The Journal of Systems and Software 163 (2020) 110533

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

In Practice

An Android application risk evaluation framework based on minimum permission set identification

Jianmao Xiao[a], Shizhan Chen[a], Qiang He[b], Zhiyong Feng[a], Xiao Xue[a,*]

[a] College of Intelligence and Computing, Tianjin University, Tianjin, China
[b] School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia

ARTICLE INFO

Article history:
Received 16 May 2019
Revised 20 January 2020
Accepted 21 January 2020
Available online 23 January 2020

Keywords:
Permission overprivilege
App risk evaluation
Minimum permissions
Static analysis
Collaborative filtering

ABSTRACT

Android utilizes a security mechanism that requires apps to request permission for accessing sensitive user data, e.g., contacts and SMSs, or certain system features, e.g., camera and Internet access. However, Android apps tend to be overprivileged, i.e., they often request more permissions than necessary. This raises the security problem of overprivilege. To alleviate the overprivilege problem, this paper proposes MPDroid, an approach that combines static analysis and collaborative filtering to identify the minimum permissions for an Android app based on its app description and API usage. Given an app, MPDroid first employs collaborative filtering to identify the initial minimum permissions for the app. Then, through static analysis, the final minimum permissions that an app really needs are identified. Finally, it evaluates the overprivilege risk by inspecting the app's extra privileges, i.e., the unnecessary permissions requested by the app. Experiments are conducted on 16,343 popular apps collected from Google Play. The results show that MPDroid outperforms the state-of-the-art approach significantly.

1. Introduction

In the past decade, the popularity and ubiquitous use of smartphones have greatly fueled the growth of mobile application (referred to as app hereafter). According to AppBrain,[1] as of September 2018, the number of available apps on Google Play-the world largest Android app store, has exceeded 2.8 million. Apps are now playing an extremely important role in our daily life. Many mobile users store a lot of sensitive and private data on their devices. Such data are at the risk of exposure to malicious activities, which has become a major vulnerability of the entire mobile ecosystem (Roy et al., 2015).

Android has long been a major target of malicious apps (Huang et al., 2015). One of its major vulnerabilities is the permission mechanism (Zhang et al., 2016). Android's permission mechanism requires apps to request permission for accessing sensitive user data, e.g., contacts and SMSs, or certain system features, e.g., camera and Internet access. Thus, the security of Android heavily depends on the effectiveness of this permission mechanism. A major threat is that a malicious app may furtively request extra permissions for accessing users' sensitive and private data. To minimize this threat, some researchers have designed user-oriented permission prompts to ensure that smartphone users are properly notified of the permissions requested by apps. However, due to the complexity of Android's permission mechanism, most of these efforts have proven to be ineffective (Wijesekera et al., 2015; Acar et al., 2016). The main reason is that most users do not fully understand Android's permissions mechanism. They often simply ignore the prompts and accept apps' requests for permissions without inspecting the prompts (Felt et al., 2012b). As a result, apps can easily obtain extra permissions, which increase the risks of user privacy leaks. This is referred to as the over privilege problem (Felt et al., 2011a). A study conducted by Yu et al. (2016) shows that more than 80% of Android apps are overprivileged. The vulnerability of Android's permission mechanism puts mobile users at the risk of privacy leak in the mobile ecosystem (Algarni and Malaiya, 2014). This has become one of the major threats to the health of mobile ecosystem (Wei et al., 2012a). This threat is made even more serious by benign apps that are overprivileged by excessively requested permissions (Felt et al., 2011a; Wei et al., 2012b).

The mainstream approach for enhancing the Android permission mechanism is to identify over-declared permissions requested by an app (Gorla et al., 2014; Pandita et al., 2013; Qu et al., 2014; Wang and Chen, 2014) and recommend reasonable permissions for an app (Huang et al., 2016; Jana et al., 2015; Liu et al., 2019). A major and common limitation of these approaches is that the re-

* Corresponding author.
E-mail addresses: zt_xjm@tju.edu.cn (J. Xiao), shizhan@tju.edu.cn (S. Chen), qhe@swin.edu.au (Q. He), zyfeng@tju.edu.cn (Z. Feng), jzxuexiao@tju.edu.cn (X. Xue).

[1] http://developer.Android.com/guide/platform/index.html.

https://doi.org/10.1016/j.jss.2020.110533
0164-1212/

quested permissions are considered as the permissions that the app really uses. However, this is not always true, especially for malicious apps, they often declare more permissions than they really needs. To address this issue, it is important to identify the minimum permissions, i.e., permissions that are truly needed by an app for the implementation of its functionalities. When given the minimum permission for an app, whether it is a malicious or benign application, the over-declared permissions can be identified and pruned without impacting the functionalities of the app.

In this paper, we propose Minimum Permission for Android (**MPDroid**), an approach for Android app risk evaluation based on the identification of minimum permissions. MPDroid identifies the initial minimum permissions for the target app by inspecting the permissions requested by apps that are similar to the target app, following the main idea of collaborative filtering-*if two users (apps) u and v have similar behaviors (functional description), they will act on other items (permissions) similarly* (Goldberg et al., 2001). Then, it obtains the final minimum permissions for the target app by using a functionality point[2]-permission identify method based on API-used code permission and the app declared permission. The major contributions of this paper are as follows:

- An over-declared permission identification algorithm is proposed. MPDroid employs the LDA technique and an improved collaborative filtering recommendation algorithm to identify and remove over-declared permission by an app. It then obtains a initial minimum permission set corresponding to the app's description (i.e., declaration functionalities).
- We employ static analysis to statically parse app related code permissions and analyze the permissions that the app actually calls. In addition, we present a functionality point-permission set model to further improve the permission configuration of the apps and obtain the final minimum permission set.
- Based on MPDroid, a permission-based risk assessment framework is proposed to detect the risk coefficient for the target app, compared with the state-of-the-art methods, the performance of detecting app risks is improved by 67.5% for the benign apps. In addition, to enable others to use MPDroid, we have published our source code and dataset on GitHub.[3]

The rest of this paper is organized as follows: Section 2 motivates this research. Section 3 presents the risky app identification process. Section 4 evaluates MPDroid experimentally. Section 5 reviews the related work and Section 6 concludes the paper.

2. Motivation

The permissions needed by an app are often related to its functionalities, which can be extracted from the app's description. For example, an application that describes itself as a social networking will likely need permissions related to the mobile device's address book and will need the permission "READ_CONTACTS". A number of malware and privacy-invasive applications have been known to declare more permissions than their purported functionality warrants (Backes et al., 2016), which is usually considered to be unreasonable. Take a screen wallpaper app named bollywoodlive for example. We parsed its APK file and found that it actually applied for WAKE_LOCK, CHANGE_WIFI_STATE and RECEIVE_BOOT_COMPLETED permissions. These permissions are completely irrelevant to its own functional description which may be harm for the privacy of app users.

Table 1
An example of app processed by MPDroid.

App Name	Bollywood Live
declared permission	ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, **CHANGE_WIFI_STATE**, GET_ACCOUNTS, **GET_TASKS**, INTERNET, READ_LOGS, **RECEIVE_BOOT_COMPLETED**, READ_PHONE_STATE, SEND_SMS, SET_WALLPAPER, WAKE_LOCK
After processed by step 1	ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, GET_ACCOUNTS, INTERNET, **SET_WALLPAPER**, READ_PHONE_STATE, **SEND_SMS, READ_LOGS**, WAKE_LOCK
After processed by step 2	ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, GET_ACCOUNTS, INTERNET, READ_PHONE_STATE, WAKE_LOCK

An app should not request more permissions than necessary to support its functionalities, and the developer should minimize the number of permissions required by apps to reduce the app security risk. This is also recommended by the Android official.[4] However, sometimes the permissions requested by an app deviate significantly from the permissions required by the functionalities specified in the app's description (Zhou et al., 2012), not only the malicious app, but also for many benign apps (Felt et al., 2011b), there are also exist declaration the unnecessary permissions problems. In this context, the research problem is defined as:

Q1:Given an app a_i ***and its functional description information*** DF_i***, then how to obtain the minimum permission set that the app really needs?***

To solve the above problem, MPDroid is proposed in this paper with the aim to identify the minimum permissions for an app, which is referred to as the *description-minimum permission set* in our work. The basic idea of MPDroid is to establish a mapping relationship between functionalities and corresponding permissions to identify abnormal permissions. As shown in Fig. 1, the functionalities and the permissions of an app are correlated. An app usually provides multiple functionalities and requests permissions. A mapping relationship between its functionalities and permissions can be established.

Fig. 2 presents MPDroid's process of identifying the minimum permissions of an app. We assume that the target app obtains the permission set PS={*P*1, *P*2, *P*3, *P*4, *P*5, *P*6} extracted from its APK. When the target app is processed by the Over-declared Permissions Identification Module (Step 1), the target app's permission set becomes PS={*P*1, *P*2, *P*4, *P*5}. *P*3 and *P*6 will be removed as over-declared permissions. For instance, Table 1 shows an example of an app processed by MPDroid. The app Bollywood Live applied for 12 permissions. The permission CHANGE_WIFI_STATE, GET_TASKS, RECEIVE_BOOT_COMPLETED are over-declared permissions, because the functionalities in the application description do not refer to these three permissions.

Meanwhile, the Functionality Point-permission Set Identification Module (Step 2) can further identify the target app's permission set PS={*P*1, *P*2, *P*4} as its final *description-minimum permission set*. *P*5 is removed because its support degree (details see in Section 3.4) is too low. For example, In Table 1, permission P5 means SET_WALLPAPER, READ_LOGS, and SEND_SMS. Because their permission support degree is lower than the threshold, they will be filtered out. The lower the permission support degree means the lower possible of the permissions required by the app. Then, the rest permissions we regard as the final minimum permissions set of the app.

[2] The functionality point in this paper refers to functionality topic which is obtained from the description of the app by LDA model.

[3] https://github.com/ztxjm123/MPDroid.

[4] https://developer.android.com/training/articles/security-tips.html#RequestingPermissions.

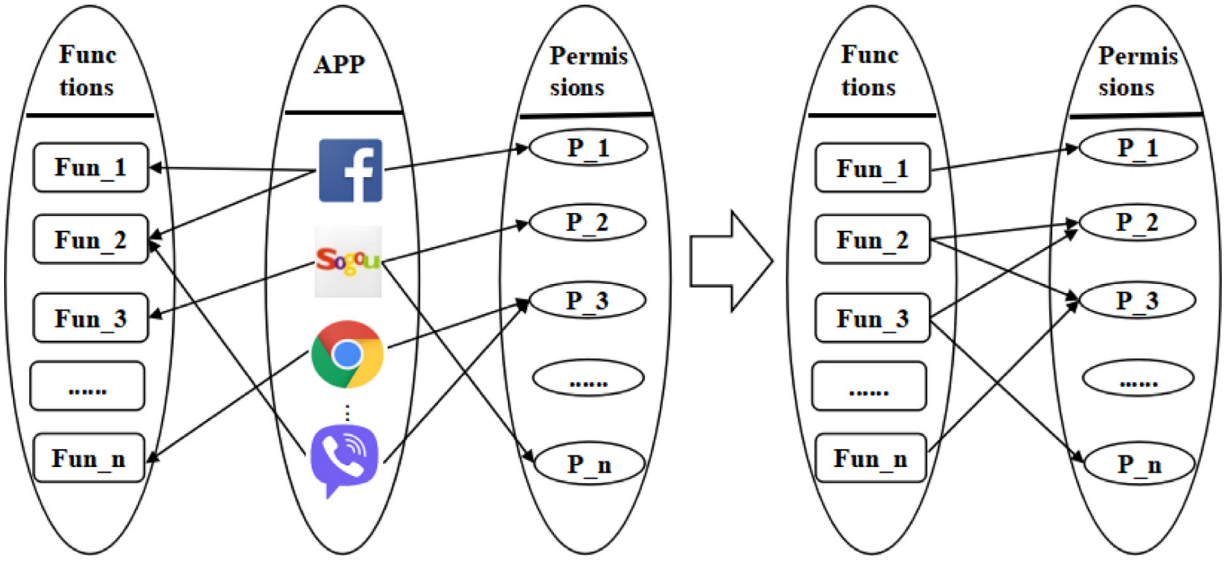

Fig. 1. Association between app's functionalities and permissions.

