ARTICLE IN PRESS

Pervasive and Mobile Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

Android fine-grained permission control system with real-time expert recommendations

Bahman Rashidi [a,*], Carol Fung [a], Tam Vu [b]

[a] Virginia Commonwealth University, Richmond, VA 23284-3068, USA
[b] University of Colorado Denver, Denver, CO 80217-3364, USA

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Smartphone
Permission
Recommendation
Crowdsourcing

ABSTRACT

In current Android architecture design, users have to decide whether an app is safe to use or not. Expert users can make savvy decisions to prevent unnecessary privacy breach. However, inexperienced users may not be able to decide correctly. To assist inexperienced users to make a right permission granting decisions, we propose RecDroid. RecDroid is a crowdsourcing recommendation framework that facilitates a user-help-user environment regarding smartphone permission control. In this framework, the responses from expert users are aggregated and recommended to other users. We implement our prototype on Android platform and evaluated the system through simulation and real user study.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Mobile apps have brought huge impact to businesses, social, and lifestyle in recent years. Various app markets offer a wide range of apps from entertainment, business, to health care and social life. Android app markets, which share the largest user base, have gained a tremendous momentum over the years since its first launch in 2008. According to Gartner [1], 1.1 billion Android devices were shipped in 2014, marking its 80% mobile market share. The rise of Android phones brought the proliferation of Android apps, which provides an ever-growing application ecosystem.

As users rely more on mobile devices and apps, the privacy and security concerns become prominent. Malicious third-party apps not only could steal private information, such as contact list, text messages, online accounts, and the location from its users, but can also cause financial loss of the users by making secretive premium-rate phone calls and text messages [2]. At the same time, the rapid growth on the number of apps makes it impractical for app market places, such as Google App Store, to thoroughly verify the legitimacy of all apps. As a result, mobile users are left to decide whether an app is safe to use or not.

In the current Android architecture, users decide what resources are given to an app at installation time. However, this permission control mechanism has been proven to be ineffective to protect users from malicious apps. Study shows that more than 70% of smart phone apps request to collect data irrelevant to the main function of the app [3]. Among the 1.4 million apps in Google Play, a significant percentage of them have permissions going beyond the apps' intended use. The situation is even worse in the third-party markets which are also available to Android users. In addition, study shows that only a very small portion (3%) of users pay attention and make correct responses to the resource being requested on installation, since they tend to rush through to get to use the application. The current Android permission warnings do not help most users make correct security decisions [4].

* Corresponding author.
E-mail addresses: rashidib@vcu.edu (B. Rashidi), cfung@vcu.edu (C. Fung), tam.vu@ucdenver.edu (T. Vu).

http://dx.doi.org/10.1016/j.pmcj.2016.04.013
1574-1192/© 2016 Elsevier B.V. All rights reserved.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

Realizing these shortcomings in the current Android architecture, much efforts have been made to address the problems. Many resource management systems are proposed such as in [5–7]. Going down to the system level, L4Android [8] isolates smart phone OSes for different usage environments in different virtual machines (VMs). QUIRE [9] proposes a set of extensions addresses a form of attack, called resource confused deputy attacks, in Android. However, such approaches are not efficient since users are either not paying attention to permissions being requested or not aware of the permissions' implications. Hence, no mechanism that assumes high technical and security knowledge of users will be usable for a wide audience.

As pointed out in [10,11], the reasons for the ineffectiveness of the current permission control system include: (1) inexperienced users do not realize resource requests are irrelevant and could compromise their privacy, (2) users have the urge to use the app and may be obliged to trade their privacy for using the app. To address this problem, we propose RecDroid, a framework to assist mobile users to control of their resource usage and privacy through crowd sourcing. First, the framework allows users to use apps without granting all permissions. Second, RecDroid facilitates a user-help-user environment for permission control. Specifically, RecDroid allows users to install untrusted apps under a "*probation*" mode, while the trusted ones are installed in normal "*trusted*" mode. In probation mode, users make real-time resource granting decisions when apps are running. The framework facilitates a user-help-user environment, where expert users are identified and their decisions are recommended to inexperienced users.

To support such an environment, an effective expert users seeking is the major challenge. RecDroid evaluates the expertise levels of users using a Bayesian learning model. We evaluate the effectiveness of the model through simulation and survey data from real users. Compared with our previous work [12], the major contributions of this paper include:

- A comprehensive Android permission control framework to facilitate a user-help-user environment in terms of permission control.
- A Bayesian inference model to evaluate the expertise of users based on their historical responses to the permission requests.
- A low-risk recommendation algorithm which can help inexperienced users with permission control decision making.
- Implementing a central and extensible permission enforcing solution. Users do not need to root the device to use this solution.
- A prototype implementation of the system and real user evaluation on the usability of the system.

In the next section, we will provide an in-depth discussion of existing literature in resource management and permission controls for Android applications. We then provide an overview of RecDroid in Section 3, followed by the presentation of our algorithms in Section 4. Section 5 shows the implementation of our system on Android platform. Section 6 shows our evaluation results brought about by our real-user experiments. We discuss the possible threats and all attacks scenarios to the system in Section 7. Finally, we conclude this paper by a discussion of future directions and a conclusion of our work.

2. Related work

Due to its inherent constraints in resources, much effort has been done toward the principles and practices to manage resource usage [5–7,13–17] and privacy protection [18] of mobile applications. However, the most common practice for resource access management today is Mandatory Access Control (MAC) mechanism [19,20], which is found in API from major mobile players such as iPhone, Android, and Windows Phone. In such paradigm, resource access from apps needs to be granted by users. In Android, this is done through its *Static Permission Model* [21,22] where users need to grant all requested permissions on installation.

Exploring user perceptions of privacy on smartphones using crowdsourcing is being a trend in contemporary research on smartphone privacy protection. Our proposed work belongs to this category. Concurrent with our work, several other researchers have looked at the problem of Android permission management and especially with focus on crowdsourced privacy recommendation to users. ProtectMyPrivacy (PMP) is the closest work to RecDroid. Agarwal et al. proposed PMP, a crowdsourced recommendation system to collect and analyze users protection decisions on app granted permissions to help make privacy recommendations to iOS users [23]. They collect granting decisions from users and use them to generate recommendations. After filtering non-contributor (inactive) users using some predefined thresholds, if 55%–100% of valid decisions are to deny, the system recommends deny, while if only 0%–45% of valid decisions deny, the system will recommends grant. In PMP, they assume all users are on the same level of expertise and they all have the same level of influence on recommendation generation. However, in such crowdsourcing-based works which are user-oriented, majority users are inexperienced (novice) users. In PMP, since they use the simplest way (*majority opinion*) to generate recommendation, it gives high false positive and false negative in most of generated recommendations. In contrast, our previous work [12] proposed to use a simple threshold-based expert seeking algorithm to differentiate expert users from inexperience in order to utilize the experts' responses for recommendation generation. This paper further provides a Bayesian learning model for expert rating and uses a weighted voting model for recommendation decision making. This way, the generated recommendation is more likely to be correct and accurate. The recommendation algorithm can also use multiple properties as criteria to make recommendations. For example, Carullo et al. [24] proposed a system to generate recommendations (friendship suggestions) for online social networks based on multiple properties. Their proposed recommendation system is based on *triadic closure* and *homophily* properties.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

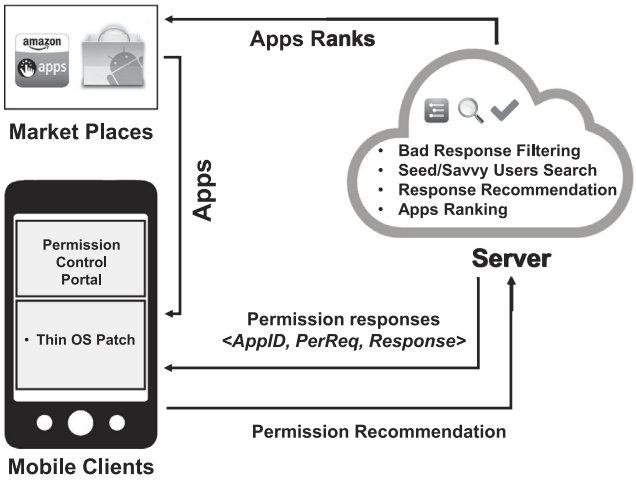

Fig. 1. RecDroid service overview.

Other works using crowdsourcing for permission control include the work from Ismail et al. [25], which is a crowdsourcing solution to find a minimal sets of permissions that will preserve the usability of the app for diverse users. They crowdsource the minimal sets of permissions through exploring all combinations of apps' requested permissions using users opinions on usability. This way, they can find the effective minimal permission set. Their proposed work has a few shortcomings. Repackaging apps for all possible permissions combinations and doing crowdsourcing, make the system infeasible. In addition, since they do not differentiate users' expertise levels, their generated permissions sets recommendations give high false positive and negative. Yang et al. propose a crowdsourcing based solution to help Android users better understand application permissions [26]. In their approach, collections of users of the same application use the system to help each other on permission understanding by sharing their permission reviews. Users leave comments on permissions and the system rank reviews and send the top ranked comments to users. App Ops [27], a feature in Android v4.3, allowed users to selectively disable permissions for apps on their phones. However, Google removed this feature in their next update, reporting that it was experimental and could cause apps to behave in unexpected ways.

