as repackaging, and update attack causes obvious delay for the permission detection methods [10]. Therefore, many efforts are shifting towards to a more promising track which uses the machine learning to automatically recommend the required permissions and identify the risk ones from the functionality such as category and description, or similarity with other apps [11–14]. Most of these approaches are based on the assumption that *permissions are well-configured for the popular apps so that they can be used for permission recommendation*. However, conversely, not only the malware applications but also many popular applications are suffering from the permission over privilege problem [6, 15]. H*ow to identify the abnormal permissions and remove the negative-effect* becomes important for permissions recommendations and risk evaluation.

In this paper, we present a framework based on collaborative filtering to help developers and users to configure the permissions. The basic hypothesis here is that "*applications with similar functions should request similar permissions*" [16]. Therefore, we combine the Latent Dirichlet Allocation and collaborative filtering framework to build the mapping between the functionality and the corresponding permissions. Note that, even the popular applications also request some unexpected permissions [6]. Good news is that *if an application request significantly different permissions from other similar ones, then the permissions configuration is abnormal and the application is risky*. Hence, we develop a two-phase skewness-based methodology to identify the abnormal candidate applications and permissions to guarantee the performance of the recommendation. Finally, *malicious applications request not only necessary permissions to enable the functionally, but also risk permissions for attack*. So we use the gap between the popular applications and malicious applications to further identify the risk permissions to evaluate the risk of the application.

Therefore, the main contribution of this paper is the skewness-based collaborative filtering framework for permission recommendation and risk evaluation, consisting of the following folds:

- A two-phase skewness-based strategy is developed to identify and remove the abnormal applications and permissions.
- A methodology based on the permission request difference between malicious and popular applications is proposed to evaluate the risk permissions and applications.
- The experiments based on Apps from Google Plays Store shows the effectiveness of our proposed framework: comparing with the state-of-the-art [17], 83.43 % and 61.61 % improvement in identify unexpected and risk permissions for malicious applications; 188.49 % and 99.13 % MAP improvement of permission recommendation for popular and malicious applications; as well as 26.00 % and 53.47 % permissions reductions for and malicious popular applications.

The remainder of this paper is organized as follows. Section 2 defines the problem and presents the framework's overview. Section 3 details the permission recommendation and risk evaluation. Section 4 reports the experiments. Section 5 discusses the related work and Sect. 6 draws the conclusion.

2 Framework Overview

2.1 Problem Definition

As we are focusing on the relations between the requested permissions and the application's functionality, we can formally define each mobile application as:

$$a_i = <F_i, P_i> = \{<tf_{i,1}, \ldots tf_{i,K}>, <p_{i,1}, \ldots p_{i,n_i}>\} \quad (1)$$

Where $F_i = <tf_{i,1}, \ldots tf_{i,K}>$ refers to the functionality, K is the total number of different functional domains for all the applications; $0 \leq tf_{i,j} \leq 1$ represents the probability that an application is relevant to a certain functional domain. $P_i = <p_{i,1}, \ldots p_{i,n_i}>$ refers to the requested permissions, n_i is the permissions' number.

Therefore, the *permission recommendation* problem can be defined as:

Q1: *given the application a_i and its functionality F_i, how to recommend its permissions RP_i?*

The *risk evaluation* problem can be defined as:

Q2: *given the application a_i, its functionality F_i and its requested permission P_i, how to evaluate its risk?*

2.2 Framework Overview

In order to solve these two problems, as shown in Fig. 1, we present the overview of our collaborative filtering variant which combines the collaborative filtering framework and Latent Dirichlet Allocation to build the mapping between functionality and permission. It consists of the following five processes:

- **Functionality Topic Detection:** The goal for this process is to identify the functionality for all the applications based on their functional descriptions, such as description, title and category. Here we follow the methodology using in [12], using