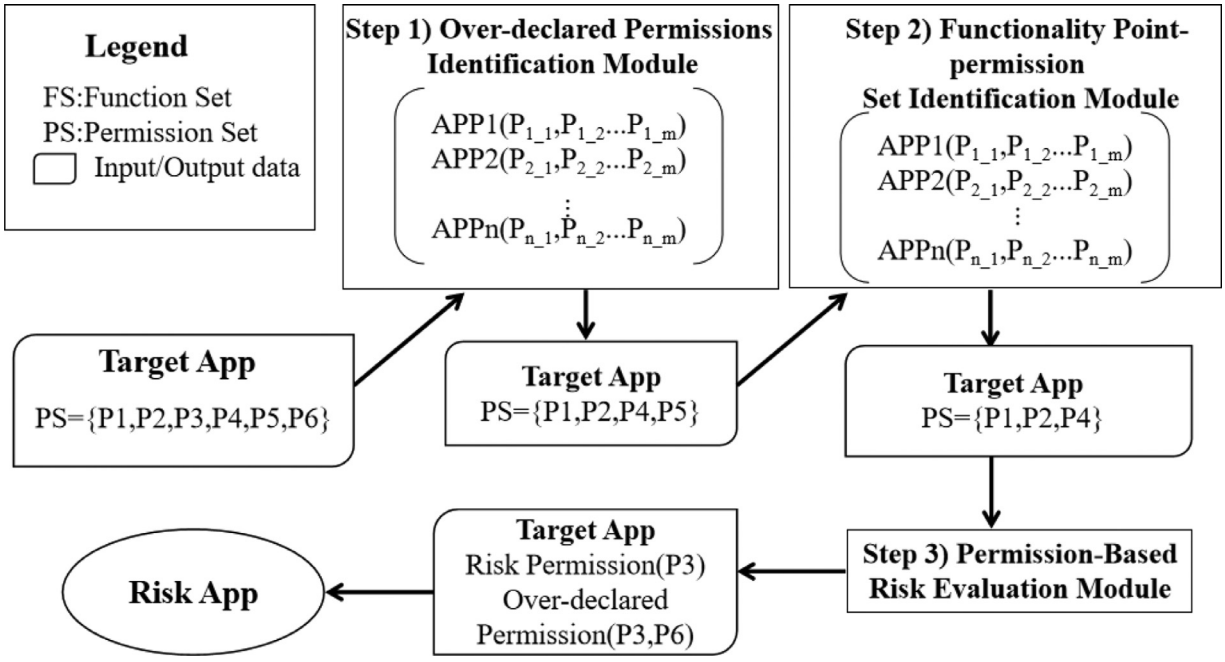

Fig. 2. The process of overprivileged app identification.

Finally, the Permission-Based Risk Evaluation Module (Step 3) identifies the permission *P*3 as a risk permission (defined as Eq. (14) in Section 3.5). As a result, the target app is regarded as a risk app because *P*3 belongs to both unexpected permissions (defined as Eq. (13) in Section 3.5) and risk permissions. The details of each module will be discussed in Section 3.

3. Risk App Identification

MPDroid employs a permission-based app risk evaluation process for measuring the risk level of an app. Fig. 3 shows the process which includes the following 4 phases.

1. Over-declared Permissions Identification. In this phase, MPDroid employs an improved collaborative filtering algorithm to identify and remove over-declared permissions in the app.
2. Initial Description-minimum Permission Set Identification. In this phase, MPDroid iterates the over-declared permissions identification process to obtain the app's initial *description-minimum permission set*.
3. Functionality Point-permission Set Identification. In this phase, MPDroid recommends the app permissions that combine with the actual declared permission and real requested permissions of the app actually uses by calling APIs. As a consequence, MPDroid further refines the initial description-minimum permission set of the app to obtain the final *description-minimum permission set*.

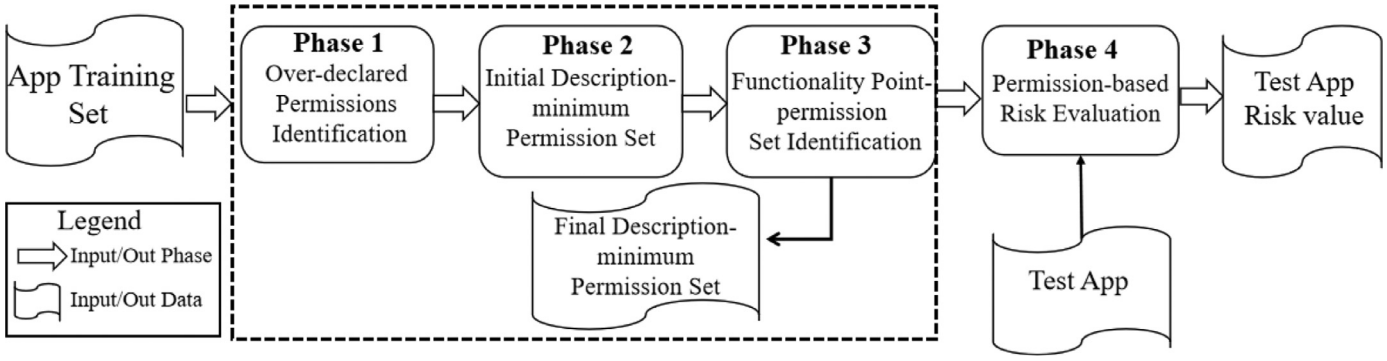

Fig. 3. The overview of MPDroid.

4. Permission-based Risk Evaluation. In this phase, MPDroid calculates the risk level of the app and classifies the app as risky or not.

3.1. Definitions

In order to establish the mapping relationship between app functionalities and permissions, MPDroid first identifies the functionality points implemented by all the local apps' descriptions, and then establishes a mapping relationship between the app and its functionalities. This is referred to as description functionality point extraction. Secondly, MPDroid detects the list of permissions that the app actually requests, and establishes a mapping relationship between the app and its permissions. This is referred as declared permission extraction. Here, we give some formal definitions.

Definition 1. An app a_i is defined as:

$$a_i =< DF_i, DP_i, CP_i, \text{Min}\,P_i > \tag{1}$$

where DF_i represents $a_i's$ functionalities, DP_i represents $a_i's$ declared permissions, CP_i represents $a_i's$ API-based permissions, i.e., permissions parsed from the code, and $MinP_i$ represents $a_i's$ minimum permissions identified by MPDroid.

Definition 2. An app's declared functionalities are defined as:

$$DF_i =< DF_{i,1}, \ldots, DF_{i,k} >, 0 < DF_{i,k} < 1 \tag{2}$$

where $DF_{i,k}$ is the probability of declared functionality point of the app and k is the number of description functionality points contained in app's description.

Definition 3. Declarative permission information for Android app.

$$DP_i =< DP_{i,1}, \ldots, DP_{i,m} > \tag{3}$$

DP_i represents the permission set declared by a_i, and m represents the number of permissions extracted from $a_i's$ AndroidManifest file.

Definition 4. Code permission information for Android apps.

$$CP_i =< CP_{i,1}, \ldots, CP_{i,n} > \tag{4}$$

where CP_i represents the code permissions of a_i. MPDroid first employ static analysis to obtain the Android-related API that the APK calls, followed by the Android-related API to map to the its permissions. Here, n is the number of permissions that are parsed based on the code and $n \geq 0$.

Definition 5. Minimum permission set information for Android app.

$$\text{Min}\,P_i =< \text{Min}\,P_{i,1}, \ldots, \text{Min}\,P_{i,q} > \tag{5}$$

where $MinP_i$ represents $a_i's$ minimum permissions identified by MPDroid, and q is the number of permissions in $MinP_i$.

Based on the above definitions, MPDroid can now perform data (i.e., functionality point and permission data) feature extraction, MPDroid extracts the functionality information and the declared permissions from the textual functional description of the Android app and its corresponding APK file respectively. The goal here is to extract the app data features for building the mapping relation between the declared functionalities and declared permissions for each app. It consists of the following three contents.

Description functionality Point Extraction. We obtain the declared functionality topics for all the app based on their functionality descriptions, and use the Latent Dirichlet Allocation (LDA) on the descriptions to cluster app into different functionality topics. We define the functionality vector set for each app as a *Func* matrix, formulated as follows:

$$Func = \begin{bmatrix} DF_{1,1} & DF_{1,2} & \ldots & DF_{1,n} \\ DF_{2,1} & DF_{2,2} & \ldots & DF_{2,n} \\ \ldots & \ldots & \ldots & \ldots \\ DF_{m,1} & DF_{m,2} & \ldots & DF_{m,n} \end{bmatrix} \tag{6}$$

$DF_{i,j}$ represents the probability that app a_i belongs to topic F_j, $0 < DF_{i,j} < 1$, n represents the number of topics in all app, m represents the number of apps. For the newly target app, we can get the declaration functionality vector of the app by matching the described information with the LDA trained model.

Declared Permission Extraction. For each app, MPDroid extracts its APK file with apktool[5] and obtains the declared permissions from its AndroidMainfest file. By parsing the AndroidMainfest file, the full permission set can be obtained as follows:

$$DP = \begin{bmatrix} DP_{1,1} & DP_{1,2} & \ldots & DP_{1,n} \\ DP_{2,1} & DP_{2,2} & \ldots & DP_{2,n} \\ \ldots & \ldots & \ldots & \ldots \\ DP_{m,1} & DP_{m,2} & \ldots & DP_{m,n} \end{bmatrix} \tag{7}$$

where $DP_{i,j} = 1$ represents that a_i applies for permission DP_j or 0 otherwise.

API-based Permission Extraction. Certain permissions will be required when an app calls the Android APIs. To find out the real requested permissions that an app actually uses by calling APIs, MPDroid adopts an open-source tool named Androguard[6] to statically analyze the app's APK, and obtains the code permissions of the app. MPDroid first traverses all the code files in the APK, and detects the API in the file to obtain all the Android-related methods. Then, according to the result of Pscout (Au et al., 2012), the

[5] http://ibotpeaches.github.io/apktool.

[6] Androguard: https://code.google.com/p/androguard.

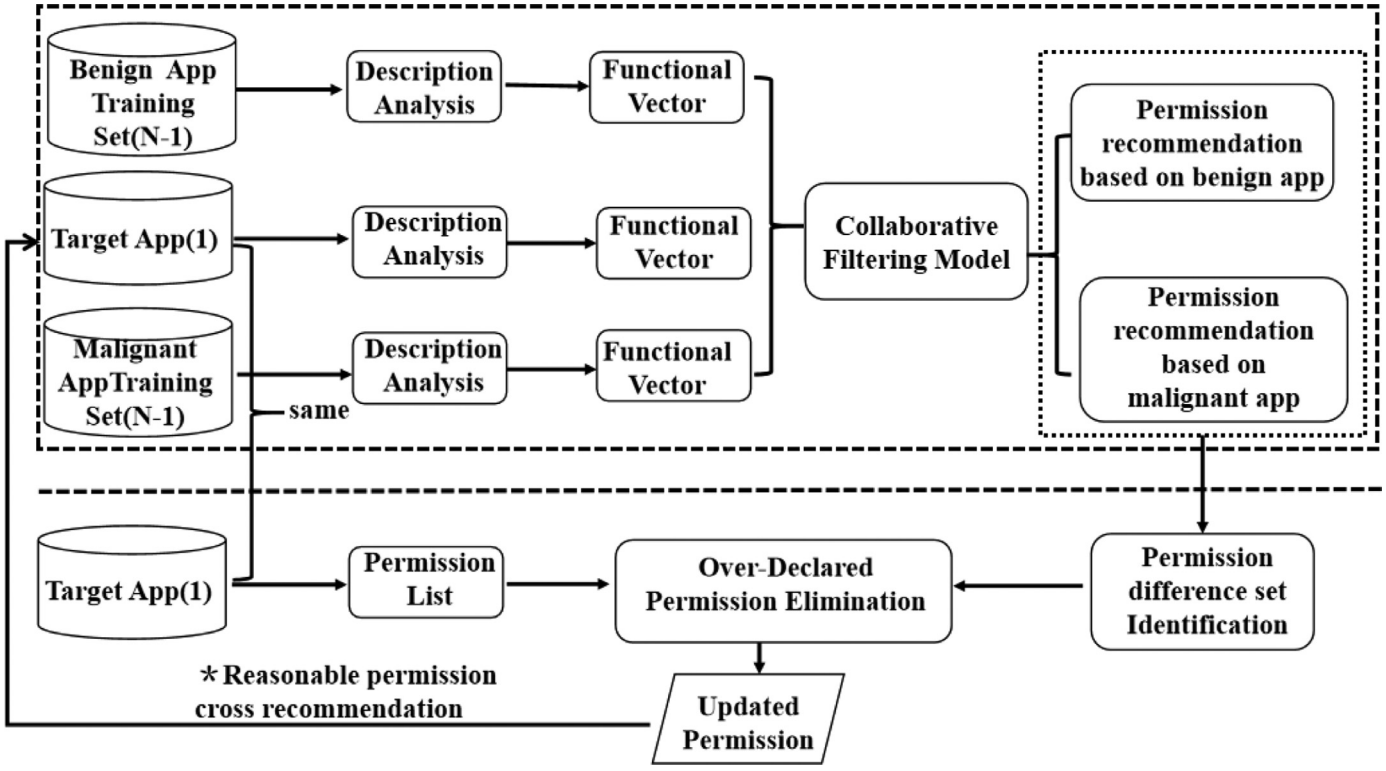

Fig. 4. Overall framework of over-declared permission identification.

correspondence relation table between the API and the permissions is built, and the Android API obtained by the traversal scan is mapped with the permissions. Finally, we obtain the app's code permissions. In our study, 16,343 apps were parsed, and a tree-shaped relationship diagram of app API-permission information is constructed.