3. System design

Our general approach is to build RecDroid with four functional processes, of which two are on mobile clients and the other two are on remote servers. In particular, RecDroid (1) collects users permission-request responses, (2) analyzes the responses to eliminate untruthful and bias responses, (3) suggests other users with low-risk responses to permission requests, and (4) ranks apps based on their security and privacy risk level inferred from users' responses. Fig. 1 shows an overview of RecDroid architecture, which is composed of a *thin OS patch* allowing mobile clients to automatically report users responses to and receive permission request response suggestions from a *RecDroid* service. The differentiating factor of RecDroid is the ability to seek expert user based on a small set of seed users (Section 4). In the rest of this section, we describe four key features of the RecDroid system.

Probation mode and requests handling: When a user downloads and installs an application, the installer needs to request permission to access resources on the device. Instead of sending requests to the Android system's legacy permission handler (e.g. Package Manager Service), the RecDroid handle the permission requests through the logistics illustrated in Fig. 2. On the installation of an app, RecDroid allows users to install the app on one of the two modes: *Probation* mode and *Trusted* mode. On *Probation* mode, RecDroid closely monitors the requests to access a list of user-defined critical permissions, such as location access, contact list access, and camera access, during the application execution. When those resources are requested, a dialog box will pop up to guide users to make customized decision on whether the access to those critical resources should be granted to the app or not. Otherwise, on *Trusted* mode, all requested permissions are permanently granted to the app.

Permission granting recommendation: To help inexperience users with their decisions, RecDroid also provide recommended response to the users (Fig. 4(c)). If a user chooses to deny a request, a dummy data or *void* will be returned to the application. For example, a denied GPS location request could be responded with a random location. The user decisions are recorded by the RecDroid client and sent to the RecDroid server for further analysis. After that, the requests are forwarded to legacy permission handler for book keeping and minimizing RecDroid's unexpected impact on legacy apps. It is important to note that this process only happens once when the app is installed. In a later phase when sufficient data is collected, and a security ranking of the app is available, RecDroid server can decide whether to pop up permission requests to users or automatically respond them based on prior knowledge. Therefore, RecDroid manages to achieve a balance between the fine-grained control and the usability of the system.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

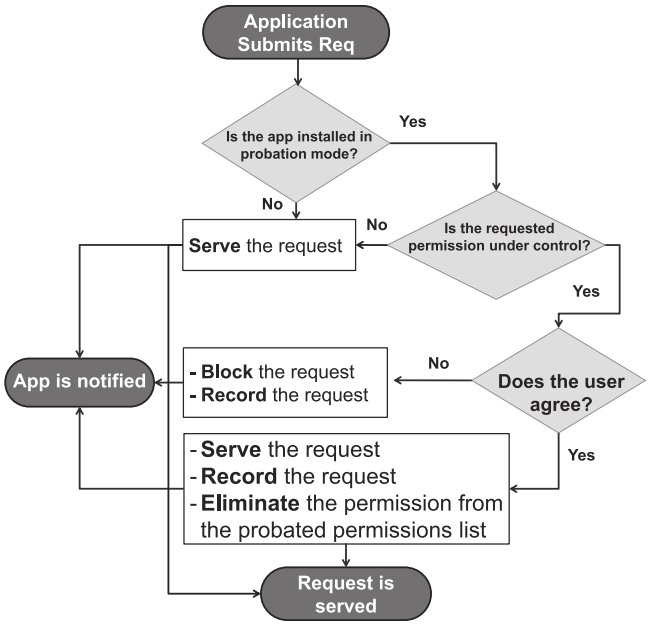

Fig. 2. Permission request flow in RecDroid.

Intercepting permission requests: To realize the realtime interaction between the system and users, we design a *Permission Control Portal* on the mobile devices to intercept apps' permission requests, records the requests, and collect users' response to the requests. Since intercepting permission requests requires OS level access, we create a small software patch to modify client's operating system. We investigated different potential approaches to perform OS modification and designed a solution that causes minimum impact to legacy apps and applicable to a broad range of OS versions, hardware platforms, and permission access models. Regarding the privacy concerns, RecDroid does not collect any sensitive information, the data it collects does not contain private information. In fact, the portal merely communicates three-tuple data in the form of *<Hashed AppID, Permission Request, User's responses>*.

Bootstrapping the service: To suggest plausible responses to users, RecDroid starts from a set of trusted *seed expert users* and make recommendation based on their responses. However, it is impractical to have our expert users to provide responses to all available apps on the market due to the extremely large volume. To address this scalability challenge, we propose a spanning algorithm that searches for *external expert users* based on the similarity of their responses to our set of internal experts, in combination with the user's accumulative reputation. Our recommendation for an app is based on the average of top N expert users in combination with the response that is selected by majority of participants. Having the same nature as the spanning algorithm described in [28], RecDroid spanning algorithm is sketched as follow.

4. RecDroid recommendation system design

In RecDroid, the responses to permission requests from all users are logged by a central server and they can be used to generate recommendation to un-savvy users to help them make right decisions to avoid unnecessary permission granting. For example, if a restaurant finding app is requesting for access to the users camera, then the request is suspicious and very likely will be declined by an expert user. The responses of expert users are then aggregated and the system would suggest other users of the same app not to accept the similar requests. However, how to find expert users and how to aggregate the responses from expert users are the focus of this section.

4.1. Rank RecDroid expert users

In this section, we investigate an algorithm to seek expert RecDroid users. Suppose we have a set of users U and among them set $E \subset U$ is a set of initial seed expert users. For instance, the security experts are employed by RecDroid to monitor the permission requests from apps. The seed experts respond to the permission requests from selected apps based on their professional judgment. Their responses are considered accurate and are used in the system as ground truth. However, due to limited budget and manpower, the number of apps that can be covered by seed experts is small, compared to the entire app base that RecDroid monitors. In this paper, we use set A to denote all apps that are monitored by RecDroid.

Suppose initially each of our seed expert users E have responded to a set of apps of their choice. The apps responded by an seed expert user $e \in E$ are denoted by $A(e)$. Since there may be multiple permission requests popped up during an app

usage, we use $R(a)$ to denote the set of requests the app $a \in A$ may have. We use R_e to denote all requests that are covered by RecDroid seed expert users, named *labeled requests*. Then we have,

$$R_e = \cup_{\forall e \in E} \cup_{\forall a \in A(e)} R(a). \tag{1}$$

How to determine whether a user is expert user or not? In our approach we propose a ranking algorithm to evaluate the expertise level of a user based on the ratio of correctness on his/her responses to app requests. Let p_i denote the probability that user i correctly responds to permission requests. Our mission is to estimate p_i based on the number of correct and incorrect responses that user i has responded in the past. Our approach is to observe all labeled requests that are independently responded (without recommendation) by the user, and compute the ranking of the user based on the number of correct/incorrect responses to those requests. For the convenience of presentation, we drop the subscript i in the rest of the notations.

We use notation α to represent the cumulative number of requests that are responded in consistent with the seed experts' advise, and β requests are responded opposite to the experts' advice (note that the labels from seed experts arrive later than the user's responses). Furthermore, we use a Bernoulli random variable $X \in \{0, 1\}$ to denote a user answers the permission requests correctly or not. Where $X = 1$ represents that user responds to a request correctly, vice verso. Therefore, we use $p = \mathbb{P}(X = 1)$ to denote the probability that a user responds to requests correctly. Since the Beta distribution is the conjugate prior probability distribution for the Bernoulli distribution, given a sequence of observations on X, a beta distribution can be used to model the distribution of p.

In Bayesian inference, the probabilities of Bernoulli variable given a sequence of observed outcomes of the random event can be represented by a beta distributions. The beta-family of probability density functions is a continuous family of functions indexed by the two parameters α and β, where they represent the accumulative observation of occurrence of outcome 1 and outcome 0, respectively. The beta PDF distribution can be written as:

$$f(p|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}. \tag{2}$$

The above can also be written as,

$$p \sim \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} y^{\alpha-1}(1-y)^{\beta-1}. \tag{3}$$

In our scenario, the parameters α and β are the accumulated number of observations that the user respond to the permission request correctly and incorrectly, respectively.

Let $x_n \in \{0, 1\}$ be the n_{th} observation in the past, where $n \in \mathbb{N}$. The accumulative observations of both correct and incorrect responses from a user after n observations can be written as,

$$\begin{aligned} \alpha_n &= \sum_{k=1}^{n} q^{n-k} x_k + q^n C_0 \\ &= x_n + q x_{n-1} + \cdots + q^{n-1} x_1 + q^n C_0 \end{aligned} \tag{4}$$

$$\begin{aligned} \beta_n &= \sum_{k=1}^{n} q^{n-k}(1-x_k) + q^n C_0 \\ &= (1-x_n) + q(1-x_{n-1}) + \cdots + q^{n-1}(1-x_1) + q^n C_0 \end{aligned} \tag{5}$$

where C_0 is a constant number denoting the initial believe of observations; $q \in [0, 1]$ is the remembering parameter which is used to discount the influence from past experience and therefore emphasize the importance of more recent observations.