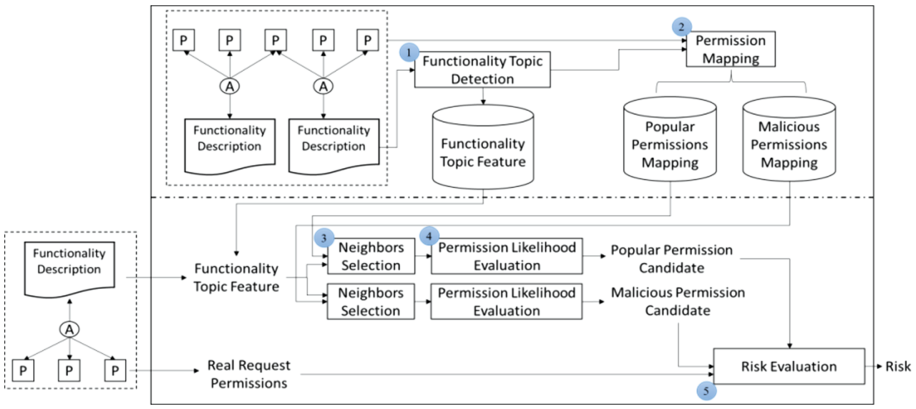

Fig. 1. Overview of Skewness-based Framework (SF)

the Latent Dirichlet Allocation (LDA) on the functional description to cluster applications into different functionality topics so that we can get the probability that an application is relevant to the certain functionality. Note that the summation of all the elements for each topic feature vector equals to 1.

- **Permission Mapping:** For each application, we extract its binary APK file with Apktool[1] and obtain the requested permission features from its manifest file in the APK file. As we will use the difference between popular and malicious applications' permission configurations for risk evaluation, we build the relations between permissions, applications and functionality topics for both the popular applications and the malicious applications separately.
- **Neighbors Selection:** Given an application a_i, based on the functionality topic feature, it is easy for us to get its functionality topic feature $F_i = < tf_{i,1}, \ldots tf_{i,K} >$. Then we can identify its neighbor applications with similar functionality based on the popular and malicious permissions mapping. We will discuss the detail for the neighbor selection in Sect. 3.1.
- **Permission Likelihood Evaluation:** Based on the selected neighbor applications, similar to the user-based collaborative filtering recommendation, we can calculate the permission's likelihood for the given application to generate the recommended permission candidates. Detail will be discussed in Sect. 3.2.
- **Risk Evaluation:** Given the recommended permission candidates, including the popular permission candidates and the malicious permission candidates, we can calculate the difference between them and the real request permissions to evaluate the risk of the given application. Detail will be discussed in Sect. 3.3.

[1] http://ibotpeaches.github.io/Apktool.

3 Permission Recommendation and Risk Evaluation

3.1 *Skewness-Based* Neighbors Selection

As we use LDA to map the applications into the functionality space, each application is represented as a topic feature vector. Therefore, we can calculate the correlation distance between two applications a_i, a_j as follow:

$$Dist(a_i, a_j) = \sqrt{\sum_{k=1}^{K} (fp_{i,k} - fp_{j,k})^2} \tag{2}$$

Then, we can calculate the similarity between two applications a_i, a_j as follows:

$$Sim(a_i, a_j) = \frac{1}{1 + Dist(a_i, a_j)} \tag{3}$$

Therefore, we can get a list of related applications with similarity to the given applications. It is well known that *using a subset of applications for recommendation can gain a better performance in the collaborative filtering framework because of the noise in the dataset*. Therefore, we use the threshold method which obtains a subset of appropriate applications with similarity larger than the given threshold θ_{sim}.

Further, just as we discuss above, the permission over-privilege problem is common for the applications, no matter for the malicious apps or the popular ones. Therefore, we need to remove the abnormal applications to reduce the negative effect of the permission over privilege. In fact, if an application request unusually large or small number of permissions comparing with the majority, then it can be considered as ***abnormal application*** that we should remove it in further processes.

The metric "*Skewness*" is a measure to evaluate the symmetry of a given distribution. If the distribution is symmetric, which means the left side is exactly the same to the right side of the center point, the skewness will be 0. A negative skewness indicates that there is a long tail on the left side that some data are very small while a positive skewness means that some data are very large comparing with the others. Therefore, we can use skewness to identify the abnormal candidate applications for the given application.