Compared to the information extracted from the AndroidManifest file, the permissions obtained from the code are often more accurate, and in addition, the code permissions are the foundation for the implementation of the Functionality Point-Permission Set Identification in Section 3.4.

3.2. Over-declared permissions identification

The purpose of this phase is to identify and remove the permissions that are over-declared in the app by the recommendation algorithm for the target app. Fig. 4 shows the overall framework of the over-declared permission identification in MPDroid.

MPDroid leverages two datasets, i.e., the benign app dataset and the malicious app dataset. We divided the benign app dataset into N parts, N-1 parts form the benign training set, and the remaining part as the target app set that needs to remove the over-declared permissions. For the malignant app, we perform the same operation. Our goal is to map the declared functionality information to the corresponding reasonable permissions for the target app. MPDroid employs an improved collaborative filtering algorithm to recommend permissions for the remaining app set, i.e., the target app. The permission recommendation procedures include following contents.

3.2.1. Similarity computation

We assume that there are m apps. Each app has n functionalities, the relationship between apps and functionalities is denoted by an $m \times n$ matrix, i.e., the *Func* matrix. Each entry DF_{ij} in the matrix represents the probability that the app belongs to this functionality point. The larger the probability value, the more likely the app belongs to this functionality. Conversely, the smaller the probability value, the less likely the app includes this functionality. Here, the "Euclidean distance" is employed for the similarity computation. "Euclidean distance" employs the Eq. (8) to compute the distance between the target app and the app located in the training set:

$$\mathrm{D}ist(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)} \tag{8}$$

where X and Y represent the functionality vectors of app a_i and a_j respectively. $\mathrm{D}ist(X,Y)$ represents the distance between app a_i and a_j, the similarity between app is calculated as Eq. (9):

$$\mathrm{S}im(a_i,a_j) = \frac{1}{1+\mathrm{D}ist(X,Y)} \tag{9}$$

3.2.2. Similar app selection

After calculating the similarity values between the target app and the app located in the training set, a set of similar apps can be identified. On the other hand, in the process of permission identification, if a part of the high similarity application is used for recommendation, the effect will be better than the recommendation using all the applications. It is worth noting that since the number of similar members obtained by the Top-K algorithm is fixed each time, in fact, the number of similar members of the app is not well determined and the number of similar members for each app is not necessarily the same, which may lead the quality of similar members of the app being unguaranteed, and resulting in poor recommendation results. Thus, MPDroid employs the similarity threshold method to obtain a subset of similar app by setting a certain threshold parameter T.

3.2.3. Permission recommendation

MPDroid selects similar app with similarity greater than the threshold T as candidate recommendation members for each tar-

get app, and then it uses the similarity of the declaration functionality vectors of these similar members to perform a comprehensive weighted calculation on the permissions they declare, finally, it obtains a list of the permission recommendation results for the target app. For example, the permission p_i that the app a_i may declare is weighted according to the similarity value to calculate the recommended permission for the target app, it is similar to the user-based collaborative filtering method. The comprehensive recommendation value is defined as $Rv_{a_i,i}$, and the calculation formula is as Eq. (10):

$$Rv_{a_i,i} = k \sum_{a_j \in F} Sim(a_i, a_j) Rv_{a_j,i} \tag{10}$$

where k is a normalizing factor defined as $1/\sum_{a_j \in F} Sim(a_i, a_j)$, F refers to the app recommended member set whose similarity is greater than the threshold T, $Rv_{a_j,i}$ is equals to 1 if app a_j declares the permission, otherwise $Rv_{a_j,i}$ is equals to 0 that indicates app a_j does not declare the permission. After calculating a recommended value vector of the target app, we can get a list of recommended permissions ranking from high recommended value to low. The larger the value is, the higher possibility that the permission is needed for the app. Finally, MPDroid uses the adaptive parameter based method [16] to generate the final permission list.

It is worth noting that in order to avoid the wrong removal of normal permissions, MPDroid uses the malicious apps and the benign apps as the training sets separately, and treats the difference between the recommendation result of the malicious app and the benign app recommendation result as the permission difference set. If the permissions are located in the difference set and also located in the target app's permissions, so we call them the over-declared permissions. Thus for the given target app, the difference set of the two recommended permission is removed from the declared permissions. As the recommendation result of benign app often represents the permissions necessary for the app to implement the functionality, so we do not remove this part permissions. But for the results of malicious app recommend permissions often tend to be more, because in addition to the permissions required to support normal functionalities, there are also include many dangerous permissions. So we believe that the different set of recommendation permissions between a malicious app and a benign app also represents dangerous permissions.

For each app in the training set, MPDroid extracts the declared functionality from the description of the app and then establishs a mapping between functionalities and permissions. Note that our method is a reasonable permission cross recommendation process as shown in Fig. 4, after updating a set of target app permissions, we put the set of apps back into training set and continue to train until all apps in the training set are recommended with reasonable permissions, i.e., removing the over-declared permissions, compared with the Fig. 2 in Section 2, we remove the over-declared permissions (*P*3, *P*6) of the target app through the over-declared permission identification module.

3.3. Initial description-minimum permission set identification

In order to generate a minimum set of permissions corresponding to the description information, we propose an iterative algorithm called description-minimum permission set identification, its purpose is to generate a minimum permission sets corresponding to the Android app description (declaration functionality). The whole process of iterations can be described as follow:

1. Initialize the training set: We divide the entire benign app set into N parts marked as $< A_1, A_2, \ldots, A_N >$. Next we use the rest of N-1 parts $< A_2, A_3, \ldots, A_N >$ to identify the over-declared permissions for the target app set $< A_1 >$ by using the over-declared permission identification method in Section 3.2.
2. Get the over-declared permissions for $< A_1 >$ and then remove them.
3. Update the training set: Put $< A_1 >$ that had removed the over-declared permissions back to the training set and use the next part of the app as the target app to identify the over-declared permissions. For example, we use the app $< A_1, A_3, \ldots, A_N >$ to identify the over-declared permissions for the $< A_2 >$. Finally, all the N parts would be removed the over-declared permissions.
4. After all the N parts have been removed the over-declared permissions, we get the new update dataset, we call this process ***one-time iteration***. Next, we repeat the step 1) to step 3) until the permissions do not change. We finally get the minimum permission set of the benign app after several times iteration.

3.4. Functionality point-permission set identification

MPDroid obtains the initial minimum permission set corresponding to the target app by Description-minimum permission set identification algorithm in Section 3.3. However, that is only the theoretical result. We need to further combine the permissions of the app actually calls to further refine the minimum permission set corresponding to the target app. So in this phase, we will mine the permission set corresponding to the functionality point obtained by the LDA topic model from the perspective of the functionality point-permission set. We use the static parsing permissions and the permission of the app declare itself to build a model to obtain each functionality point corresponding permissions. Thus, it can further refine the initial *description-minimum permission set*. Through this, we can obtain our final *description-minimum permission set*.

We select all the apps with the same functionality topic in *Func* matrix and traverse all the apps for each topic T_m. For each app a_i, the permission that it contains is expressed as $a_i = [P_1, P_2, P_l, \ldots P_N]$. Accordingly, we define the support degree corresponding to each permission as Eq. (11):

$$RP_l(a_i|P_l) = Pr_m \times 1 \tag{11}$$

where $RP_l(a_i|P_l)$ represents the support degree for each permission, Pr_m is the probability corresponding to the functionality T_m.

Next, we add up the values of the same permissions for all apps (assuming that total number of apps is n) under each topic T_m. Then for each permission P_l, its total permission support degree to the topic T_m is:

$$RP_l(T_m|P_l) = \frac{\sum_{i=1}^{n} RP_l(a_i|P_l)}{\sum_{i=1}^{n} \mathrm{Pr}_m(a_i|T_m)} \tag{12}$$

where $\mathrm{Pr}_m(a_i|T_m)$ represents the probability that a_i belongs to T_m. We calculate the permission with the highest m value as the most relevant permission for the topic. If give out a new app with the functionality topic is $a_j = [T_1, T_2, T_m, \ldots, T_N]$, and its corresponding permissions probability is $[Pr_1, Pr_2, Pr_m, \ldots, Pr_N]$. As for the topic T_m, the corresponding most relevant t permissions value is $\{RP_1 \times Pr_m, RP_2 \times Pr_m, \ldots, RP_t \times Pr_m\}$. Then we plus the permission values of all the same permissions in $[T_1, T_2, T_t, \ldots, T_N]$ and arrange them from large to small to obtain the permission that the RP_l value is greater than the value of support degree threshold θ_{support}, i.e., the permissions recommended according to the functionality point directly.

Finally, we inspect whether the initial *description-minimum permission set* in Section 3.3 exists the permission that the support degree is too low. If it exists, we remove these permissions. This way, we obtain the final *description-minimum permission set* according to

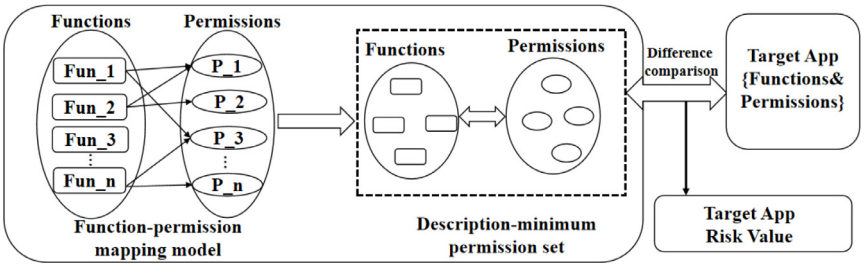

Fig. 5. Permission-based risk assessment framework.

the target app. As show in Fig. 2, the permission set of the target app changes from {*P*1, *P*2, *P*4, *P*5} to {*P*1, *P*2, *P*4} in this phase since *P*5 is the permission whose support degree is too low.

3.5. Permission-based risk evaluation

We can perform permission-based risk evaluation with MP-Droid. Fig. 5 shows a permission-based risk evaluation framework. The key idea is to take use of the *description-minimum permission set*, when a target app comes, we recommend the permissions to the target app by use the collaborative filtering method. Then, we compare the difference between the recommendation permissions and the actually declared permissions of the target app, and through this, we can further calculate the risk value for the target app.

Considering the fact that almost all the malicious apps have declared more permissions than what the original application needs, we calculate the risk level based on the gap between the malicious app and the benign app. Specially, the minimum permission set of the benign app has been identified already. Since the permissions recommended by benign apps often represent normal and required permissions, so the one that is not recommended in the app's declare permissions can be regarded as abnormal or does not support the permissions required by its declare functionalities. Therefore, given a target app a_i, its declared permissions are $DP_i =< DP_{i,1}, \ldots DP_{i,M} >$ and the recommended candidate permissions are $RP^*(a_i) =< p_{i,1}, p_{i,2}, \ldots, p_{i,N} >$. We can define the unexpected permissions $UP\ (a_i)$ as Eq. (13):

$$p_i \in UP^*(a_i) \leftrightarrow p_i \in DP(a_i) - RP^*(a_i) \cap DP(a_i) \tag{13}$$

where $^* \in \{B, M\}$. B represents the permissions recommended by the benign app, M represents the list of permissions recommended by the malicious app.