Eqs. (4) and (5) can also be written into an iterative form as follows:

$$\alpha_0 = \beta_0 = C_0 \tag{6}$$

$$\alpha_n = x_n + q\alpha_{n-1} \tag{7}$$

$$\beta_n = (1-x_n) + q\beta_{n-1}. \tag{8}$$

Let random variable Y represent the possible value that the true expertise level of a user can be, then we have,

$$\mathbb{E}[Y] = \frac{\alpha}{\alpha+\beta} \tag{9}$$

$$\delta^2[Y] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \tag{10}$$

We use s to represent the expertise ranking of the user. Then we can compute s using the following formula,

$$\begin{aligned} s &= \max(0, \mathbb{E}[Y_i] - t\theta\delta[Y_i]) \\ &= \max\left(0, \frac{\alpha}{\alpha+\beta} - t\theta\sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}\right) \end{aligned} \tag{11}$$

where $t \in [0, \infty)$ represents a conservation factor where a higher value means our ranking mechanism is more conservative to less confident estimation. $\theta = \sqrt{\frac{2-q}{1-q}}$ is the normalization factor, which is to decouple the forgetting factor and expertise rating.

The higher ratio a user responds to permission requests correctly, the higher ranking score it receives through RecDroid system; the more samples the system knows about the user, the higher score it receives.

Given the ranking scores of users, we can use a simple threshold τ to identify expert users from novice users. That is, if a user i has $s_i \geq \tau$, then it is labeled as an expert user. The ranking scores will be used to make recommendations to other app users when permission requests pop up.

In the next subsection, we propose an algorithm to generate recommendation on app permission requests based on existing responses from other users who have used the same app.

4.2. Response aggregation through weighted voting

When a user receives a permission request from an app in probation mode, RecDroid system attempts to make a recommendation to the user regarding whether he/she should grant the request. If the app has been investigated by our seed expert users, then the response from the seed experts will be recommended to the user. However, due to the limitation of our seed experts, majority of apps may not be covered by seed experts. In this case, we aggregate the responses from other users and recommend the aggregated response if confidence level is high enough.

The proposed approach in this paper is called weighted voting. The voting process is divided into three steps: qualification, voting, and decision. The algorithm is described in Algorithm 1.

Algorithm 1 Weighted Voting for Recommendation Decision

```
1:  This algorithm is to decide whether to make recommendation, and what recommendation to make given the response from other regular users.
2:  Notations :
3:  M : the set of users who have responded to the permission question
4:  s_i : the ranking of the i_th user
5:  x_i : the response of the i_th user
6:  τ_e : the minimum required ranking score to be classified into expert users
7:  τ_d : the recommendation threshold
8:  a, b : the cumulative ballots for reception and rejection decision
9:  D_0 : the initial ballot count for both decisions
10: //initialize voting parameters
11: a = b = D_0
12: for each user u in M do
13:     if s_u > τ_e then
14:         //only qualified users responses are considered into the voting
15:         if x_u = 1 then
16:             a+ = s_u
17:         else
18:             b+ = s_u
19:         end if
20:     end if
21: end for
22: //decision making based on final ballots result
23: if a/(a+b) > 1 − τ_d then
24:     Recommend to accept the request
25: else if a/(a+b) < τ_d then
26:     Recommend to reject the request
27: else
28:     No recommendation
29: end if
```

In the qualification step (Line 12–21), only responses from qualified users are included into the voting process. Initially the ballot count for the acceptance decision and the rejection decision are equally initialized to D_0. Each qualified voter casts a ballot in the voting process. The weight of each casted ballot is the expertise ranking score of the voter. After the voting process finishes, the average ballot score is computed to make a final decision (Line 23–28). If the average ballot score exceeds a decision threshold, then corresponding recommendations are made. Otherwise, no recommendation is made.

In this decision making algorithm, users with higher expertise level impose higher impact to the final decisions. When there are not enough existing responses from expert users, or the responses from expert users conflict with each other, the system may choose not to make any recommendation.

5. Implementation

In this section we describe RecDroid's implementation process. The goal of RecDroid is to provide a platform for users to grant permissions to apps based on recommendations from expert users. To implement this system, we modified the permission management component of Android operating system. In addition, we provide users with an Android application to monitor and manage resource access permissions at fine-grain level. Fig. 3 depicts RecDroid's implementation architecture and modified and attached components to the OS.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

5.1. Android OS modification

To implement a real-time resource permission control, RecDroid is designed to capture all resource access requests (system calls) at run-time. To achieve this goal, we created a software patch to modify the Android operating system, where a few components and methods are changed according to our specific needs.

App installation mode: The core of the modification is the *Package Manager Service*, the place where the majority of modifications occur and plays an important role in the installation of apps and their requested permissions management. Installation is managed by the *PackageInstaller* activity, our changes have been to `InstallAppProgress.java` which is notified when an application install has completed. This is the best location to put a post install hook, to prompt the user if they would like the application placed into probation mode. This is an activity that is spawned off by the OS, by modifying this activity we were able to push an AlertDialog to the UI thread used by the system. The user is presented with two choices, trusted and probation (untrusted) installation. If the user selects trusted mode installation then the application is not managed by RecDroid, and no information is recorded about the application within RecDroid related repositories. If user select Probation mode, the application UID used by the android system, all permissions requested by the app that RecDroid is probating are recorded within the *Probated Apps Repository (PAR)* and *Request/Recommendation Repository (RRR)* datasets. Note that communication is obtained through the use of these repositories that all layers (framework and application) read and write from.

Algorithm 2 Permission Enforcing Flow

```
1: This algorithm is to decide whether to grant a requested permission to app or deny it
2: Notations :
3: PAR :Probated Apps Repository
4: RRR :Request/Recommendation Repository
5: flag : denotes that permission is probated
6: uid :Package identifier
7: p :Permission name (identifier)
8: r :user's response for a permission request
9: //initialize voting parameters
10: while (there is an incoming permission request) do
11:     Fetch UID's info from PAR
12:     if uid ∉ PAR then
13:         //grant the requested permission
14:     else
15:         Fetch apps' probated Ps from RRR
16:         if p's flag = True then
17:             //grant the requested permission
18:         else
19:             Prompting user through a popup
20:             if r = True then
21:                 //grant the requested permission and record the user's response
22:             else
23:                 //deny the request and record the user's response
24:             end if
25:         end if
26:     end if
27: end while
```

System calls and permission enforcement: Our implementation was designed to be extensible and generic. While our implementation requires multiple changes in one place, it does not require modifications on every permission request handler, as it was the case on some previous works, such as in MockDroid [29]. The modification is presented in the form of a patch, which can be executed from a user's space, making this technique easier to adopt.

In order to design an extensible and central permission enforcing point, we have modified the `enforce()` method of `ContextImpl.java` class of the context component of the Android. This is called whenever an application seeks to use some permissions that are not hardware related. When this method is called it is passed a UID and a permission name. Since UIDs 1000–9999 are reserved by the system and as the Android documentation [30] says, the range of UIDs, reserved for user applications is 10 000–99 999. We first check to see if the UID is above 10 000 for efficiency reasons a system calls happen thousands of times a second and if we reread from Probated Apps Repository each time then the whole system becomes interminably slow (high overhead). If the UID is at or below 10 000 then we let the call complete normally. If it is greater than 10 000 then we check the repository to see if the UID is present, and if it is, what value the flag associated with the passed in permission has. Algorithm 2 (Line 10–27) shows the flow of permission enforcing for any incoming permission request.

Communication with application layer: The main part of communication with application layer is users' interactions. The communication is made through prompting a user by generating a popup. Such transactions may include presenting user interfaces (popups and recommendations), receiving input from user interfaces (collecting responses to permission requests from users), managing (defining policies) requested permissions from users, app installation mode selection by users. We discuss each of these transactions and how they can be managed in more details in following subsection.

5.2. Permission control portal

The users of RecDroid have the option to install apps under a *probation mode*. We use the app "Line" (a popular chat application) as an example. The first screenshot (Fig. 4(a)) displays two options when installing the app on the smartphone.