Given the candidate applications with similarity larger than the given threshold θ_{sim}, we sort these applications based on its requested permission number and get the permission distribution, $NP(a_i) = <np(a_{i,1}), \ldots np(a_{i,n_i}) >$ where $F(a_i) = <a_{i,1}, \ldots a_{i,n_i} >$ are the candidate application list, $Sim(a_i, a_{i,k}) \geq \theta_{sim}, 1 \leq k \leq n_i$; $np(a_{i,k})$ refers to the number of request permissions for application $a_{i,k}$, $np(a_{i,k}) \leq np(a_{i,l}), 1 \leq k < l \leq n_i$. Its skewness is computed as follows:

$$sk(F(a_i)) = \frac{\frac{1}{n_i}\sum_{k=1}^{n_i}(np(a_{i,k}) - \mu)^3}{(\frac{1}{n_i - 1}\sum_{k=1}^{n_i}(np(a_{i,k}) - \mu)^2)^{3/2}} \tag{4}$$

Where $\mu = \frac{1}{n_i}\sum_{k=1}^{n_i} np(a_{i,k})$ refers to the average number of the request permissions.

If $sk(F(a_i)) = 0$ then no abnormal applications are included in $F(a_i)$ that we use it as the final candidate applications $F^*(a_i)$. Otherwise, if $sk(F(a_i)) > 0$ which means $F(a_i)$ contains the abnormal application with too many request permissions, then we remove the applications with the largest number of permissions in $F(a_i)$; if $sk(F(a_i)) < 0$ which means $F(a_i)$ contains the abnormal application with too few request permissions, then we remove the applications with the smallest number of permissions in $F(a_i)$. This filtering process will be continue until it reaches symmetry that $sk(F(a_i)) = 0$, or only a given percent of neighbors θ_{NA} are retained. In this paper, we set $\theta_{NA} = 80\,\%$.

3.2 *Skewness-Based* Permission Likelihood Evaluation

Given the selected neighbor applications after the skewness-based filtering $F^*(a_i)$, we can calculate the recommend permissions based on the combination of their request permissions. The likelihood that application a_i need permission p_j is calculated as follow:

$$l(a_i, p_j) = \frac{\sum_{a_k \in F^*(a_i)} Sim(a_i, a_k) I(a_k, p_j)}{\sum_{a_k \in F^*(a_i)} Sim(a_i, a_k)} \tag{5}$$

$I(a_k, p_j) = 1$ if application a_k requests permission p_j, otherwise $I(a_k, p_j) = 0$.

Therefore we can get the likelihood vector for the given applications $L(a_i) = < l(a_i, p_{i1}), \ldots l(a_i, p_{iM}) >$ where $l(a_i, p_{ij}) \geq l(a_i, p_{ik}) \geq \theta_{NP}, 1 \leq j < k \leq M$, θ_{NP} is the likelihood threshold. Similarly, we can calculate its skewness as:

$$sk(L(a_i)) = \frac{\frac{1}{M}\sum_{k=i1}^{iM}(l(a_i, p_k) - \overline{\mu})^3}{(\frac{1}{M-1}\sum_{k=i1}^{iM}(l(a_i, p_k) - \overline{\mu})^2)^{3/2}} \tag{6}$$

Where $\overline{\mu} = \frac{1}{M}\sum_{k=1}^{iM} l(a_i, p_k)$ is the average of the likelihood for each permission.

Note that, if $sk(L(a_i)) \leq 0$, then all the permissions have a relative small likelihood value that these permissions should not be considered as the recommended permissions. Additionally, the larger the likelihood value is the higher possibility that the permission is needed for the application. Therefore as detailed in Algorithm 1, when

$sk(L(a_i)) > 0$, we get the permission p_{ij} with the largest likelihood value from $L(a_i)$ and add it into the recommended list, then we revise its likelihood value as $\overline{\mu}$. This process will be ended until the skewness value equals to or smaller than 0.