Also, the recommended permissions by the malicious set $RP^M(a_i)$ will contain not only the necessary permissions but also the risk permissions. On the contrary, most of the permissions recommended by the benign set are likely to be necessary. Thus, we define the risk permissions based on the permission gap $RP^M(a_i) - RP^M(a_i) \cap RP^B(a_i)$. The gap between the malicious sets and benign sets can be used to find the risk permissions, such as the permission *P*3 of target app in Fig. 2. Here, the risk permissions $RiP\ (a_i)$ can be formally defined as Eq. (14):

$$p_i \in RiP(a_i) \leftrightarrow p_i \in RP^M(a_i) - RP^M(a_i) \cap RP^B(a_i) \tag{14}$$

If the target app has an unexpected permission and also belongs to the risk permissions, we consider the app is a risky app, a risky app is defined as follow equation:

$$a_i \in Risk \leftrightarrow p_i \in RiP(a_i) \& p_i \in UP^B(a_i) \tag{15}$$

Finally, we can calculate the risk value as follow equation:

$$Risk(a_i) = \sum_{p_j \in \left(UP^B(a_i) \cap RiP(a_i)\right)} r\left(p_j\right) \tag{16}$$

Table 2
DataSet summary.

App Datasets	Number of Applications
Benign App	16343
Malicious App	524
App for Functionality Point-Permission Set Identification	32671
Permission Dataset	Number of Permissionss
System permissions	285

We calculate the risk level based on the permissions protection levels: normal and dangerous according to the Android permission mechanism. The scores of the permissions for two protection levels are assigned as 1 and 2 respectively. Here, $r(p_j)$ refers to the risk of permission based on its protection level.

4. Experiments

4.1. Experimental Setup

We use the app market dataset from Viennot et al. (2014). After processing the dataset, we choose 16,343 app with at least 100+ downloads and five stars in Google Play as the benign dataset finally. The malicious dataset was retrieved from VirusShare.[7] Since the VirusShare does not offer app descriptions, so we use the package identifier to map it into the one in the "app market" dataset. Finally, we find 524 matched items and regard them as the malicious app dataset. For the dataset of functionality point-permission set, we have selected 32,671 apps as the dataset to mine functionality point-permission relationships, which contains both the text information and APK files.

There are two kinds of permissions in the Android ecosystem: one is the system permission which is defined by the Android platform, and the other is the custom permission defined by developers themselves. In our experiment, we do not consider the custom permissions in the app because they are only defined and used by developer themselves. Especially, as the Android platform has many different versions, we take all permissions that have ever been defined, whatever used or not. Table 2 summarizes the dataset and Table 3 summarizes the experiment parameters.

4.2. Analysis the results for each phase

4.2.1. Over-declared permissions identification

Over-declaration permission identification process has been described in Section 3.2. In fact, the results by the LDA model often result in a sparse declaration functionality matrix. Therefore, in the specific implementation process, we made an inverted table according to the functionality relevance, and only the functionality-related app is calculated, which can greatly reduce the amount

[7] http://virusshare.com.

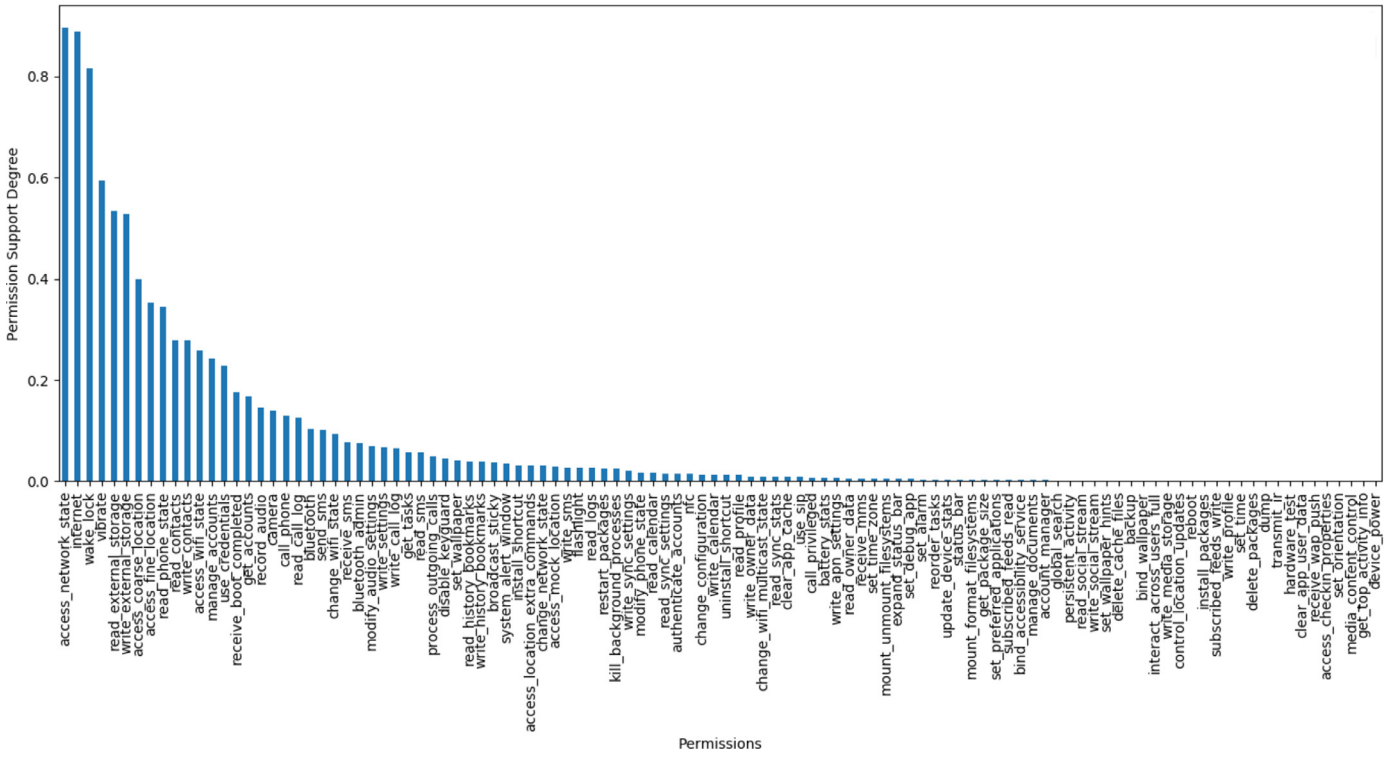

Fig. 6. An Example of Functionality point-permission set identification.

of calculation. On the other hand, considering that the number of benign app is much larger than the malicious app, so we select the threshold for the benign app as $T_b = 0.6$ and malicious app we have chosen the threshold $T_m = 0.4$ according to Huang et al. (2016). Furthermore, we use the method based on adaptive parameter in (Huang et al., 2016) to filter out all the permissions that are significantly higher than other permissions until the data is balanced. Finally, we remove the difference set recommended by the malicious and benign app in declared permissions, thus an over-declared permission identification process is completed.

4.2.2. Initial description-minimum permission set identification

In this phase, our goal is to identify the minimum permission set information corresponding to the app. We first randomly divide the benign apps into 5 parts, i.e., $N = 5$ in Section 3.3. Then we perform the initial *description-minimum permission set* identification method. In the actual experiments, we found that after 2 iterations, the permission data no longer changes, which means that the initial *description-minimum permission set* is obtained.

In addition, to illustrate our experimental results, we also provide an example of the minimum permission set identification as shown in Table 4. From that, we can see that an app in our benign dataset requires 10 permissions which are more than it actually needs. We identify the over-declared permissions and generate the initial minimum permission set. The words like "locat, video, music, north" in the functionality descriptions are related to 'location' or 'audio'. Thus, these permissions like RECORD_AUDIO and ACCESS_FINE_LOCATION enable the basic functionalities of the app and they are in the minimum permission set identified by MPDroid. On the contrast, we identify three permissions which are over-declared from the app descriptions: the functional descriptions do not include words related to 'call', 'sms' or 'camera'. This means that these three permissions are unrelated permissions different from the most apps with the similar description, which may lead to the leak of sensitive information.

In fact, the apps with a set of fixed functionalities will contain a minimum set of permissions corresponding to itself in theory. However, in the actual operation process, due to the inaccuracy of the app description data, the limitation of the amount of the app data, and the accuracy of the recommendation algorithm, it is impossible to find the minimum permission set. Through our method, we can continue to narrow the scope of the declare permissions without removing the reasonable permissions. This way, we can obtain we can get the most reasonable permission set of the app (i.e., the minimum permission set in this article). According to our iterative algorithm in Section 3.3, it does not delete reasonable permissions. The specific theoretical proof is as follows:

For a testing app, suppose its declared permission set is *DP*, and the permission set recommended by the bengin app is *BP*, and the permission set recommended by the maclicious app is *MP*, assume the minimum permissions set of the app is *MinP*. From a overall perspective: MPDroid will remove some permissions for each iteration. Thus, the number of permissions originally declared by the

Table 3
Experimental parameter descriptions.

Symbols	Descriptions
Topic Number	the numbers of the topic in LDA
Similarity threshold T_b, T_m	the threshold value of the similarity member selected employed for permission recommendation
Support degree threshold $\theta_{support}$	the support degree filter value in Functionality Point-Permission set identification phase
Test set ratio	the ratio of the test app set

app will gradually decrease. However, since $MinP \subseteq BP$, $MinP \subseteq MP$, there is $MP - BP \cap MinP = \emptyset$, i.e., the permissions that are removed each time are not the permissions in the minimum permission set, which can guarantee the lower bound of the final recommended result is greater than or equal to *MinP*.

From the perspective of the recommendation process: the difference set of $MP - BP$ will removed from the declaration permission set *DP* in each iteration, and *MP* includes a minimum set permissions and the risk permissions distinct from *BP*, i.e., the result of $MP - BP$ is risk permission (also called over-declared permission). In the *n*-th iteration, the recommendation result of the *MP* is unchanged, and *BP* removes some of the *MP* recommended permissions in the *n*-1 iterations. Thus, the permissions are gradually reduced compared to *n*-1 times, which also leads to fewer permissions being removed in *DP*. When the permission gap is reduced to 0 or close to 0, the permissions removed during the iteration process will be fewer, and the *DP* set tends to be stabilize. At this time, the *DP* set basically does not include the risk permissions and basically converges. Therefore, MPDroid does not incorrectly remove the permissions that the app actually uses.

In addition, the over-declared permission identification of the testing set app is based on difference set recommended by the malicious training set apps and the benign training set apps, as shown in phase 1 and phase 2 in Fig. 3. In this step, we mainly identify the over-declared permission for 80% of benign data (16,343) in training set, i.e., 13,075 apps, and the result is that 635 permissions were removed, involving 479 apps, accounting for 3.66% of the total apps. For the bengin test apps (3,268), the result is that 301 permissions were removed, involving 205 apps, accounting for 6.27% of the total benign testing apps. This also shows that there are also cases of over-declared permissions even for the benign app. However, for the malicious testing apps, 58 permissions were removed, involving 37 apps, accounting for 35.58% of the total malicious testing apps. These results show that the proportion of malicious apps which with over-declared permissions is much higher than that of a benign apps, which is in line with the real-world common sense.

4.2.3. Functionality point-permission set identification

The purpose of this phase is to mine the relationship between the functionality points obtained by LDA and the declared permissions directly. We calculate the relationship between app functionality points and permissions in real calls, and get the closest permission for each functionality point, then we get the actual permission set for each app. During the experiment, 32671 benign apps were selected, which contains the text information and APK file.

We map the functionality point information and the declared permission information of all apps, and implement the functionality point permission set mining according to Section 3.4. Then we obtain the support degree of each functionality point corresponding to the permission and arrange them from high to low. In terms of the code permission, we do the same thing. Then, we select the permission with the support degree greater than 0.1 and take the union of code permission and the declaration permission corresponding to the same functionality point. Meanwhile, we combine the results of the two to obtain the functionality point-permission set relationship. Finally, we inspect whether the initial minimum description permission set after the iterations in Section 3.3 exist the low support degree permissions. If it exists, remove these permissions, and thus we get the final description-minimum permission set corresponding to the app.