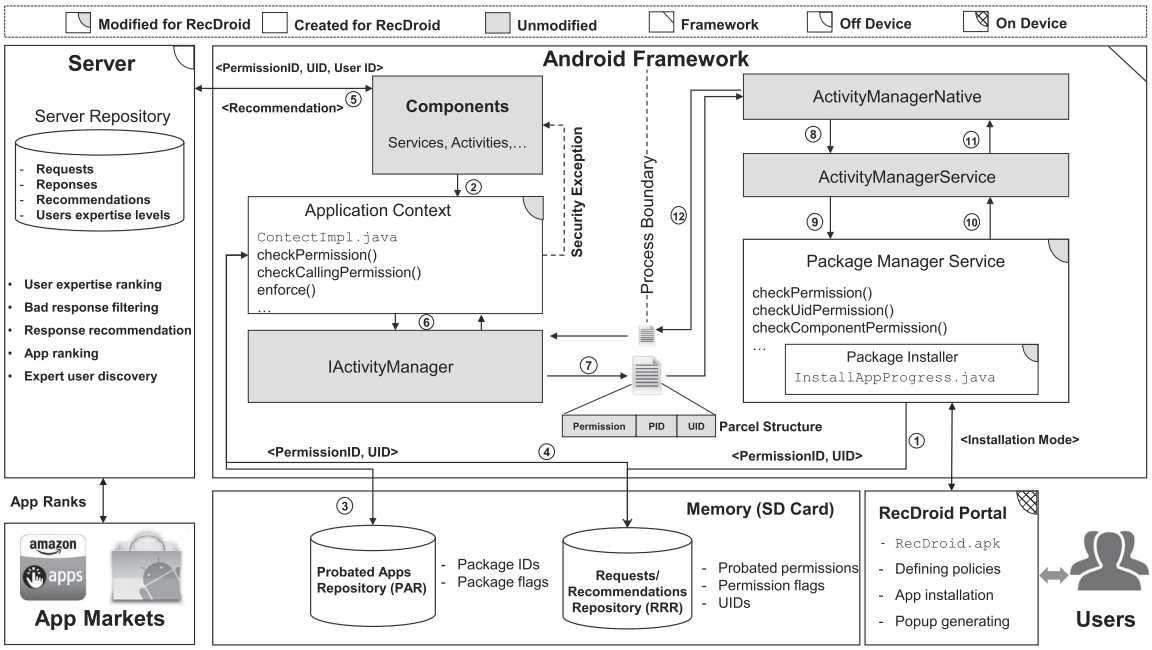

Fig. 3. RecDroid implementation architecture overview.

They can be either *probation mode* or *trusted mode*. If the user selects the probation mode, the application will be added to a list of monitored apps on the phone. On the other hand, if the user selects the trusted mode, the application will be installed with all requested permissions granted.

For each installed app, users can use the RecDroid application to view a list of applications which have been installed under the probation mode. The built OS includes `RecDroid.apk` as a pre-installed app. If the user click on an application in the list, a set of its requested permissions (see Fig. 4(b)) will be listed and users can select some of them to be monitored resources. By default all sensitive resources are listed to be monitored.

If an app is installed under the probation mode, whenever the app requests to access to a sensitive resource under monitoring, the user will be informed through a pop-up (Fig. 4(c)). In addition, the system also gives a recommendation with a level of confidence to assist users to make decisions. If the user chooses to agree, the request of the application will be served; otherwise the request will be blocked.

5.3. RecDroid recommendation server

Recording the users' responses and providing decision recommendations to users are essential to RecDroid. In our system, we create a remote server to record the responses on an online server and also compute recommendations according to the recorded responses from users. The RecDroid clients request recommendations from the server when needed.

6. Evaluation results

In this section we try to present the RecDroid evaluation results. In order to have a comprehensive evaluation on the accuracy, reliability, effectiveness, and system's performance, we conducted a set of real-user and simulation experiments.

6.1. Evaluation through simulation

For a comprehensive evaluation of the RecDroid system, we chose to use simulation to evaluate the expert rating and recommendation algorithms.

6.1.1. Simulation setup

As a proof of concept we set up a RecDroid users profile to be a set of 100 users consisting of three different levels of expertise. Note that the expertise we refer here is the probability that a user answers permission requests correctly (a.k.a. consistent with standard answers). Among the 100 users, 40% are with a high level of expertise (0.9), 30% are with a medium level of expertise (0.5), and the remaining 30% are with a low level of expertise (0.1). Unless particularly specified in the experiments, we fix the number of requests answered by users to 100.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

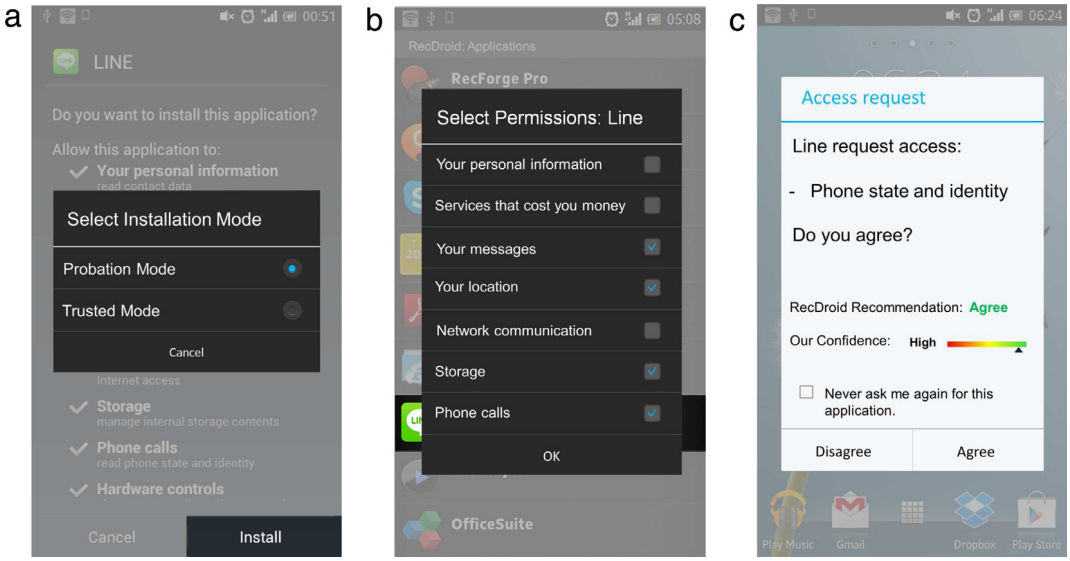

Fig. 4. An example of RecDroid online app: (a) probation and trusted installation modes; (b) Users pick which critical resources to be monitored; (c) Pop-up for permission granting with suggestion from RecDroid and its confidence.

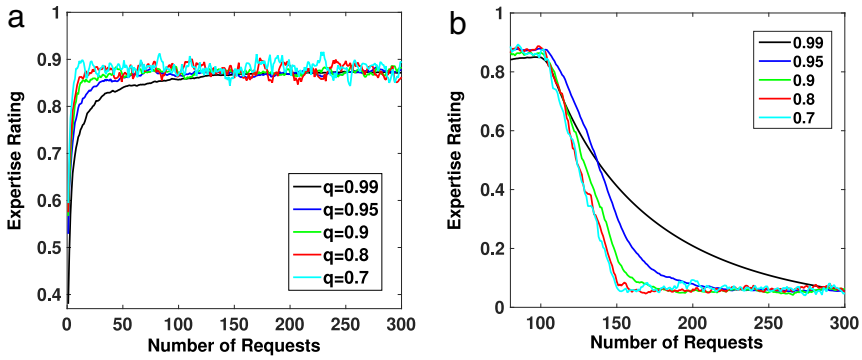

Fig. 5. Forgetting and Conservative factor: (a) Expertise level of users with different forgetting factor; (b) 100% of requests are answered.

Our simulation environment is MATLAB 2013 on a Windows machine with 2.5 GHz Intel Core2 Duo and 4G RAM. All experimental results are based on an average of 100 repeated runs with different random seeds.

6.1.2. Expertise rating and the impact of parameters

The remembering parameter q (Eq. (4)) and conservation factor t (Eq. (11)) are two essential parameters that RecDroid uses for user expertise rating. In this experiment we study the impact from the two factors and determine the parameter choices for the rest of the experiments.

In the first experiment, we track the RecDroid expertise rating of a high expertise (0.9) with the number of labeled requests they have answered under different remembering factor settings. In the second experiment, we deliberately configure the user so it immediately turns to be dishonest and gives opposite responses after the 100 honest requests.

From Fig. 5(a) we can see that with higher q setting, the curves are smoother. This is because a high q means the expertise rating largely depends on past accumulation, which brings stableness to expertise rating. However, from Fig. 5(b) we can see that high q also represents less flexibility to sudden change. To leverage the pros and cons, we decide to fix $q = 0.9$ in the rest of this paper.

In the third experiment, we track the expertise rating of high, medium, and low expertise users after 100 labeled requests under different t setting. Fig. 6(a) shows that with higher t setting, the rating of all users are lower. We chose a moderate setting $t = 0.1$ in the rest of this paper.

Fig. 6(b)–(d) shows the expertise rating of the three types of users (with expertise 0.1, 0.5, and 0.9). The blue boxes represents the central 50% of the expertise rating data while the red bars are the medium values of the samples. The vertical whiskers indicate the range of all data except outliers, which are represented by red crosses.