Algorithm 1. Skewness-based Permission Filtering

Input: $L(a_i)$: likelihood vector of permissions for a_i

Output: $R(a_i)$: recommend permission list for a_i

Procedure:

01. $R(a_i) \leftarrow \phi$
02. **FOR** $1 \le j \le M$
03. **IF** $L(a_i) \neq \phi$ **AND** $sk(L(a_i)) > 0$
04. $R(a_i) \leftarrow R(a_i) \bigcup p_{ij}$
05. $l(a_i, p_{ij}) \leftarrow \overline{\mu} = \frac{1}{M} \sum_{k=1}^{iM} l(a_i, p_k)$
06. **ELSE**
07. **BREAK**;
08. **ENDIF**
09. **ENDFOR**

3.3 Gap-Based Risk Evaluation

The candidate permissions based on the framework represent the expected permissions considering the similar applications' functionality. Then the real request permissions which are not included in the candidates can be considered as the *unexpected permissions* for the given training dataset. Therefore, given the applications a_i, its request permissions $P_i = <p_{i,1}, \ldots p_{i,np(a_i)}>$ and the candidate permissions $RP^*(a_i) = <p_{i,r1}, \ldots p_{i,n_r}>$, the unexpected permissions $UP(a_i)$ can be formally defined as:

$$p_i \in UP^*(a_i) \leftrightarrow p_i \in P(a_i) - RP^*(a_i) \cap P(a_i) \tag{7}$$

Where $* \in \{P, M\}$ refers to the training datasets to get the candidate permissions. P means the candidates are generated based on the popular applications while M means the malicious applications.

Note that the candidate permissions $RP^M(a_i)$ will contain not only the necessary permissions but also the high risk permissions. On the other hand, most of the candidate permissions $RP^P(a_i)$ are supposed to be necessary. Then, the permissions belong to the gap $RP^M(a_i) - RP^M(a_i) \cap RP^P(a_i)$ between the malicious candidates and the good candidates can be considered as the *risk permissions* $RiP(a_i)$ for the given application:

$$p_i \in RiP(a_i) \leftrightarrow p_i \in RP^M(a_i) - RP^M(a_i) \cap RP^P(a_i) \tag{8}$$

Therefore, if an application contains an unexpected permission which also belongs to the risk permissions, then we can consider the application is in risk.

$$a_i \in Risk \leftarrow \exists p_j \in P(a_i),\ s.t.\ p_j \in UP^P(a_i) \cap RiP(a_i) \tag{9}$$

Finally we can calculate its risk as follow:

$$Risk(a_i) = \sum\nolimits_{p_j \in (UP^P(a_i) \cap RiP(a_i))} r(p_j) \tag{10}$$

Here $r(p_j)$ refers to the risk of the permission based on its protection level. The android platform defines four protection levels: *normal, dangerous, signature, signatureOrsystem* and we assign their risk as 1, 2, 3, 4 respectively. For example, if an unexpected permission belongs to dangerous level, then its risk will be 2. Obviously, the more risky permissions requested by the application, the more risk the application will be, so that the user should not install it.

4 Experiment and Discussion

4.1 Data Set

Since Android ecosystem is no doubt the mainstream in the mobile ecosystem and Google Play Store is the most well-known Android application platform, we use the "*app market*" dataset from [18] which consists of 1,402,894 unique .apk files and the metadata such as name, description, version, category, user ratings or downloads. Then we choose the 22,907 applications with more than 100 downloads and five stars in Google play store, forming the "*Popular*" dataset. Furthermore, in order to study the permissions requested by the malware applications, we get the malware dataset from VirusShare[2] which consists of 24,317 malicious applications. Unfortunately, all of these malicious applications don't offer functionality information such as title or description. Therefore we map them into the "*app market*" dataset based on the package identifier and get the "*Malicious*" dataset consisting of 524 applications.