Fig. 6 shows the functionality point-permission set relationship corresponding to the app named screener2. The permission with the highest permission support degree is permission ACCESS_NETWORK_STATE, and its support degree is 0.89. It is the actual permissions required by the app, and the minimum permission support degree is the permission DEVICE_POWER, whose support degree is 0.0000146. This permission will be removed by our method since the permission support degree is too low. In this way, we have implemented the functionality point-permission set identification.

4.2.4. Permission-based risk evaluation

In order to prove the effectiveness of our method, we conduct an analysis comparison on our approach and the previous method (Huang et al., 2016) by using the following evaluation metrics, the metrics are defined as follows:

Mean Average Precision (MAP): *MAP* is a comprehensive evaluation of the accuracy for the recommended permissions. In our study, to test whether the recommended permission is in the actual declared permission list, we also consider the relative order of the recommendation results, the calculation formula is as follow:

$$MAP = \frac{\sum_{k=1}^{M} \frac{1}{N_k} \sum_{l=1}^{N_k} \frac{R_l}{l} I_l}{M} \tag{17}$$

where M represents the total number of app in the test set, N_k represents the number of permissions recommended for the k^{th} app. R_l indicates the number of permissions that really apply for the top l recommended permissions. $I_l = 1$ indicates that the l^{th} permission in the recommended ranking is really applied by the app, otherwise $I_l = 0$ indicates that the recommended permission does not been applied.

The ratio of the app with Unexpected Permissions (AUPR): The AUPR is defined as the percent of the app which has unexpected permissions in the testing dataset.

$$AUPR = \frac{1}{M} \sum_{i=1}^{M} K\left(UP^B(a_i)\right) \tag{18}$$

where if $|UP^B(a_i)| > 0$, then $K(a_i, Risk) = 1$, otherwise $K(a_i, Risk) = 0$.

Risk app Ratio (RAR): The RAR is defined as the ratio of risk app in the test datasets.

$$RAR = \frac{1}{M} \sum_{i=1}^{M} K(a_i, \text{Risk}) \tag{19}$$

where if $a_i \in \text{Ris}\,k$, then $K(a_i, Risk) = 1$, Otherwise $K(a_i, Risk) = 0$.

Table 4
An example of the initial description-minimum permission set.

Functional Description	Over-Declared Permission	Initial Minimum Permission Set
mission real music studio student love reach locat		ACCESS_COARSE_LOCATION
midtown studio offer instruct level excel musicianship	CALL_PHONE	ACCESS_FINE_LOCATION
piano voic drum involv north food spring station	AMERA	ACCESS_NETWORK_STATE
pedestrian access teacher cost lesson registr date event	SEND_SMSC	RECORD_AUDIO
program app latest watch tab wall share family class amp		WRITE_EXTERNAL_STORAGE
loop tube class channel latest music video		READ_PHONE_STATE

Table 5
Comparison of experimental results (20% benign app as a test set).

Approach	*AUPR*	*RAR*	*ARISK*	*MAP*	*NR*	*TRR*
MPDroid	**0.334**	**0.104**	**0.335**	**0.931**	**0.865**	**1.257**
SF	0.314	0.068	0.200	0.927	0.859	1.489
Improvement	6.4%	**52.9%**	**67.5%**	0.4%	0.6%	18.5%

Table 6
Comparison of experimental results (20% malicious app as a test set).

Approach	*AUPR*	*RAR*	*ARISK*	*MAP*	*NR*	*TRR*
MPDroid	**0.846**	**0.356**	**2.221**	**0.854**	**0.712**	**1.72**
SF	0.779	0.288	1.817	0.844	0.705	2.619
Improvement	10.9%	23.6%	22.2%	0.4%	0.1%	52.3%

Necessary recall (NR): The *NR* is used to measure the recall of our approach, it defined as follow:

$$NR(APP_t) = \frac{|APs\ in\ top\text{-}n\ |}{n} \tag{20}$$

where n is the number of the necessary permissions of APP_t, the AP_s are APP_t's necessary permissions, and $top-n$ is the set of *top-n* permissions returned by an approach. For a set of apps, *NR* is the mean of the *NR* values for all apps in it.

Total-recall ratio (TRR): The *TRR* is used to measure the effort our approach requires to achieve total recall, i.e., to recommend all the correct permissions for an app. It defined as follow:

$$\text{TRR}\ (APP_t) = \begin{cases} \frac{n_{mim}}{|AP_s|}, & \text{if } AP_s - RP_s = \emptyset \\ \frac{n_{all}}{|AP_s|}, & \text{otherwise} \end{cases} \tag{21}$$

where AP_s are APP_t's necessary permissions, and RP_s are the recommend permissions by our mehtod. In a nutshell, *TRR* measures that if we want to recall all the correct permissions of APP_t, how many permissions on average will be recommended by an approach for one correct permission. More specifically, if the approach can achieve total recall for APP_t, we simply compute the ratio of the minimal number of recommended permissions to achieve total recall, i.e., n_{mim}, and the number of APP_t's correct permissions, i.e., $|AP_s|$, APP_t's *TRR*. Otherwise, we penalize *TRR* by replacing $\boldsymbol{n}_{mim}$ with the total number of permissions captured by the training set, i.e., $\boldsymbol{n}_{all}$. The closer *TRR* is to one, the better it is. For a set of apps, *TRR* is the mean of the *TRR* values for all apps in it.

To evaluate the performance of MPDroid, we compare it with the SF method, which is the state-of-the-art bias-based recommendation method (Huang et al., 2016). In the experiments, we randomly selected 80% of the benign app as the training set to establish a description information and permissions mapping, as well as for the malicious app. The remaining 20% benign apps and 20% of the malicious apps constitute the test set. After using the *description-minimum permission set* identification algorithm to identify and remove over-declared permissions in benign app set, the corresponding *description-minimum permission set* relationship is obtained. Next, according to the description-permission mapping, the training set is used to recommend permissions to the test apps. We set the Topic Number=100, $T_b = 0.6$, $T_m = 0.4$, $\theta_{\text{support}} = 0.1$, and then calculate the *MAP, AUPR, RAR* and *ARISK* evaluation indicators. *ARISK* represents the average risk value for the test set app calculate by Eq. (16).

Tables 5 and 6 are the experiment results compared with SF method. Specifically, Table 5 is a benign apps that are used as a test set. Table 6 presents the results where malicious apps are used as the test set. The experimental results shows:

- 1) Under the experimental settings, MPDroid obtains higher *AUPR, RAR, ARISK, MAP* and *NR* values consistently, and *TRR* is the opposite, which indicates higher risk identification performance.
- 2) Not only for the malicious apps, some of the benign apps are overprivileged (i.e., over declared the unexpected or risk permissions). In generally, the malicious apps are generally much more likely to be overprivileged than benign apps.
- 3) For the benign apps, the proposed MPDroid outperforms on *RAR* and *ARISK* compared with SF by 52.9% and 67.5% respectively. This shows that MPDroid can provide developers and users with more reasonable permission configuration, and reducing the over privilege problem.
- 4) The *MAP* in SF and MPDroid changes little. This indicates that although the benign app has over-declared permissions, but the number of over-declared permissions is still small compared to the total number of permissions of the app. Generally, the over-declared permissions often appear in a few popular apps.

4.2.5. Studies on the parameters

In this Section, we discuss how the parameters impact the results in terms of *AUPR, RAR, ARISK, MAP* since they are the main goal of MPDroid.

(1) Impact of Number of Topic

MPDroid uses the LDA topic model to process the app description informations to obtain the functional feature vector. In order to study the impact of different topics on the final experimental results, we use the benign apps and the malicious apps as test sets respectively to evaluate the performance of MPDroid under different number of topics. In the experiments, the number of topics varies from 60 to 100 with a step value of 5, the test set ratio is set as 20%. In addition, according to Huang et al. (2016), we set the similarity threshold $T_b = 0.6$, $T_m = 0.4$ since it can obtain better results.

Figs. 7 and 8 show the experimental results of benign apps and malicious apps as the test set respectively, we can obtain that:

- whether it is a benign app or a malicious app as a test set, MPDroid can obtain better experimental results overall. This indicates that removing the over-declaration permission of the benign app can better identify over-declared permissions.
- the number of topics has a certain impact on each metric. For the benign app as the test set, the *RAR, ARISK* are higher when the number of topics is 60. However, for malicious app as the test set, the number of topics is 65. This indicates that the same number of topics has little impact on different test sets.
- For the *AUPR* and *MAP*, the number of topics will affect *AUPR* and *MAP* to a certain extent, but there is no fixed rule. For example, for the benign app as a testing set, when the number of topics is 60, the *MAP* is the lowest, and for the malicious app as the testing app set, the *MAP* achieve the lowest value is when the number of topics is 90.

(2) Impact of Support degree threshold $\theta_{support}$

The support degree threshold θ_{support} in Section 3.4 mainly considers the permissions required for the actual API call in the app, which can make the recommended permissions more accurate. To study the impact of the support degree threshold, we tested the benign app and the malicious app test set separately and compared the test results with different support degree thresholds. We set the initial value is 0.0.5, and then we vary the θ_{support} from 0.1 to 0.6 with a step value of 0.1, the similarity threshold $T_b = 0.6$, $T_m = 0.4$, and the test set ratio is 20%. In addition, since the support calculation only occurs in the MPDroid method, and the SF

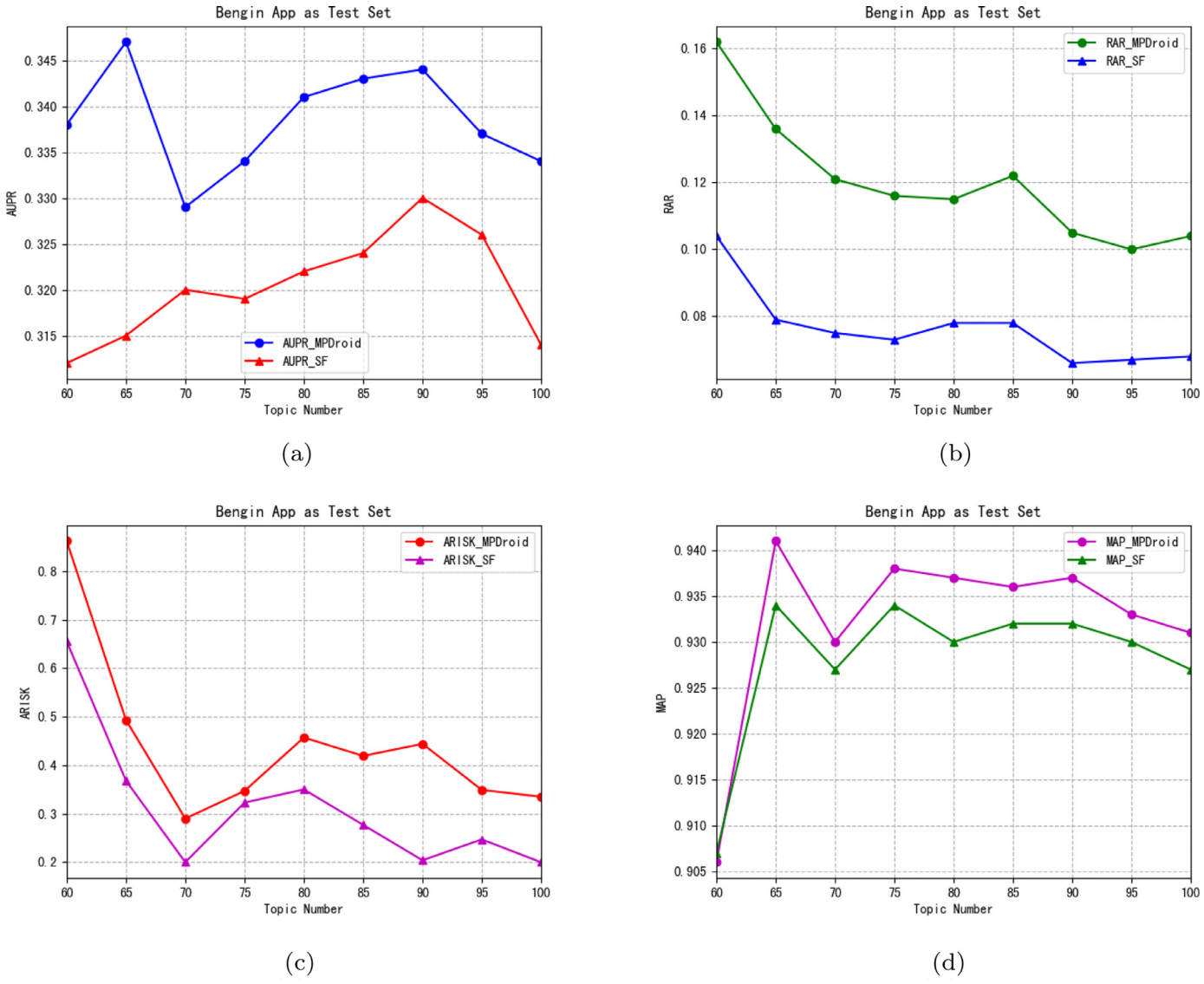

Fig. 7. Benign apps as the test set.

method is not affected by the parameter so that the result of SF is a horizontal line.