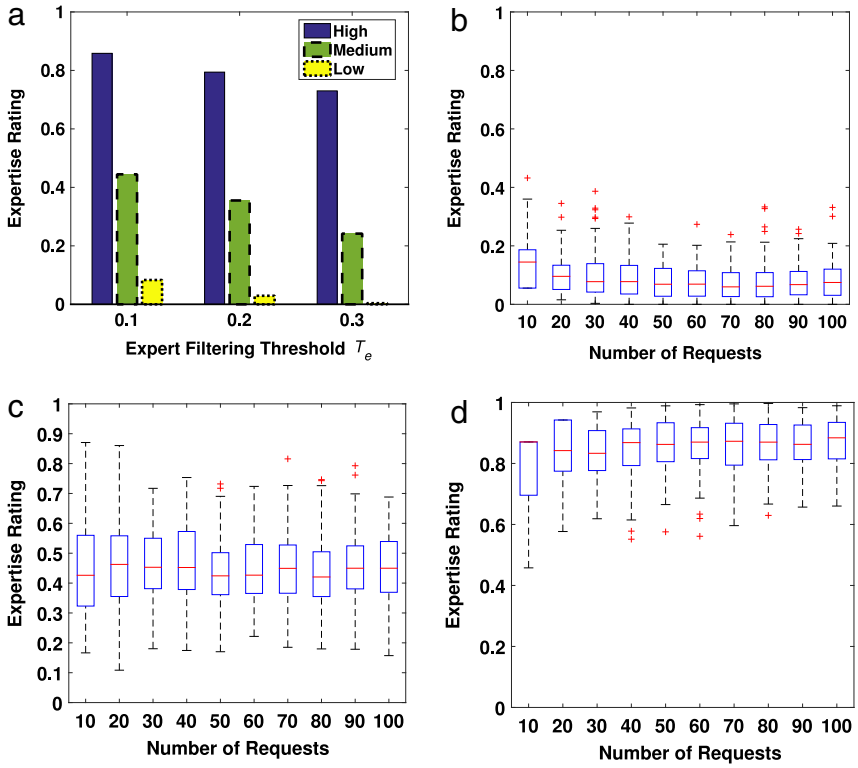

Fig. 6. Expertise ratings of: (a) Expertise level of users for different conservation factors; (b) Low expertise nodes; (c) Medium expertise nodes; (d) High expertise nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.1.3. Coverage and accuracy of RecDroid recommendation

In this experiment we evaluate the performance of RedDroid recommendation by measuring its recommendation coverage and accuracy. We define *coverage* to be the percentage of the requests that RecDroid decides to give recommendation to users given the existing responses from users participating RecDroid. We define *accuracy* to be the percentage of correct recommendation that RecDroid makes. Note that if a request is covered by a seed expert, then RecDroid always recommend the response from the seed expert.

In the first experiment we investigate the scenario that 100 requests receive responses from all 100 users, except seed experts. RecDroid uses Algorithm 1 to determine whether to make a recommendation to new users or not and what recommendation it should make. Note that we assume all 100 users have received expert rating scores previously. We plot in Fig. 7(a) and (b) the percentage of requests (among 100) that RecDroid decides to make recommendation and percentage correct recommendations that RecDroid makes, under different τ_e and τ_d settings. We can see that with higher τ_d (which means wider acceptance range for the recommending decision, see Algorithm 1), the coverage increases while the accuracy decreases. This is because the more selective RecDroid is regarding the voting score results, the higher accuracy it achieves and less voting results will be qualified for recommendation. We also notice that the accuracy increases with experts filtering threshold τ_e. However, with very low or very high τ_e, the coverage is low. This is because when all users are included in the decision process, the conflict of responses among users leads to low voting score and therefore RecDroid is less likely to make recommendations. On the other side, high filtering threshold causes few or no users are qualified to voting process, which also leads to no recommendation.

Fig. 7(c) depicts the result on percentage of qualified users of the three types under different τ_e setting. We can see that with higher expert filtering threshold τ_e, less users can be involved in the decision process as described in Algorithm 1. Also the involving rate of low expertise users is lower than high expertise users.

Finally, we simulate a scenario that no users are rating previously and all users have responded to 100 request. As a coordinator of the RedDroid system, we hire a seed expert to respond to some application requests as ground truth, which will then be used to rate the expertise of other users. Regarding the requests not covered by the seed expert, RecDroid may provide recommendation based on the response from other users. We study the coverage rate by seed expert and the percentage of requests that are covered by RecDroid. As shown in Fig. 7(d), the overall RecDroid recommendation rate increases with coverage rate from the seed expert. The linear line represents the coverage rate from the seed user. The difference between the overall coverage and seed expert coverage is called the *bonus coverage*. Higher bonus coverage represents a higher utilization of Recdroid. From an economic point of view, if we consider the coverage of seed expert brings

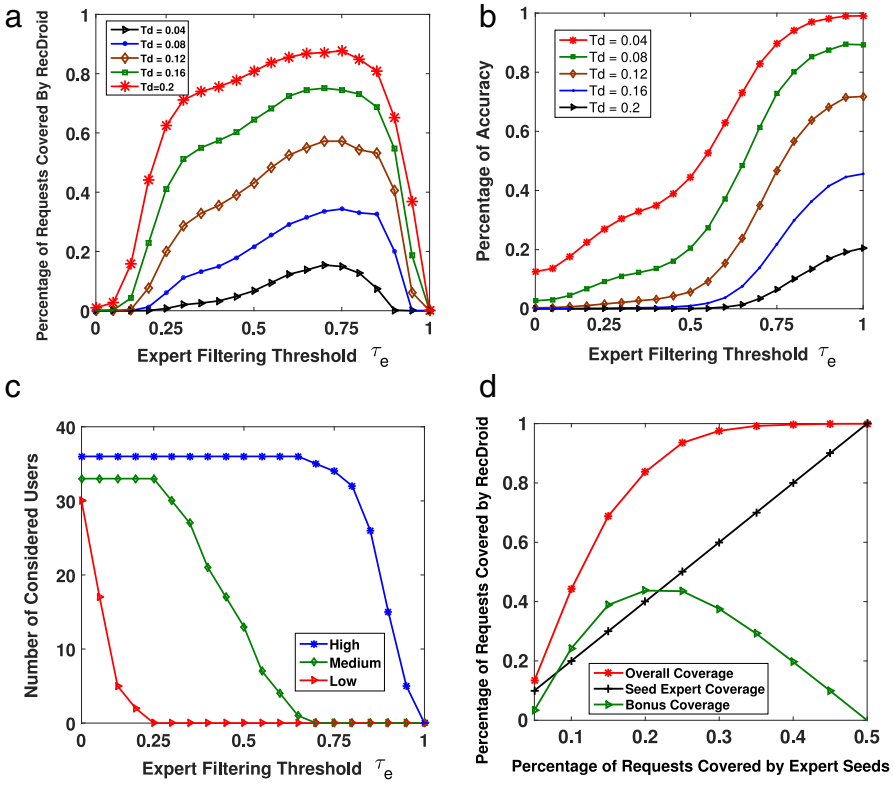

Fig. 7. Coverage and accuracy: (a) the percentage of requests that RecDroid makes recommendation; (b) the percentage correct recommendations that RecDroid makes; (c) the percentage of users who pass expert filtering and participate into recommendation voting.

Table 1
Diversity of participants.

Educational level	High-school	Undergrad	Grad	Unspecified
Number of users	16	20	28	40

cost to the RecDroid coordinator (since the seed expert is hired), then the bonus coverage brings saving to the coordinator. The decision makers can choose the optimal seed coverage based on its optimal profit. Note that in the above experiment, the values of simulation parameters are $\tau_e = 0.5$, $\tau_d = 0.2$, $q = 0.9$, and $t = 0.1$.

6.2. Real-user evaluation

In this section we describe our experiments evaluating the proposed framework through real user participation. The experiment is based on real users and devices. More specifically, we conducted a set of experiments to measure the accuracy, reliability, and effectiveness in both Ease-of-Use and applicability of the system.

6.2.1. Experiment setup

The set up for the experiments can be summarized as follows:

Participants: We recruited 104 real users to participate in our experiments. To introduce diversity of the participants, we have users from different educational levels (High-School, Undergraduate, Graduate and Unspecified). Table 1 shows the division of participants in our experiments by educational levels.

Applications: We selected 12 applications to be evaluated in our experiments. The selected apps include 6 "*trusted*" apps (top ranked) and 6 *aggressive* (not ranked) apps. We define *trusted* apps to be those developed by trusted developers such as *Instagram* (social network), *Weather Chanel* (weather category), etc. We define *aggressive* apps to be those that request excessive permissions (irrelevant requests) that they do not actually need to perform the main function of the app. The apps request various resources including Internet communication, location, camera, storage (photos/media/files), SMS service, and user's contacts. We selected apps from different app categories such as communication, social network, finance, weather, music-audio, card game, and arcade. In each of these categories we downloaded a pair (trusted and aggressive) of apps. There are 72 permission requests in total.