For the permissions in the Android ecosystem, there exist two kinds of permissions: *system permissions* defined by the Android platform and *custom permissions* defined by developers themselves. As the custom permissions are only used by the application itself, we only take the system permissions into account. Furthermore, as the Android platform has grown into different versions, we consider all the permissions which are ever defined, no matter deleted in new version or not, for applications, resulting into 285 unique permissions (see Table 1).

[2] http://virusshare.com.

Table 1. Ovewview of datasets

DataSet	Number of applications
Popular (P)	22,907
Malicious (M)	524
Permissions	Number of permissions
System permissions	285

4.2 Evaluation Metrics

In this paper, we use the following metrics to evaluate the performance:

Mean Average Precision (MAP). Mean Average Precision (MAP) is widely used to evaluate the performance of accuracy for recommendation algorithm which can take the relative order into account. It can be formally defined as follow:

$$MAP = \frac{1}{T}\sum_{k=1}^{T}\frac{1}{N_k}\sum_{j=1}^{N_k}\frac{H_j}{j}I_j \tag{11}$$

Where T refers to the number of applications in testing dataset, N_k refers to the number of the recommended permissions for kth application a_k, H_j refers to the number of actually used permissions in the top j recommended permissions, $I_j = 1$ indicates the permission at jth ranking position is actually used while $I_j = 0$ means it is not used.

Difference Between Recommendations Based on Different Dataset. Note that given an application, the recommended permissions based on different dataset are different. Therefore, we can define the following two metrics to represent this gap: *DMG* refers to the average difference in recommended permission number while *RMG* refers to the average number of the risk permissions:

$$DMG = \frac{1}{T}\sum_{i=1}^{T}\left(|RP^M(a_i)| - |RP^P(a_i)|\right) \tag{12}$$

$$RMG = \frac{1}{T}\sum_{i=1}^{T}|RiP(a_i)| = \frac{1}{T}\sum_{i=1}^{T}\left(|RP^M(a_i)| - |RP^P(a_i) \cap RP^M(a_i)|\right) \tag{13}$$

Obviously, $DMG \leq RMG$. The larger the gap between DMG and RMG is, the more permissions recommended based on the malicious dataset are not included in the recommended permissions based on the popular dataset.

Ratio of Applications with Unexpected Permissions (AUPR). Here we consider the percent of applications which have unexpected permissions in the testing dataset, formally defined as *AUPR*:

$$AUPR = \frac{1}{T}\sum_{i=1}^{T} I(UP^P(a_i)) \tag{14}$$

Where $I(UP^P(a_i)) = 1$, if $|UP^P(a_i)| > 0$. Otherwise $I(UP^P(a_i)) = 0$.

Risk Application Ratio (RAR). As discussed above, our approach can identify whether an application is risk or not. Therefore, we can define the ratio of applications which are considered as risky (RAR) in the datasets as follows:

$$RAR = \frac{1}{T}\sum_{i=1}^{T} I(a_i, Risk) \tag{15}$$

Where $I(a_i, Risk) = 1$ if $a_i \in Risk$; otherwise $I(a_i, Risk) = 0$.

In this paper, we don't consider the time cost as an evaluation metric. Actually, we are using 20-threads for parallel computing, sorting the apps based on the functionality similarity and only considering the related apps, for average, it takes about 0.04 s to generate the permission recommendations for an app, which is definitely acceptable.

4.3 Parameter Selection for Similiarity Threshold θ_{sim}

In our framework, we use the similarity threshold θ_{sim} to select the neighbor applications with reasonal similarity for the given application. Therefore, in order to evaluate its influence, we randomly select 20 % applications from the popular dataset as the testing dataset and the consider the four experiments reported in Table 2.