Figs. 9 and 10 show the experimental results of benign apps as the test set and malicious apps as the test set, respectively. we can know that:

- Regardless of whether it is a benign or malignant test set, when the support threshold vary from 0.1 to 0.4, *AUPR*, *RAR*, and *ARISK* change little. However, when the θ_{support} is greater than 0.4, the detection decreases, indicating that the risk permission is not filtered when the θ_{support} below 0.4. When θ_{support} is too high (greater than 0.4), the app risk permissions are filtered basically, the remaining permissions are likely to belong to the app itself and the detected risk is smaller.
- The recommendation accuracy rate of the benign test set is always greater than the malicious test set on the *MAP*. When the θ_{support} changes, the *MAP* of the benign test set changes little, but the malicious test set changes significantly. It indicats that the risk permission contained in the malicious app are more, and it will affect the accuracy of the recommendation.

(3) Impact of test set ratio

The test set ratio indicates the performance of MPDroid under different data scales. In order to study the effect of different test set ratios on experimental performance, we also tested the benign apps and the malicious apps test set separately, and compared the test results under different test set ratio. In the experiments, we vary the test set ratio from 10 to 40% with a step value of 5% and the similarity threshold is $T_b = 0.6$, $T_m = 0.4$, $\theta_{\text{support}} = 0.1$.

Figs. 11 and 12 show the experimental results with benign apps as the test set and malicious apps as the test set, respectively. It can be obtained that:

- For the benign app as the test set, the *AUPR*, *RAR*, *ARISK* value are the highest when the test set ratio is 10%, indicating that the test performance is the best. For the malicious test set, there is no fixed rule, indicating that different data sets and test set ratios will affect the effect of the model, and MPDroid is generally better than the SF method.
- For the *MAP*, the recommended effect is best when the benign test set ratio is 20%, and for the malicious test set app, the test set ratio is 25%. It shows that different test sets and test set ratios will affect the accuracy of the recommendations.

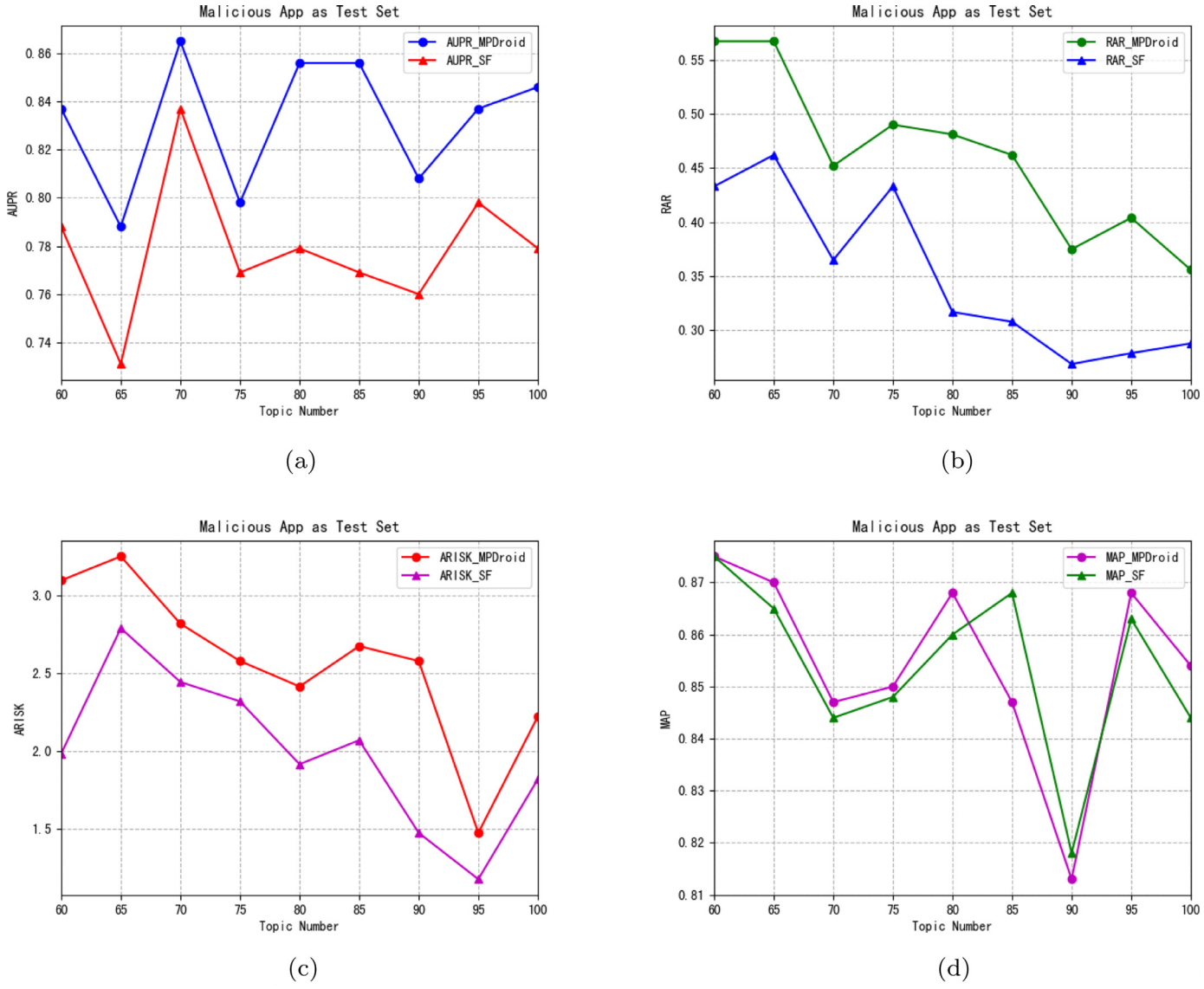

Fig. 8. Malicious apps as the test set.

4.3. Limitations

MPDroid has some limitations:

1. The implementation of MPDroid relies on the mapping the functionalities to permissions, but the functionality is only the features extracted from the app description. If the app description does not describe the functionalities of the app properly, the recommendation results will also be affected. To alleviate this issue, we use a large amount of app description information as a training set for training, which can largely ensure that the training model contains most of the description information, thus it can ensure the accuracy of subsequent permission recommendations. Meanwhile, we will also try to extract more granular and accurate functional features, such as adding other auxiliary information in future. In addition, we would add manual annotation if necessary.
2. Currently, the method of *description-minimum permission set* identification relies on the recommendation precision, and the establishment of model depends on data set. To alleviate this issue when we choose the dataset, the apps with at least 100+ downloads and five stars in Google Play are considered as the benign dataset, and the malicious app set we select from VirusShare website directly, which can slove the problem to some extent. In the future, we will enrich the mechanism for model building and further optimize the model to make it more generalized.
3. In the static analysis phase of the model, we cannot achieve the completely correct permissions to parse the app since some malicious apps re-packaging, or the interference of code mixing. Generally, some developers are not familiar with Android's business logic and lead to some useless code, result in matching too many permissions. Therefore, we will focus on improving static analysis and increasing the accuracy of the analysis in our follow-up work.

5. Related work

The spurt development of mobile Internet has promoted the security problem of mobile app to become a hot spot in the industry. Due to the imperfection of the Android permission mechanism, some irresponsible developers use the permissions of the Android app indiscriminately, and the user does not realize the importance

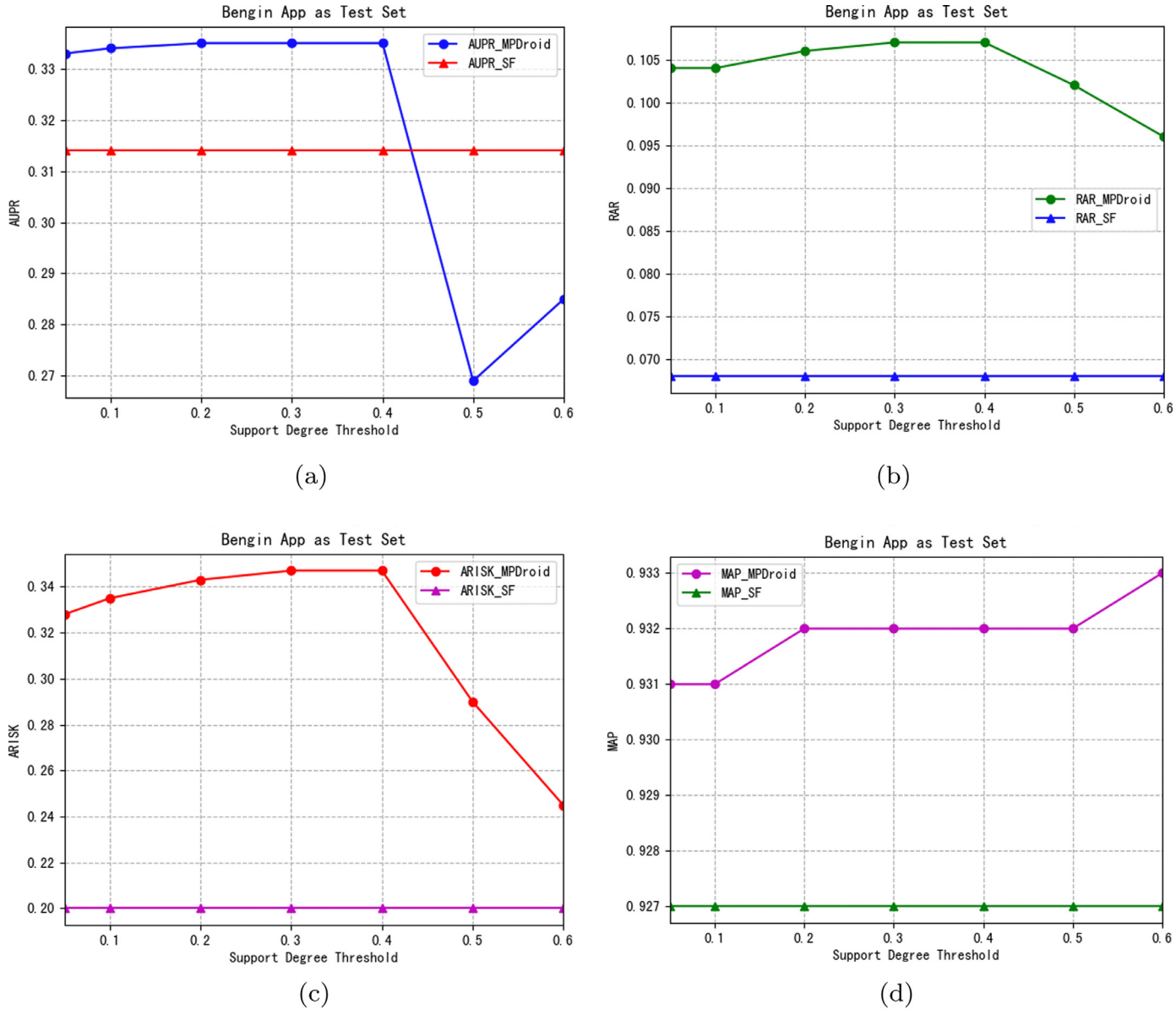

Fig. 9. Benign apps as the test set.

of the problem so that led to a series of privacy leaks and would pose significant risks to users and the entire mobile ecosystem.