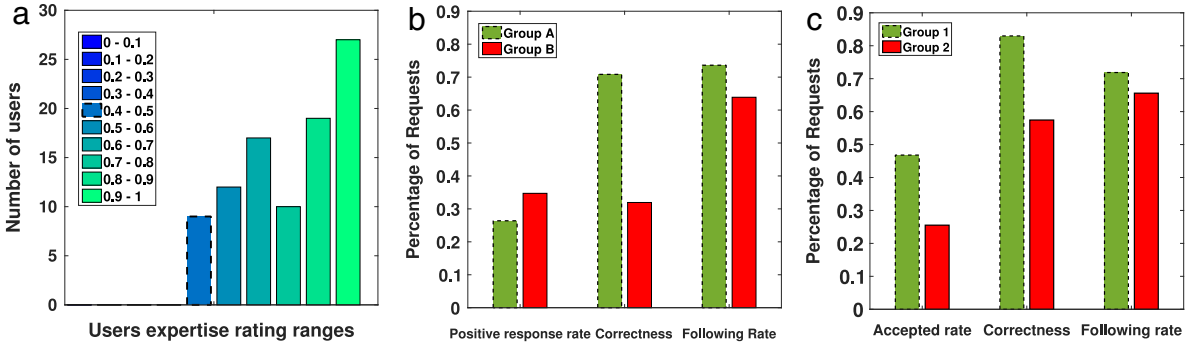

Fig. 8. Correctness and recommendation positive response/following rates: (a) expertise ratings of participants; (b) users responses analysis; (c) applications' permission requests analysis.

Devices and OS: To prepare for the experiment, we built a customized Android platform (Android 4.3, Jelly Bean) equipped with RecDroid system. We have used 4 LG Nexus 4 devices running RecDroid system.

Server: RecDroid records all responses from users and store them on an online server hosted by the Computing Center of Virginia Commonwealth University. The server is implemented on a LAMP stack and uses CentOS, Apache, SQLite3, and the latest version of php.

All users responded the permission requests from all apps independently. Among the 72 requests, only 30 requests have RecDroid recommendation available. Those recommendations are fabricated on purpose and may be incorrect to test how likely users will follow those recommendations blindly. We collected the responses from all users for analysis. The ground truth of those permission requests were provided by our seed experts.

6.2.2. Correctness and recommendation granting/following rates

In this subsection, we present the results of participants and apps analysis for correctness, positive response rate, and recommendation following rate. Before presenting the results, we show an overall view of the expertise level of users. Fig. 8(a) shows the expertise rating of all participants based on our collected data and ground truth. After applying expertise rating algorithm on recorded responses, the expertise rating results (without expertise ranking normalization) of users starts from 0.4 (lowest) to 0.95 (highest). Considering the calculated users' expertise ratings, we group them into three groups, users with low (<0.5), medium (0.5–0.7) and high (>0.7) level of expertise.

Regarding the participants' educational level, we have not found any significant correlation between expertise level and the educational level of users.

Fig. 8(b) shows the users responses divided by users' demographic groups. We divided users into two groups. Group A are users who are more experienced (expertise rating higher than 0.8) than the users in group B (expertise rating below 0.5). The *correctness rate* is the percentage of correctly responded request, and the *following rate* is the percentage of requests which followed the RecDroid recommendations. We can see that both correctness rate and following rate of users in group A are higher than users in group B.

Fig. 8(c) shows the applications' permission requests/responses analysis results grouped by types of apps. As it is shown, we divided apps into two different groups: trusted group (group 1) and aggressive group (group 2). The accepted rate of requested permission for apps in group 1 is higher than apps in group 2. The correctness rate of responses to permission requests and recommendation following rate for apps in group 1 are also higher than group 1.

It is worth noting that, regardless of the apps' trustworthiness and users' background, the recommendation following rate is almost the same in any case.

6.2.3. Coverage and accuracy of recommendations

In this experiment we evaluate the performance of RedDroid recommendation by measuring its recommendation coverage and accuracy. In order to measure the accuracy of the generated recommendations, we did an experiment with different τ_e between 0 and 1 (after expertise ranking normalization) and $\tau_d = 0.4$ settings. Fig. 9(a) shows the results of this experiment. From this experiment, we notice that percentage of wrong recommendations decreases by increasing τ_e. The percentage of requests that RecDroid does not generate recommendation also increases with expertise filtering threshold τ_e.

Fig. 9(b) also depicts the result on percentage of qualified users of the three types under different τ_e setting (0–1 with 0.05 step). We can see that with higher expert filtering threshold τ_e, less users can be involved in the decision process as described in Algorithm 1. Also the involving rate of low expertise users is lower than high expertise users.

Finally, we did an experiment to measure the utilization rate of the RecDroid. In this experiment, users are not rated previously and all users have responded to 72 requests. As we described before, we as seed experts, responded to all application' permission requests as ground truth, which will then be used to rate the expertise of other users. Regarding

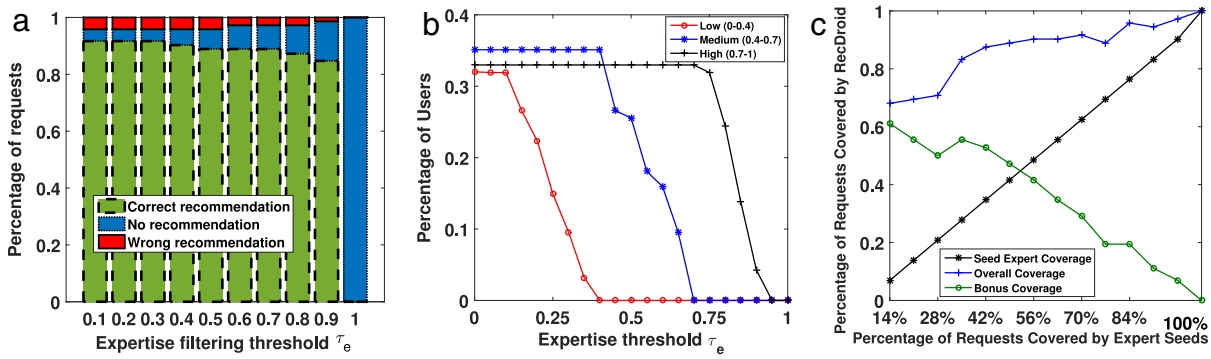

Fig. 9. Coverage and accuracy: (a) accuracy of prepared recommendations under different τ_e and a fixed $\tau_d = 0.4$ setting; (b) Qualified users of the three types under different τ_e setting; (c) Coverage of overall requests vs. coverage of seed experts.

Table 2
Users' opinion data and device security.

	Secure	Neutral	Not secure	Total
Concerned	28%	23%	15%	66%
Neutral	11%	7%	3%	21%
Not concerned	3%	7%	3%	13%
Total	42%	37%	21%	100%

Table 3
RecDroid's trustworthiness and ease-of-use.

	Low	Medium	High
Ease-of-use	2%	14%	84%
Trustworthiness	8%	20%	72%

the requests not covered by the seed expert, RecDroid may provide recommendation based on the response from other users. We study the coverage rate by seed expert and the percentage of requests that are covered by RecDroid. The range of covered permission requests by seed experts users starts from 10 (14%) and end to 72 (100%) requests.

As shown in Fig. 9(c), the overall RecDroid recommendation rate increases with coverage rate from the seed expert. The linear line represents the coverage rate from the seed user. The difference between the overall coverage and seed expert coverage is called the *bonus coverage*. Higher bonus coverage represents a higher utilization of RecDroid. From an economic point of view, if we consider the coverage of seed expert brings cost to the RecDroid coordinator (since the seed expert is hired), then the bonus coverage brings saving to the coordinator. The decision makers can choose the optimal seed coverage based on its optimal profit. Note that in the above experiment, the values of experiment parameters are $\tau_e = 0.5$, $\tau_d = 0.2$, $q = 86\%$, and $t = 0.1$.

6.3. Survey statistics

Along with the real data collection, we also conducted a survey to measure different factors of the RecDroid system. In addition to participating in our test, we asked all the 100 users to fill a questionnaire and answer to some objective multiple-choice questions. Table 2 shows that the majority percentage (66%) people are concerned with their data privacy on mobile phones, while a large percentage (42%) people believe the smart phone they are using is secure.

We also surveyed the Ease-of-Use and trustworthiness of the RecDroid system. Table 3 shows that the majority (84%) of the participants believed that RecDroid is easy to use. 72% of the users think that RecDroid's recommendations are reliable.

7. Threats and defense

Recommender systems have become popular in recent years, and have been applied in a variety of applications. One widely-used approach for the recommendation system design is the collaborative filtering [31]. Collaborative filtering methods are based on the collection and analysis of a large amount of information on users' behaviors, activities or preferences. In the cases where anyone can provide recommendations, people may give a large number of positive recommendations to their own entities and negative recommendations to their competitors. There is a typical attack to recommender systems called *Shiling* attack [32] that affects the quality of generated recommendations through injecting fake profiles. Therefore, securing the recommendation systems against existing attacks is a major challenges. For example,

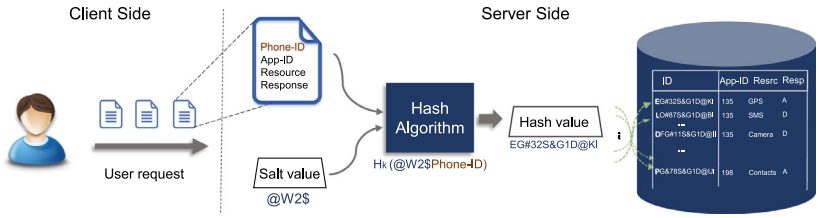

Fig. 10. Use one-way ID hashing to protect users' privacy.