Table 2. Experiments for similarity threshold

		Training data base	
		Rest 80 % popular applications	Malicious applications
Skewness-filtering for abnormal application	Yes	Popular-Skewness filtering	Malicious-Skewness filtering
	NO	Popular	Malicious

Finally, we vary θ_{sim} from 0 to 1 to generate different permission recommendations. The MAP of the recommendation is reported in Fig. 2. As shown in Fig. 2, it can be seen that for the popular dataset, if $\theta_{sim} > 0.6$, the MAP is decreasing; similarly, if $\theta_{sim} > 0.4$, the MAP for the malicious dataset is decreasing. This is because that with a too larger similarity threshold, most of applications can not have enough neighbor applications to generate a valid recommendation. Therefore, in the rest of this paper, we will set $\theta_{sim} = 0.6$ for the popular permission mapping, and $\theta_{sim} = 0.4$ for the malicious permission mapping.

Additionally, it can be seen that given the selected threshold, using the skewness to filter the permissions with low likelihood can gain a relative better performance. Actually, for the recommendation based on popular applications, it can gain a 14.7 % improvement in MAP.

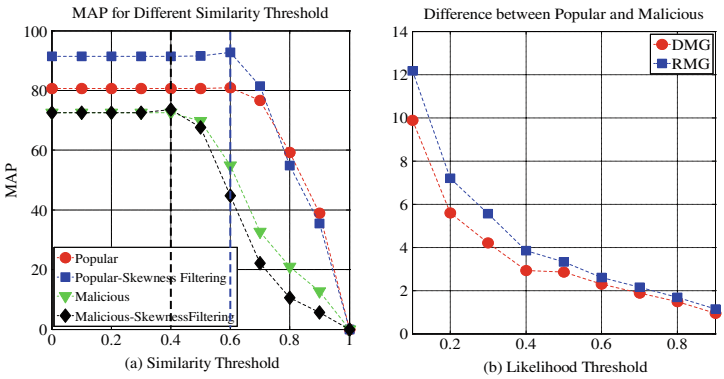

Fig. 2. Similarity threshold selection and gaps between popular and malicious

4.4 Difference Between Popular Applications and Malicious Applications

From Fig. 2 (a), we can see that RP^P gains a better performance than RP^M. This is because that we use the 20 % popular applications as the testing dataset and the malicious applications have a different pattern in permission configuration.

Furthermore, in order to understand the differences between the popular applications and the malicious applications, based on the given θ_{sim}, we can calculate the *DMG* and *RMG* for different threshold of the permission likelihood value. As shown in Fig. 2 (b), it can be seen that DMA and RMG are always larger than 0, which means that *the malicious applications request more permissions than the popular applications*. Additionally, with the increasing of the permission likelihood threshold, these two metrics are both decreasing while the gap bettwen DMG and RMG is also decreasing, representing that *the permissions with high likelihood value are requested by both the popular and malicious applications*. These permissions are the neccessary requirements to support the functionality of the applications.

4.5 Performance Comparison

In order to prove the effectiveness of our approach, we consider the state-of-the-art methodology which has been partially deployed in Chrome Web Store and Google Play Store:

- Peer-group Model (PGM) [17]: In PGM, given the permission used by the candidate application, the applications in the same category, named *peer group*, are used to calculate the ratio of applications using the given permission; if the ratio is

smaller than a given threshold, then the permission is considered as the unexpected permissions. The threshold is set as 0.05.

Then we randomly selected 80 % of the popular applications to train the popular permission mapping, 80 % of the malicious applications to train the malicious permission mapping. The rest of the popular and malicious applications are used as the training data. The similarity threshold is set as $\theta^{P}_{sim} = 0.6, \theta^{M}_{sim} = 0.4$. The likelihood threshold is set as $\theta_{NP} = 0.1$. Table 3 summarizes the improvement of our framework comparing with PGM. Here *ARISK* refers to the average of the risk value for the testing applications, *AP* refers to the average number of the real requested permissions, *ARP* refers to the average number of the recommended permissions.