One cornerstone of the Android security design is the permission system. An app must hold the permissions to successfully access the security and privacy critical methods. The permissions or the risk caused by the permissions have received a lot of attentions by many security types of research. Some studies are also dedicated to reminding users to potential risks through study permission dialogs, further or through improvements to enhance the permissions dialog (Felt et al., 2012b; Yu et al., 2016), but the impact of these methods on end users is still very small, users may still experience the risk behind it.

The above research is mainly based on the permission mechanism already contained in Android, and does not modify the underlying architecture of Android permissions. Their research content mainly focuses on two aspects. On the one hand, it is aimed at users, they try to make users understand the permissions by giving hints. For example, Kelley et al. (2013) designs the dialog content of the permission prompt box in more detail, reminds the user of the consequences when using specific permissions and what is the implied risk will cause, but they does not explain the details of the resources accessed and whether it may lead to the disclosure of some private information, Liu et al. (2016) propose a methodology PPA to learn privacy profiles for permission settings and leverage these profiles in a personalized privacy assistant that actively supports users in configuring their permission settings. On the other hand, it is mainly facing for developers, it remind the developers to strengthen their understanding of permissions, do not declare unnecessary or wrong permissions during the development process, but because Android development document do not have mature API descriptions, not only not enough details, but there may be some errors (Felt et al., 2012a). So in a summarize, no matter whether it is for users or developers, the issue of permissions has always been an urgent problem to be solved.

In order to solve the above problems, many studies decided to turn to use machine learning (Roy et al., 2015; Peng et al., 2018) or

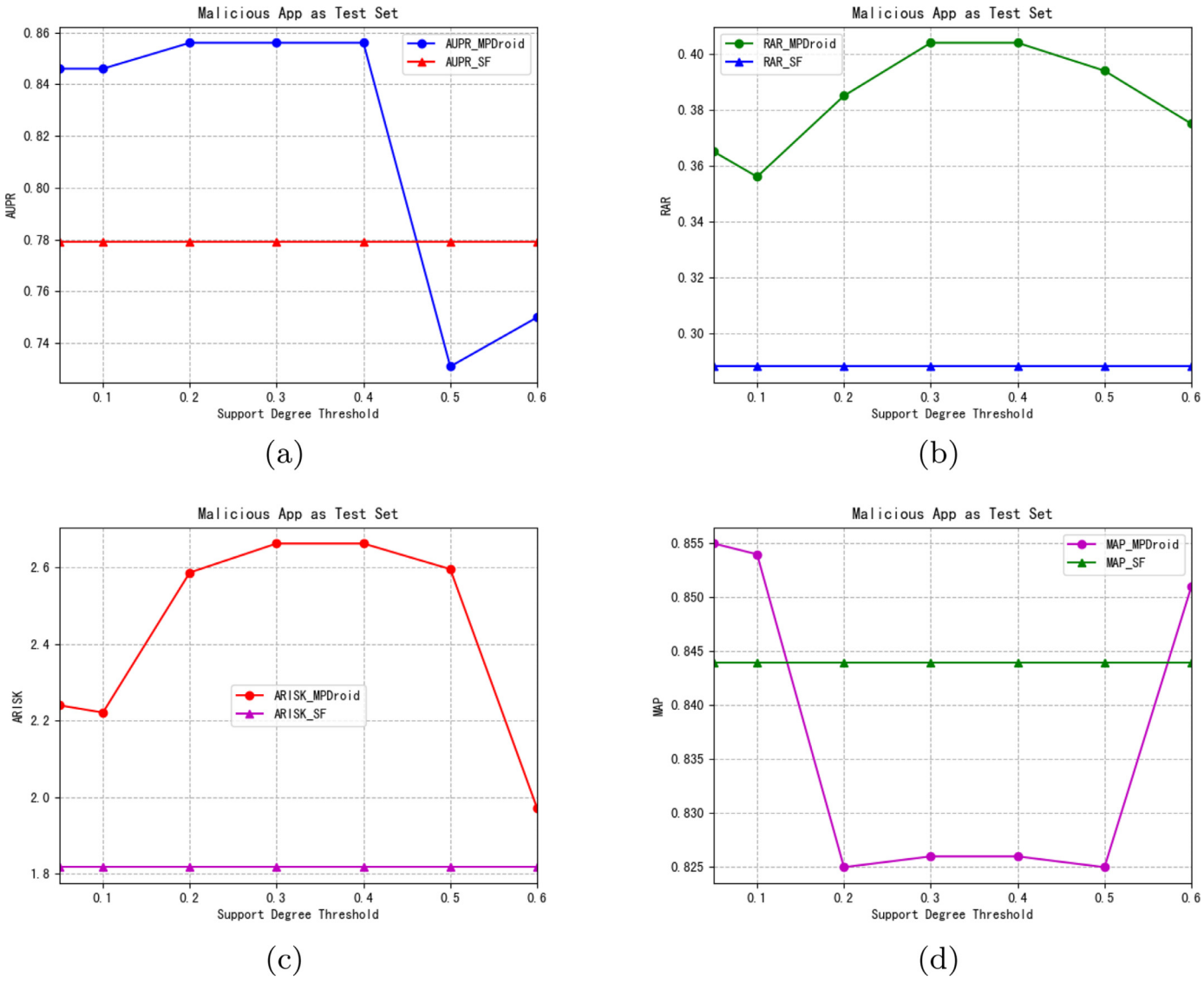

Fig. 10. Malicious apps as the test set.

recommended algorithms to solve the problem of permission redundancy, and then recommend reasonable permissions. For example, Peng et al. (2018) proposed a app risk score calculating method ARSM based on app-permission bipartite graph model which combine the correlation of apps' permissions and users' interests. Whyper Pandita et al. (2013) and AutoCog Qu et al. (2014) extracts various feature word combinations from the description information of the Android app as the app's declaration functionality and map them to the app declared permission, and through the gap between them to determine whether the declared permission meets the described functionality, and further determine the risk.

Besides, there are some other methods to extract a variety of feature vectors through code analysis to detect malicious apps. For example, Bartel et al. (2012) proposed a tool named "COPES" to detecting permission gaps using static analysis. It extracts from the Android framework bytecode a table that maps every method of the API to a set of permissions the method needs to be executed properly. Mujahid et al. (2018) implement an technique in a tool called PERMLYZER which automatically detects permission issues from apps APK to study the permission related issues in Wearable apps. More recently, some studies (Karim et al., 2016; Bao et al., 2016; Bao et al., 2017) also focus on recommend permissions by the used APIS, such as Karim et al. (2016) presented a tool named ApMiner which combines static analysis and association rule discovery to make app permission recommendations, and the results shown that ApMiner performs better than PScout and Androguard in terms of app permission recommendations. However, the average F1-score of APMiner is not sufficiently high (only approximately 55%) for it to be used in practice. Bao et al. (2016) propose an approach named APRecCF, which is based on collaborative filtering technique, the start point is that apps that use similar APIs often support similar features, it measures the similarity of two apps based on the APIs used by the apps and the result show that their approach achieve significant improvement in terms of the precision, recall, F1-score and MAP of the top-k results over Karim et al.'s approach. More over, based on (Bao et al., 2016), (Bao et al., 2017) also propose two novel approaches to realize permission recommendation, the first approach utilizes a collaborative filtering technique utilizes and the second approach recommends permissions based on a text mining technique that uses a naive Bayes multinomial classification algorithm,

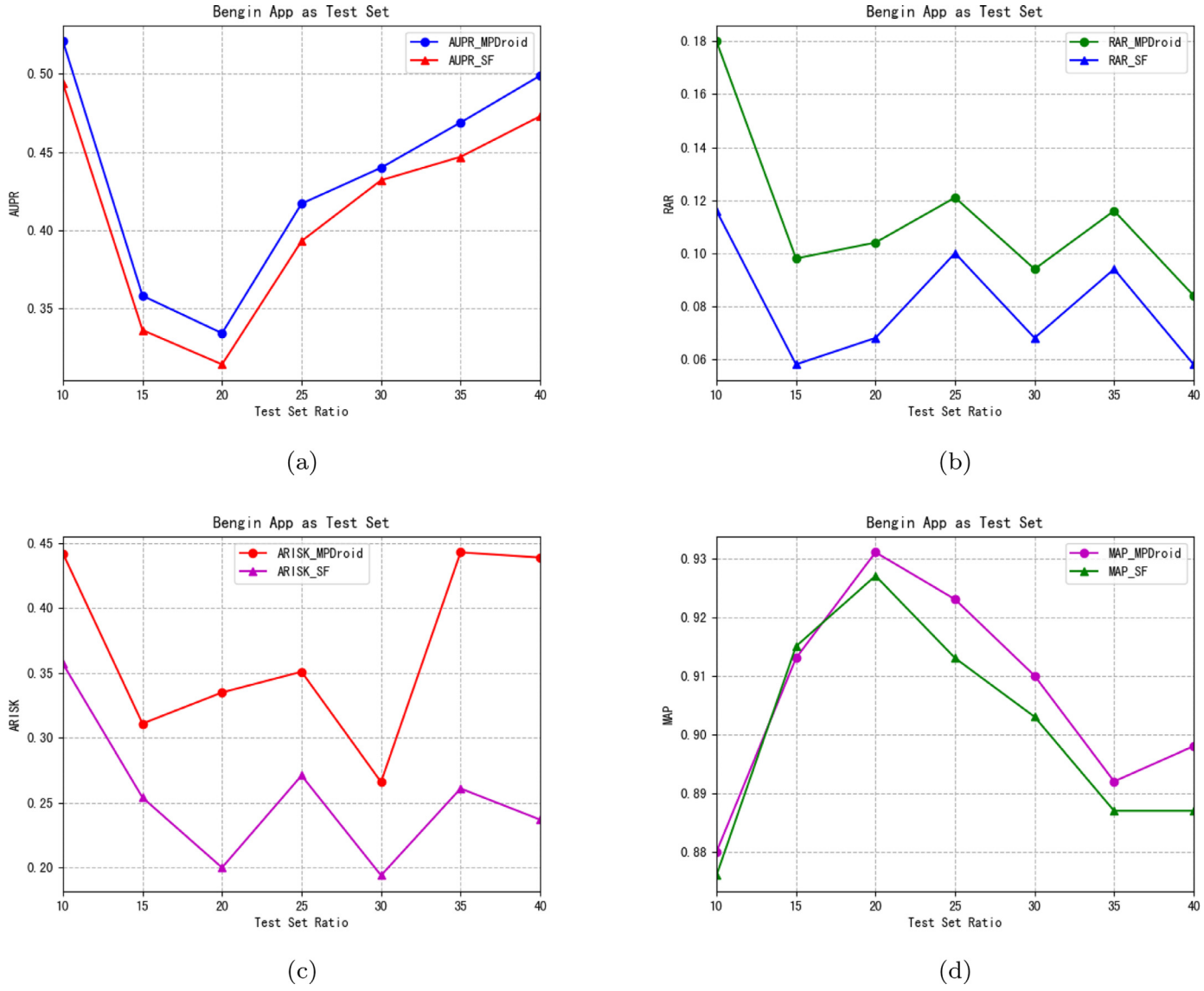

Fig. 11. Benign apps as the test set.

which show a better preference than the others. To best of our knowledge, the best work of permission recommendation is PerRec (Liu et al., 2019), which leverages mining-based techniques and data fusion methods to recommend permissions for given apps according to their used APIs and API descriptions, and their results show that PerRec significantly improves the state-of-the-art approaches APRecCF (Bao et al., 2017), APRecTEXT (Bao et al., 2017) and Axplorer (Backes et al., 2016).

The main disadvantage of the above methods is that the permissions declared by the app are treated as actually needed permissions by the app, but this is not always true, Android apps tend to be overprivileged. According to several studies, e.g., Felt et al. (2011a) and Yu et al. (2016), overprivileged apps are not the minority, but the majority. Therefore, it is necessary to identify the permissions that an app really needs, that is, its minimum permissions set as the permissions the app actually needs. Further, if given a functionality point, we can find a way to identify the minimum permission set corresponding to the functionality point, i.e., an app can declare the permissions to implement this functionality point. Then, it is possible to significantly improve the performance of over-declared permission detection so that to find a minimum set of permissions for the app of its describe information.

In order to achieve the goals, different from the above methods, at a higher level, our study is similar and creative to these former approaches. Our approach combines static analysis and collaborative filtering to identify the minimum permission set of Android apps. We measure the textual descriptions of the app in Google Play and the relevant permissions it required in a more accurate way.