Maw [33] proposed a secure personalized recommendation system in mobile tourism domain. An algorithm based on *significant* and *trust* weighting was used to detect or prevent the Shiling attack.

Although the purpose of RecDroid is to protect inexperienced smartphone users from being attacked by malicious apps, RecDroid itself may be the target of attacks. In this subsection, we discuss a few potential threats to RecDroid that we can foresee at this stage. We then show that through integrating strategical defensive design into RecDroid framework, we can detect, deter, or mitigate the threats addressed. We also address the privacy concerns which may rise from RecDroid users and we show that our privacy-aware data collection design can reduce this concern to a minimum.

Injecting false recommendations: One of the main important threats to the system is the injection of false responses to mislead the recommendation system. For example, during the external expert users seeking process, malicious users/attackers behave well in order to be rated as expert users. After being chosen as expert users, they turn around and suggest dishonest recommendations to mislead the recommendation system.

We have investigated this potential threat and developed a multi-agent game theory model [34,35] to study the gain and loss of malicious user and the RecDroid defense system. We derived a system configuration to discourage rational attackers to launch such attacks. Through the proposed game model we analyzed the interaction (request/response) between RecDroid users and RecDroid system using a static Bayesian game formulation. In the game, both the RecDroid system and attackers choose the best response strategy to maximize their expected payoff and we studied the Nash Equilibria of the game. We also find the strategies that RecDroid can use to disincentivize attackers in the system, since they have no gain by attacking the system.

Bot users: Bot users are fake users which are set up and controlled by attackers to fulfill some specific purpose. For example, the vendor of a malicious app may create many "expert" RecDroid users who will be honest when responding to other applications except the particular app owned by their "master". Since RecDroid heavily relies on the responses from expert users, many dishonest expert users may misguide RecDroid into providing wrong recommendations if not detected and handled properly. How to detect those bot users and mitigate their impact is an important problem for RecDroid.

In order to address this issue, we have developed a clustering-based method called BotTracer [36] to finding groups of bot users controlled by the same masters, which can be used to detect bot users with high reputation scores. The key part of the proposed method is to map the users into a graph based on their similarity (*features*) and apply a clustering algorithm to group users together. Specifically, we found that malicious users controlled by the same master may:(i) downloaded and responded to the malicious app at as soon as the app is available. (ii) have unusual high overlap on the apps they installed and responded since they are from the same master. (iii) respond to the malicious app differently than benign expert users. We also proposed a *weighted* distance function, which weights the proposed features and aggregates them into the similarity between users.

The privacy concern: RecDroid is a crowdsourcing-based solution and seeking expert users in the network is an important task. RecDroid collects permission responses from all participating users to discover experts. To protect the privacy of users, we design a privacy-aware data collection mechanism that uses *hashing and salting* method (Fig. 10) to protect the true identity of the users. The salt is randomly generated upon installation. Note that this mechanism provides *double-blind* protection, which means attacker who successfully attacked the database will not be able to reverse the function to find out the real phone ID or even verify whether an given phone ID is in the database. Therefore, the identity of the users are well-protected, and the mechanism does not compromise the usability of the collected data.

Bad-mouthing attacks: In this type of attack, attackers collude to give negative feedback on a user in order to lower or destroy its reputation. As we discussed in the above paragraph, since we use a hashing and salting method to protect the true identity of the users, users do not have access to others' identities. In addition, our trust model is not based on users' feedback on others and users do not give feedback. Therefore, this attack does not apply to RecDroid.

Threats to RecDroid server: Attackers may also attack the RecDroid server by launching (Distributed) Denial of Service (DoS) attack against the server. For example, an attacker can use multiple controlled RecDroid smartphones to flood the server with a large volume of requests from those clients. Attackers can also create a large volume of fake handshaking requests to the RecDroid server as if they are from new RecDroid clients. If the attack is successful, the server will be unavailable to normal RecDroid clients and will not be able to provide recommendations to the clients. To address these issue, our strategies are: (1) RecDroid server should be distributed and deployed in cloud to gain resistance to bandwidth attack. (2) Data-rate from a single RecDroid client should be limited to limit the impact from a single attacker. (3) Limit one client per IP address to deter sybil attack. (4) To deter syn-flood type DDoS attack, we can deploy anti-DDoS solutions [37–41] on the RecDroid server. However, the study of those solutions is not the focus of our proposed research.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013

8. Discussion

In this section, we discuss some potential issues in RecDroid and our future plan to build such system.

Ads libraries. A potential issue is that, although the main app is legitimate to access some resource, the integrated ads from third parties may also gain access to those granted resources. How to separate the permission granting for the app and its integrated ads is another interesting problem to solve.

Context awareness: The initial premise for this framework is that permission request should be handled differently on different context. For example, a request for a user's current location should be granted if the user is issuing a location-based query (e.g. What are the best French restaurants around here?"), while the same request should be denied if the user is listening to music at the time of request. The current design of RecDroid, however, does not capture the context at which the permission is requested. Rather than tracking a single behavioral dimension, RecDroid needs to monitor multiple dimensions. For example, Carullo et al. [42] proposed FeelTrust, a trustworthy communications model that helps people decide whether or not an interaction with a stranger is desirable. They have modeled both the *context specific* and *multi-faceted* properties to calculate the trust and reputation levels of users based on multiple dimensions (behaviors). Hence, the most logical next step is to integrate context information to the system so that the server can make informed recommendations not only depending the relevant context. It is important to note that enhancing the service with context, however, has both pros and cons. On one hand, it explains the behavior of the application in the associated context, hence improving the analytic process of the recommendation and ranking system. On the other hand, collecting more context information could also raise privacy concern. We are investigating how to integrating context in without compromising users' privacy.

User participation: Promoting the participation is a key to the success of our system and other crowdsourcing-based works as well. To attract more people to participate the RecDroid crowdsourcing, it is important to offer them some incentive. Designing effective incentive mechanism is a challenge across many social computing contexts, such as attracting crowdworkers and sustaining online contributions.

Due to the importance of this issue, much effort has been done toward incentivizing crowdsourcing to study research participation [43–46]. Ghosh and Kleinberg [47] developed and analyzed a game-theoretic model of student participation in online forums for education. They used the game model to address questions regarding the optimal use of such forums. In the other words, the main goal was to incentivize higher student participation rate into answer questions from others. Ying et al. [48] proposed a set of incentive mechanisms to attract user participation crowdsourced spectrum sensing. The crowdsourcing system architecture periodically acquires data from users. In order to incentivize users, they introduced an auction-based incentive mechanism that is computationally efficient, individually rational and truthful. Furthermore, there were some efforts done to optimize the quality of incentivizing participants. For example, Kamar et al. [45] addressed the challenge in crowdsourcing systems of incentivizing people to contribute to the best of their abilities. They introduced a new payment rule, called consensus prediction rule, which uses the consensus of other workers to evaluate the report of a worker.

By adopting existing solutions we can improve the participation rate and system performance, which can ultimately benefit the inexperienced users. In the experimental phase, we can encourage people to participate into our data collection by offering them monetary awards. We can also use reciprocal game design and rewards offering to keep high participation rate. Expert users can receive credits by answering the permission requests correctly through RecDroid and inexperienced users can benefit from the recommendations from RecDroid.

Communication overhead: Regarding the network bandwidth usage, RecDroid's communicated packets between client and server are not larger than 10 bytes so RecDroid does not have high network overhead. In addition, since system call hook is embedded into the framework, and it does not check frequently, so battery consumption is not an concern.

9. Conclusion

In this paper we present RecDroid, an Android permission control and recommendation system which serves the goal of helping users perform low-risk resource accessing control on untrusted apps to protect their privacy and potentially improve efficiency of resource usages. We propose a framework that allows users to install apps in either trusted mode or probation mode. In the probation mode, users are prompt with resource accessing requests and make decisions to grant the permissions or not. Our RecDroid recommendation algorithm can effectively use crowdsourcing techniques to find expertise users in the user base and provide recommendation based on the responses from expertise nodes. Our evaluation results demonstrate that RecDroid recommending system can achieve high accuracy and good coverage when parameters are carefully selected. We also shows that RecDroid only need a small seed expert coverage to bootstrap the system. We implemented our system on Android phones and demonstrate that the system is feasible and effective.

References

[1] Gartner: 1.1 billion android smartphones, tablets expected to ship in 2014, Last Visit: September, 2015.
[2] W. Rothman, Smart phone malware: The six worst offenders, Last Visit: September, 2015. http://www.nbcnews.com/tech/mobile/smart-phone-malware-six-worst-offenders-f125248.