Table 3. Performance comparing with State-of-the-art

DataSet/Approach		*AUPR*	*RAR*	*ARISK*	*MAP*	*AP*	*ARP*
Popular (P)	*SF*	0.732	0.191	0.701	0.877	6.77	5.01 (–26.00 %)
	PGM	0.258	0.105	0.293	0.304		20.20 **(+198.38 %)**
		/	/	/	**188.49 %**		/
Malicious (M)	*SF*	0.908	0.543	2.536	0.914	8.94	4.16 (–53.47 %)
	PGM	0.495	0.336	1.460	0.459		17.20 **(+92.39 %)**
		83.43 %	**61.61 %**	**73.70 %**	**99.13 %**		/

As shown in Table 3, comparing with PGM, it can be seen that for malicious detections, our approach shows that for the malicious applications, 90.8 % request unexpected permissions and 54.3 % contain risk permissions. However, PGM only finds 49.5 % malicious applications with unexpected permission, and 33.6 % malicious applications use risk permission. Additionally, our approach can assign a 73.7 % higher risk value for the malicious applications, as well as a 99.13 % improvement in MAP. For the popular applications, we get a 188.49 % improvement in MAP.

Note that our approach also shows that even the popular applications will request the unexpected and risk permissions, though the RAR is not as high as the malicious ones. This result is consistent with the observation in [6, 15]. Furthermore, from Table 3, it can be seen that our approach will recommend a fewer number of permissions for both the popular applications and the malicious applications which can help to reduce the over-privilidge problem. Actually, we get a 26.00 % reductions for popular applications and 53.47 % for malicious applications, while PGM even recommends more permissions.

Therefore, comparing with PGM, we can draw the conclusion that our approach can effectively identify the abnormal usage of permission, generate a better recommended permission configuration to reduce the permission over-privilidge.

5 Related Work

Permission system is one of the most important mechanisms to protect the mobile ecosystem. Due to the fact that permission dialogs fail to lead to the desired effect [4], some approaches are turning to identify the risky of the permissions requested by an application [19]. However, as the application's functionality is ignored that it often results into spurious warnings. In order to solve this problem, many efforts are shifting towards to a more promising track which uses the machine learning to automatically recommend the required permissions from the functionality such as category and description, or similarity with other apps [11–14]. Peng et al. [20] uses the similarity with other applications to evaluate the risk of a given application. Pandita et al. [14] and Qu et al. [13] build a permission semantic model to determine which sentences in the description indicate the use of permissions. By comparing the result with the requested permissions, they can detect gap between the description and requested permissions. Wang et al. [11] further identifies the minimum set of permissions an app needs by tailored the requested permissions that are not listed in the semantic permissions in the app descriptions. Gorla et al. [12] develops a tool called CHABADA that uses Latent Dirichlet Allocation (LDA) on app descriptions to cluster Android applications with similar textual descriptions together. The most similar approach to our framework is the peer-group model (PMG) introduced by Jana et al. [17], which has been partially deployed in Chrome Web Store and Google Play Store. In PMG, the software peer group analysis is developed to identify least privilege violation by the proportions of permissions requested by applications from peer group. Peer groups are generated by different information, ranging from pre-defined static categories, list of other related applications, textual descriptions.

Most of these approaches are based on the assumption that permissions are well-configured for good applications or the applications developers are all malicious. By contrast, most of the careless and lazy developers are not actively malicious [17]. Additionally, due to the complexity of the permission system, most of the popular applications also break the least-privileged principle. Therefore, unlike these approaches, based on the assumptions that *the malicious applications will request different permissions with the good applications* and *the application with similar functionality should request similar permissions*, we develops a collaborative filtering variant framework which employs the skewness-based filtering strategy to recommend permissions and uses the difference between good applications and malicious applications to identify the risk permissions.

6 Conclusions

Mobile ecosystem has penetrated into people's every aspect of life recently. More and more web services are turning to offer mobile application for users to consume the services. As the security of the mobile ecosystem becomes an important issue for the web service community, while permission system is designed to protect the mobile ecosystem, how to help the developers and consumers to use the permission system is attracting attentions both from academia and industry. In this paper, based on the

collaborative filter framework using LDA to identify applications' functionality topics, we develop a two-phase skewness-based filtering strategy to remove the abnormal application candidates and the low-likelihood permissions to generate recommended permissions. Furthermore, based on the recommended permissions from popular applications and malicious applications, we identify the unexpected and risk permissions for each application so that we can evaluate their risk just based on the permission configurations. The experiment based on the dataset from Google Play Store shows that comparing with the state-of-the-art:

- For the malicious applications, our approach gains a 83.43 % and 61.61 % improvement in unexpected permission and risk permission identification;
- We gain a 188.49 % improvement for popular applications and 99.13 % for malicious applications;
- More importantly, comparing with the real request permissions, we can cut 26.00 % permissions for popular applications and 53.47 % for malicious ones, which is helpful for reduce the permission over-privilege problem.