6. Conclusion and futurework

Detecting the overprivileged permissions for the mobile applications is considered as a critical and valuable task to enhance the permission system for the mobile service ecosystem. Considering the miss match between the declared functionalities and the requested permissions to support the declared functionalities, we propose an iteration approach that combines static analysis and collaborative filtering to identify the minimum permission set for the mobile apps. The static analysis is used to

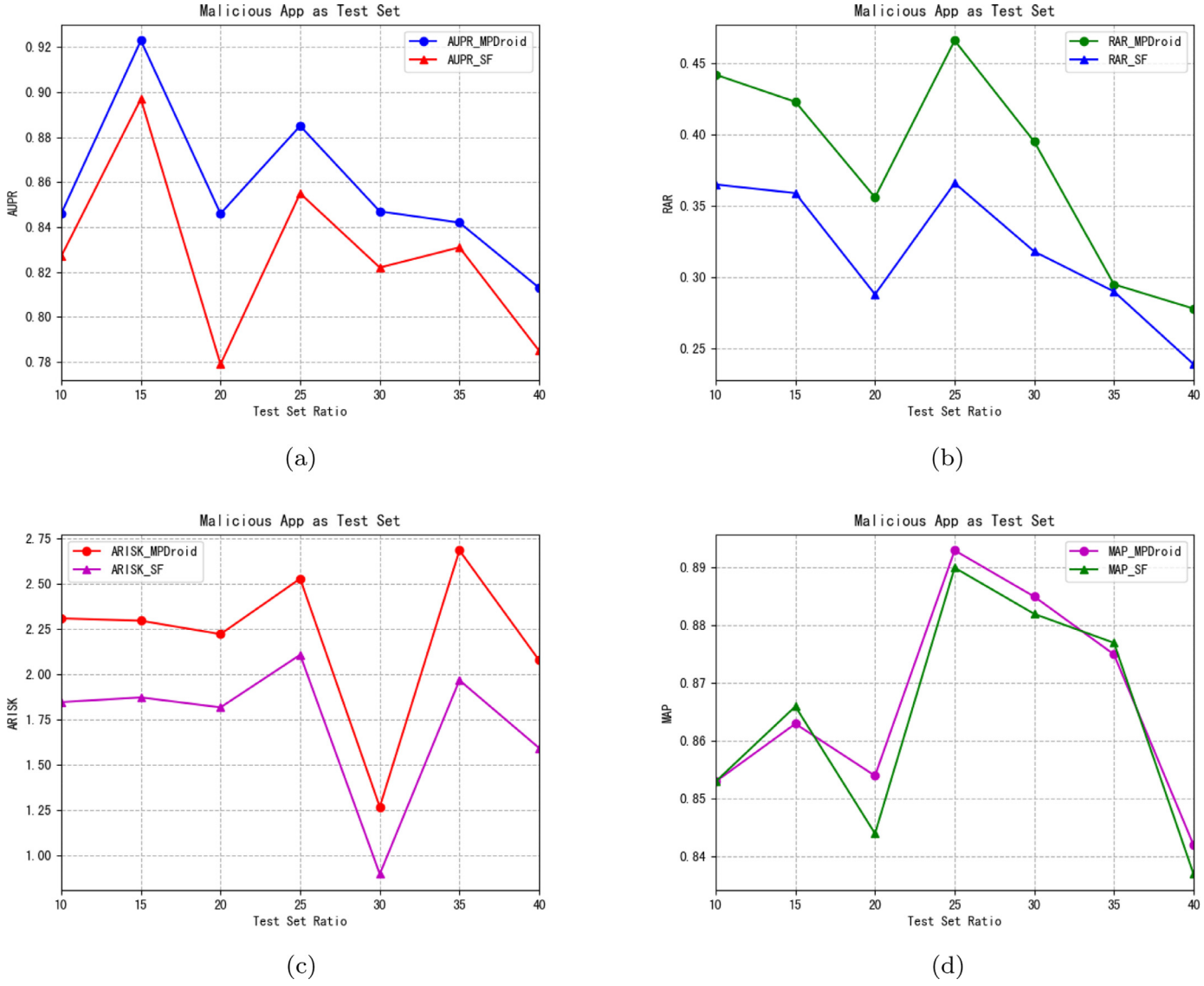

Fig. 12. Malicious apps as the test set.

achieve the real requested permissions for each mobile app, and a *description-minimum permission set* iteration algorithm based on collaborative filtering is developed to mine the relationships between the declared functionalities and the minimum requested permissions, so that we can detect the over-requested permissions and identify the high risk applications. Comparing with the previous state-of-the-art method, our MPDroid method can effectively identify the abnormal use of permissions and generate better permission recommendation configuration results, thereby reducing the problem of mismatch between functionalities and permissions. For the users, only need to have a description information of the app, so we can predict the permission required for the description information according to our model. In addition, this model also can be used to analyze the permissions of existing Android applications and evaluate the risks of the app.

For the future works, we will consider more declarative functionality information, such as user's comment information, app privacy policy information, and the classification information to improve the accuracy of the description functionality. At the same time, we will also add a new data cleaning mechanism to ensure the security of the declare permissions. In addition, since the *description-minimum permission set* identification module relies on datasets, we will further improve our model by focusing on related data cleaning and mining techniques, combining with the most advanced permission identification algorithms. We envision our method would help the developers to configure the requested permissions and design the declared functionality.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Jianmao Xiao: Conceptualization, Methodology, Writing - original draft. **Shizhan Chen:** Software, Validation, Funding acquisition. **Qiang He:** Writing - review & editing. **Zhiyong Feng:** Supervision, Resources. **Xiao Xue:** Data curation, Formal analysis.

Acknowledgment

This work is supported by the National Key R&D Program of China grant No.2017YFB1401201, the National Natural Science Foundation of China grant No. 61572350, the National Natural Science Key Foundation of China grant No. 61832014 and the Shenzhen Science and Technology Foundation (JCYJ20170816093943197).

References

Acar, Y., Backes, M., Bugiel, S., Fahl, S., McDaniel, P., Smith, M., 2016. SoK: lessons learned from android security research for appified software platforms. In: 2016 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 433–451.

Algarni, A., Malaiya, Y., 2014. Software vulnerability markets: Discoverers and buyers. Int. J. Comput. Inf. Sci. Eng. 8 (3), 71–81.

Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D., 2012. PScout: analyzing the android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, pp. 217–228.

Backes, M., Bugiel, S., Derr, E., McDaniel, P., Octeau, D., Weisgerber, S., 2016. On demystifying the android application framework: re-visiting android permission specification analysis. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 1101–1118.

Bao, L., Lo, D., Xia, X., Li, S., 2016. What permissions should this android app request? In: 2016 International Conference on Software Analysis, Testing and Evolution (SATE). IEEE, pp. 36–41.

Bao, L., Lo, D., Xia, X., Li, S., 2017. Automated android application permission recommendation. Sci. China Inf. Sci. 60 (9), 92110.

Bartel, A., Klein, J., Le Traon, Y., Monperrus, M., 2012. Automatically securing permission-based software by reducing the attack surface: an application to android. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 274–277.

Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D., 2011a. Android permissions demystified. In: Proceedings of the 18th ACM conference on Computer and Communications Security. ACM, pp. 627–638.

Felt, A.P., Egelman, S., Finifter, M., Akhawe, D., Wagner, D.A., et al., 2012a. How to ask for permission.. HotSec 12. 7–7

Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D., 2011b. A survey of mobile malware in the wild. In: Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices. ACM, pp. 3–14.

Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D., 2012b. Android permissions: user attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security. ACM, p. 3.

Goldberg, K., Roeder, T., Gupta, D., Perkins, C., 2001. Eigentaste: a constant time collaborative filtering algorithm. Inf. Retr. 4 (2), 133–151.

Gorla, A., Tavecchia, I., Gross, F., Zeller, A., 2014. Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 1025–1035.

Huang, K., Han, J., Chen, S., Feng, Z., 2016. A skewness-based framework for mobile app permission recommendation and risk evaluation. In: International Conference on Service-Oriented Computing. Springer, pp. 252–266.

Huang, K., Zhang, J., Tan, W., Feng, Z., 2015. An empirical analysis of contemporary android mobile vulnerability market. In: 2015 IEEE International Conference on Mobile Services. IEEE, pp. 182–189.

Jana, S., Erlingsson, Ú., Ion, I., 2015. Apples and oranges: detecting least-privilege violators with peer group analysis[J]. arXiv preprint arXiv:1510.07308.

Karim, M.Y., Kagdi, H., Di Penta, M., 2016. Mining android apps to recommend permissions. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1. IEEE, pp. 427–437.

Kelley, P.G., Cranor, L.F., Sadeh, N., 2013. Privacy as part of the app decision-making process. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 3393–3402.

Liu, B., Andersen, M.S., Schaub, F., Almuhimedi, H., Zhang, S.A., Sadeh, N., Agarwal, Y., Acquisti, A., 2016. Follow my recommendations: a personalized privacy assistant for mobile app permissions. In: Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016), pp. 27–41.

Liu, Z., Xia, X., Lo, D., Grundy, J., 2019. Automatic, highly accurate app permission recommendation. Autom. Softw. Eng. 1–34.

Mujahid, S., Abdalkareem, R., Shihab, E., 2018. Studying permission related issues in android wearable apps. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp. 345–356.

Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T., 2013. {WHYPER}: towards automating risk assessment of mobile applications. In: Presented as Part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13), pp. 527–542.

Peng, M., Zeng, G., Sun, Z., Huang, J., Wang, H., Tian, G., 2018. Personalized app recommendation based on app permissions. World Wide Web 21 (1), 89–104.

Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z., 2014. AutoCog: measuring the description-to-permission fidelity in android applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1354–1365.

Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V.P., Li, H., Guevara, N., 2015. Experimental study with real-world data for android app security analysis using machine learning. In: Proceedings of the 31st Annual Computer Security Applications Conference. ACM, pp. 81–90.

Viennot, N., Garcia, E., Nieh, J., 2014. A measurement study of google play. In: ACM SIGMETRICS Performance Evaluation Review, vol. 42. ACM, pp. 221–233.

Wang, J., Chen, Q., 2014. ASPG: generating android semantic permissions. In: 2014 IEEE 17th International Conference on Computational Science and Engineering. IEEE, pp. 591–598.

Wei, T.-E., Mao, C.-H., Jeng, A.B., Lee, H.-M., Wang, H.-T., Wu, D.-J., 2012a. Android malware detection via a latent network behavior analysis. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, pp. 1251–1258.

Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M., 2012b. Permission evolution in the android ecosystem. In: Proceedings of the 28th Annual Computer Security Applications Conference. ACM, pp. 31–40.

Wijesekera, P., Baokar, A., Hosseini, A., Egelman, S., Wagner, D., Beznosov, K., 2015. Android permissions remystified: A field study on contextual integrity. In: 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 499–514.

Yu, L., Luo, X., Qian, C., Wang, S., 2016. Revisiting the description-to-behavior fidelity in android applications. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1. IEEE, pp. 415–426.

Zhang, Y., Yang, M., Gu, G., Chen, H., 2016. Rethinking permission enforcement mechanism on mobile systems. IEEE Trans. Inf. ForensicsSecur. 11 (10), 2227–2240.

Zhou, Y., Wang, Z., Zhou, W., Jiang, X., Ning, P., 2012. Detecting malicious apps in official and alternative android markets. In: Proceedings of the Second ACM Conference on Data and Application Security and Privacy.

Jianmao Xiao Ph.D. student at the School of Computer Science and Technology, Tianjin University. His main research interests include service computing, software engineering. He has participated in a key research and development projects of science and technology department and a number of the National Natural Science Foundation of China.

Shizhan Chen Professor and Ph.D. supervisor in College of Intelligence and Computing, Tianjin University. His main research interests include service computing, Software ecosystem mining and analysis, etc.

Qiang He Senior Lecturer, School of Software and Electrical Engineering, Department of Computer Science and Software Engineering. His main research interests include Cloud Computing, Software Engineering.

Zhiyong Feng born in 1965. Professor and Ph.D. supervisor in the School of Software, Tianjin University. His main research interests include service computing, software engineering, Internet of things, etc.

Xiao Xue born in 1979. Professor in the School of Software, Tianjin University. Also adjunct professor in the School of Computer Science and Technology, Henan Polytechnic University. His main research interests include service computing, computational experiment, Internet of things, etc.