[3] What is the price of free, Last Visit: September, 2015. http://www.cam.ac.uk/research/news/what-is-the-price-of-free.
[4] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android permissions: User attention, comprehension, and behavior, in: SOUPS'12, ACM, New York, NY, USA, 2012, pp. 3:1–3:14.
[5] R. Mittal, A. Kansal, R. Chandra, Empowering developers to estimate app energy consumption, in: 18th CMCN, Mobicom'12, ACM, New York, NY, USA, 2012, pp. 317–328.
[6] O.R.E. Pereira, J.J.P.C. Rodrigues, Survey and analysis of current mobile learning applications and technologies, ACM Comput. Surv. 46 (2) (2013) 27:1–27:35.
[7] B. Liu, S. Nath, R. Govindan, J. Liu, DECAF: Detecting and characterizing ad fraud in mobile apps, in: 11th USENIX CNSDI, NSDI'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 57–70.
[8] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, M. Peter, L4android: A generic operating system framework for secure smartphones, in: SPSMD, SPSM'11, ACM, New York, NY, USA, 2011, pp. 39–50.
[9] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, D.S. Wallach, Quire: Lightweight provenance for smart phone operating systems, in: USENIX Security Symposium, 2011.
[10] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android permissions: User attention, comprehension, and behavior, in: UPS, SOUPS'12, ACM, 2012, pp. 3:1–3:14.
[11] A.P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner, Android permissions demystified, in: 18th CCS, ACM, 2011, pp. 627–638.
[12] B. Rashidi, C. Fung, T. Vu, Recdroid: A resource access permission control portal and recommendation service for smartphone users, in: Proceedings of the ACM MobiCom Workshop on Security and Privacy in Mobile Environments, SPME'14, ACM, New York, NY, USA, 2014, pp. 13–18.
[13] J. Crussell, R. Stevens, H. Chen, MAdFraud: Investigating ad fraud in android applications, in: 12th CMSAS, MobiSys'14, ACM, New York, NY, USA, 2014, pp. 123–134.
[14] A. Gember, C. Dragga, A. Akella, ECOS: Leveraging software-defined networks to support mobile application offloading, in: Eighth ACM/IEEE ANCS, ANCS'12, New York, NY, USA, pp. 199–210.
[15] J. Lin, S. Amini, J.I. Hong, N. Sadeh, J. Lindqvist, J. Zhang, Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing, in: 2012 CUC, UbiComp'12, ACM, New York, NY, USA, 2012, pp. 501–510.
[16] D. Barrera, J. Clark, D. McCarney, P.C. van Oorschot, Understanding and improving app installation security mechanisms through empirical analysis of android, in: SPSMD, SPSM'12, ACM, New York, NY, USA, 2012, pp. 81–92.
[17] A. Pathak, A. Jindal, Y.C. Hu, S.P. Midkiff, What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps, in: 10th CMSAS, MobiSys'12, ACM, New York, NY, USA, 2012, pp. 267–280.
[18] S. Guha, M. Jain, V.N. Padmanabhan, Koi: A location-privacy platform for smartphone apps, in: NSDI, NSDI'12, USENIX Association, 2012, pp. 183–196.
[19] W. Enck, M. Ongtang, P.D. McDaniel, et al., Understanding android security, IEEE Secur. Privacy 7 (1) (2009) 50–57.
[20] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, C. Glezer, Google android: A comprehensive security assessment, IEEE Secur. Privacy 8 (2) (2010) 35–44.
[21] J. Hildenbrand, Android app permissions—how google gets it right, Last Visit: September, 2015. http://www.windowsphone.com/.
[22] B. Rashidi, C. Fung, A survey of android security threats and defenses, J. Wirel. Mob. Netw., Ubiquitous Comput., Dependable Appl. (JoWUA) 6 (3) (2015) 3–35.
[23] Y. Agarwal, M. Hall, Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing, in: MobiSys'13, ACM, New York, NY, USA, 2013, pp. 97–110.
[24] G. Carullo, A. Castiglione, A. De Santis, F. Palmieri, A triadic closure and homophily-based recommendation system for online social networks, World Wide Web 18 (6) (2015) 1579–1601.
[25] Q. Ismail, T. Ahmed, A. Kapadia, M.K. Reiter, Crowdsourced exploration of security configurations, in: CHI'15, ACM, New York, NY, USA, 2015, pp. 467–476.
[26] L. Yang, N. Boushehrinejadmoradi, P. Roy, V. Ganapathy, L. Iftode, Short paper: Enhancing users' comprehension of android permissions, in: SPSM'12, ACM, New York, NY, USA, 2012, pp. 21–26.
[27] R. Amadeo, App ops: Android 4.3's hidden app permission manager, control permissions for individual apps!, Last Visit: August, 2015. http://www.androidpolice.com/2013/07/25/app-ops-android-4-3s-hidden-app-permission-manager-control-permissions-for-individual-apps/.
[28] F. Ricci, L. Rokach, B. Shapira, Introduction to recommender systems handbook, in: F. Ricci, L. Rokach, B. Shapira, P.B. Kantor (Eds.), Recommender Systems Handbook, Springer, US, 2011, pp. 1–35.
[29] A.R. Beresford, A. Rice, N. Skehin, R. Sohan, Mockdroid: trading privacy for application functionality on smartphones, in: HotMobile'11, pp. 49–54.
[30] Tools for managing os processes, Last Visit: May, 2015. http://developer.android.com/reference/android/os/Process.html.
[31] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, pp. 43–52.
[32] I. Gunes, C. Kaleli, A. Bilge, H. Polat, Shilling attacks against recommender systems: a comprehensive survey, Artif. Intell. Rev. 42 (4) (2012) 767–799.
[33] S.Y. Maw, Secure personalized recommendation system for mobile user, in: Information Security and Cryptology—ICISC 2010: 13th International Conference, Seoul, Korea, December 1–3, 2010, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 266–277. (Chapter).
[34] B. Rashidi, C. Fung, A game-theoretic model for defending against malicious users in recdroid, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM, May 2015, pp. 1339–1344.
[35] B. Rashidi, C. Fung, Disincentivizing malicious users in recdroid using Bayesian game model, J. Internet Serv. Inf. Secur. (JISIS) 5 (2) (2015) 33–46.
[36] B. Rashidi, C. Fung, Bottracer: Bot user detection using clustering method in recdroid, in: IEEE/IFIP Network Operations and Management Symposium, NOMS'16—DISSECT 2016, April 2016.
[37] A. Yaar, A. Perrig, D. Song, Siff: a stateless Internet flow filter to mitigate ddos flooding attacks, in: Proc. of the 2004 IEEE Symposium on Security and Privacy, 2004, pp. 130–143.
[38] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, Y.-C. Hu, Portcullis: Protecting connection setup from denial-of-capability attacks, SIGCOMM Comput. Commun. Rev. 37 (4) (2007) 289–300.
[39] T.R.T. Anderson, D. Wetherall, Preventing Internet denial-of-service with capabilities, SIGCOMM Comput. Commun. Rev. 34 (1) (2004) 39–44.
[40] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, R. Govindan, Cossack: Coordinated suppression of simultaneous attacks, in: Proc. of the DARPA Information Survivability Conference and Exposition, IEEE, 2003, pp. 2–13.
[41] T. Peng, C. Leckie, K. Ramamohanarao, Protection from distributed denial of service attacks using history-based ip filtering, in: IEEE International Conference on Communications, 2003. ICC'03., Vol. 1, IEEE, 2003, pp. 482–486.
[42] G. Carullo, A. Castiglione, G. Cattaneo, A.D. Santis, U. Fiore, F. Palmieri, Feeltrust: Providing trustworthy communications in ubiquitous mobile environment, in: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, AINA, March 2013, pp. 1113–1120.
[43] B. Hoh, T. Yan, D. Ganesan, K. Tracton, T. Iwuchukwu, J.S. Lee, Trucentive: A game-theoretic incentive platform for trustworthy mobile crowdsourcing parking services, in: 2012 15th International IEEE Conference on Intelligent Transportation Systems, ITSC, September 2012, pp. 160–166.
[44] G. Hsieh, R. Kocielnik, You get who you pay for: The impact of incentives on participation bias, in: Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing, CSCW'16, ACM, New York, NY, USA, 2016, pp. 823–835.
[45] E. Kamar, E. Horvitz, Incentives for truthful reporting in crowdsourcing, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems—Vol. 3, AAMAS'12, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 1329–1330.
[46] A. Ghosh, Social computing and user-generated content: A game-theoretic approach, SIGecom Exchanges 11 (2) (2012) 16–21.
[47] A. Ghosh, J. Kleinberg, Incentivizing participation in online forums for education, in: Proceedings of the Fourteenth ACM Conference on Electronic Commerce, EC'13, ACM, New York, NY, USA, 2013, pp. 525–542.
[48] X. Ying, S. Roy, R. Poovendran, Incentivizing crowdsourcing for radio environment mapping with statistical interpolation, in: 2015 IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN, September 2015, pp. 365–374.

Please cite this article in press as: B. Rashidi, et al., Android fine-grained permission control system with real-time expert recommendations, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.04.013