In the future, we will further extend our framework to effectively identify the least privilege permissions to help the developers and users for permission configuration. Also we will mitigate the approach to Apache Mahout to guarantee the scalability of the framework.

Acknowledgment. This work is supported by the National Natural Science Foundation of China grants 61373035, 61502333, 61572350 and the Tianjin Research Program of Application Foundation and Advanced Technology grant 14JCYBJC15600.

References

1. Petsas, T., Papadogiannakis, A., Polychronakis, M., Markatos, E.P., Karagiannis, T.: Rise of the planet of the apps. In: Proceedings of the 2013 Conference on Internet Measurement Conference - IMC 2013, pp. 277–290 (2013)
2. Leavitt, N.: Mobile security: finally a serious problem? Computer **44**, 11–14 (2011)
3. Wijesekera, P., Columbia, B., Baokar, A., Hosseini, A., Egelman, S., Wagner, D.: Android permissions remystified: a field study on contextual integrity. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 499–514 (2015)
4. Acar, Y., Backes, M., Bugiel, S., Fahl, S., Mcdaniel, P., Smith, M.: SoK: lessons learned from android security research for appified software platforms. In: 37th IEEE Symposium on Security and Privacy, pp. 1–19 (2016)
5. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: user attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security, pp. 3:1–14 (2012)
6. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS 2011, p. 627 (2011)
7. Liu, B., Lin, J., Sadeh, N.: Reconciling mobile app privacy and usability on smartphones: could user privacy profiles help? In: Proceedings of the 23rd International Conference on World Wide Web, pp. 201–212 (2014)

8. Kelley, P.G., Cranor, L.F., Sadeh, N.: Privacy as part of the app decision-making process. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, p. 11 (2013)
9. Au Kathy Wain Yee, Zhou, Y.F., Huang, Z., Lie, D.: PScout: analyzing the android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 217–228 (2012)
10. Zhou, Y., Jiang, X.: Dissecting Android malware: characterization and evolution. In: Proceedings - IEEE Symposium on Security and Privacy, pp. 95–109 (2012)
11. Wang, J., Chen, Q.: ASPG: generating android semantic permissions. In: Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, pp. 591–598 (2014)
12. Gorla, A., Tavecchia, I., Gross, F., Zeller, A.: Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering, pp. 1025–1035. ACM (2014)
13. Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z.: AutoCog: measuring the description-to-permission fidelity in Android applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS 2014, pp. 1354–1365 (2014)
14. Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T.: Whyper: towards automating risk assessment of mobile applications. In: 22nd USENIX Security Symposium (USENIX Security 13), pp. 527–542 (2013)
15. Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M.: Permission evolution in the Android ecosystem. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 31–40 (2012)
16. Liu, R., Cao, J., VanSyckel, S., Gao, W.: PriMe: human-centric privacy measurement based on user preferences towards data sharing in mobile participatory sensing systems. In: 2016 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 1–8. IEEE (2016)
17. Jana, S., Erlingsson, Ú., Ion, I.: Apples and Oranges: Detecting Least-Privilege Violators with Peer Group Analysis, pp. 1–11 (2015). arXiv:1510.07308
18. Viennot, N., Garcia, E., Nieh, J.: A measurement study of google play. In: Measurement and Modeling of Computer Systems – SIGMETRICS, pp. 221–233 (2014)
19. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: Proceedings of the 16th ACM Conference on Computer and Communications Security - CCS 2009, pp. 235–245 (2009)
20. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: 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 (2012)