Dude, Ask The Experts!: Android Resource Access Permission Recommendation with RecDroid

Bahman Rashidi Carol Fung
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
{rashidib, cfung}@vcu.edu

Tam Vu
Department of Computer Science and Engineering
University of Colorado Denver
Denver, CO, USA
tam.vu@ucdenver.edu

***Abstract*—With the exponential growth of smartphone apps, it is prohibitive for apps market places, such as Google App Store for example, to thoroughly verify if an app is legitimate or malicious. As a result, mobile users are left to decide for themselves whether an app is safe to use. Even worse, recent studies have shown that most apps in markets request to collect data irrelevant to the main functions of the apps, which could cause leaking of private information or inefficient use of mobile resources. To assist users to make a right decision as for whether a permission request should be accepted, we propose RecDroid. RecDroid is a crowdsourcing recommendation framework that collects apps' permission requests and users' permission responses, from which a ranking algorithm is used to evaluate the expertise level of users and a voting algorithm is used to compute an appropriate response to the permission request (accept or reject). To bootstrap the recommendation system, RecDroid relies on a small set of seed expert users that could make reliable recommendations for a small set of application. Our evaluation results show that RecDroid can provide high accuracy and satisfying coverage with careful selection of parameters. The results also show that a small coverage from seed experts is sufficient for RecDroid to cover the majority of the app requests.**

I. Introduction

First introduced in 2008, Android has gained a tremendous momentum over the last few years. According to Gartner [3], a technology research and advisory firm, 1.1 billion devices running Google's Android OS will be shipped in 2014 alone, marking its 80 percent mobile market share. Attributing to this fast-pace increase is the proliferation of Android apps, which provides an ever-growing application ecosystem. Officially reported by Android Google Play Store, the number of apps in the store has reached 1 billion, surpassing its major competitor Apple Apps Store in late 2013 [42], and predicted to reach 1.5 billion apps by the end of 2014. The market offers a wide range of applications from entertainment, productivity, through health care and even the highly sensitive ones such as online dating, home security, business management, to mention a few.

As users more and more depend on mobile devices and applications, the privacy and security concerns become more important. For example, a malicious third-party app can not only steal private information, such as contact list, text messages, and location from its user, but can also cause financial loss of the users by making secretive premium-rate phone calls and text messages [39]. At the same time, the rapid growth of the number of applications on Android markets makes it hard for app market places, such as Google App Store for example, to thoroughly verify if an app is legitimate or malicious. As a result, mobile users are left to decide for themselves whether an app is safe to use.

In current Android architecture, users decide what resource is given to application at installation. Between applications, Android prevents malicious apps from unauthorized access to other apps' resource by isolating resources controlled by each app. In the current setting, users have to grant all resource access requests before installing and using an app. This permission control mechanism, however, has been proven ineffective to protect users privacy and resource 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 [2]. When installing a new app, a very small portion of users pay attention to the resource being requested, since they tend to rush through prompted permission request screens to get to use the application. Only a small portion (3%) of users pay attention and make correct answers to permission granting questions. In addition, the current Android permission warnings do not help most users make correct security decisions [19].

Realizing these shortcomings in the current Android architecture, much efforts have been put towards addressing the problems. Many resource management systems are proposed such as in [34], [37], [30]. Going deeper to the system level, L4Android [27] and Cells citecells isolate smart phone OSes for different usage environments in different virtual machines (VMs). QUIRE [13] 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 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 [18], [17], the reasons for the ineffectiveness of the current permission control system include: (1) inexperienced users do not realize resource requests are irrelevant and will compromise their privacy, (2) users have the urge to use the app and may be obliged to exchange their privacy for using the app. Because of that, to help inexperienced users who want to protect their privacy from untrusted apps, we propose a novel solution to assist users in making informed decision in permission control while keeping their required level of attention at minimum. We propose a participatory framework that leverages permission responses

from experts and peer users to suggest other users on how to response to a permission requests.

In particular, we provide a framework, called RecDroid, to improve and assist mobile users to control of their resource and privacy through crowd sourcing. First, the framework allows users to use apps without giving all permissions and receive help from expert users when permission requests appear. 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 also facilitate user-help-user environment, where expert users' decisions are recommended to inexperienced users. The framework provides the following functionalities:

- Two app installation modes for apps that is about to be installed: *trusted mode* and *probation mode*. In probation mode, at run time, an app has to request permission from users to access sensitive resources (e.g. GPS traces, contact information, friend list) when the resource is needed. In trusted mode, the app is fully trusted and all permissions are all granted.
- An architecture to intercept and collect apps' permission requests and responses, from which recommendations are made as for what permission from which apps should and should not be granted.
- A recommendation system to guide users with permission granting decisions, by serving users with recommendations from expert users on the same apps.
- A user-based ranking algorithm to rank security risks of mobile apps.

To the best of our knowledge, this is the first framework to (1) guide users to decide on which permissions should be granted by providing low-risk recommendations from expert users, and (2) use optional probation installation mode, in which which permission requests are adaptively prompted to users which potentially improves the usability of the permission-granting mechanisms, (3) allows market places to ranking their app based on their security and privacy rating that is implicitly submitted by users. We thoroughly evaluate our proposed algorithms through simulations.

II. Related Work

Due to its inherent constrains in resources, much effort has been done towards the principles and practices to manage resource usage [34], [37], [30], [12], [20], [29], [7], [36] and privacy protection [23], of mobile applications. However, the most common practice for resource access management today is Mandatory Access Control (MAC) mechanism [16], [41], which is found in API from major mobile players such as Apple iOS, Android, BlackBerry, and Windows Phone. In such paradigm, resource access from apps needs to be granted by users. In Android, for example, this is done through its static permission model [24], where users need to grant all requested permissions on installation. Once the permissions are granted by users, they cannot be revoked unless the application is uninstalled. Anderson et. al. studied the economical implications of this model across different application market places and platform [4]. Studies [18], [17] have shown that such permission control paradigms are not efficient since users are either not paying attention to permissions being requested or not aware of the the permissions' implications.

Based on the existing centralized market places (e.g., Android Market), Gilbert et. al. proposed AppInspector, an automated security testing and validation system [21]. The system is run by the market places or a third party to prove or disprove security properties of apps. However, they rely on a single party to perform the entire process including tracking sensitive information flow, identifying security and privacy risks, and reporting potential risks of information misuse. Therefore, AppInspector does not scale well with app base. As a result, malicious apps are prosperous in the market.

TaintDroid [14] and DroidChecker [11] are data flow tracking systems which allow users to track and analyze flows of sensitive data and potentially identify suspicious apps. TainDroid provides a tool for expert users to discover potential app misbehaving, while our system can effectively utilize responses from expert users and help other users protect their privacy through recommending appropriate permission granting decisions.

Among the most related work, Kirin [15] proposes to identify potentially malicious apps at installing time by looking at the security configuration bundled with an application to be installed. If predefined patterns are found, Kirin blocks and prevents the app from being installed. This approach is rather conservative and heavily depends on the predefined set of security rules, making it not scalable. Revising the current Android frameworks and/or runtime to provide fine-grained resource controls, Apex [35], AppFence [25], MockDroid [8], and TISSA [1] avoid giving out sensitive data by granting fake permission. While such approaches reduce the risk of leaking private information and critical resources, it requires users to make decisions on *every resource request*, which is difficult for inexperienced users and time consuming for others. To make it easier for users to control their resources, AppGuards [6], Aurasium [43], and FireDroid [40] allow users to define security policies on untrusted Android apps on which the policies are enforced through their framework. In FireDroid, an application monitor is created to track all processes spawned in Android and allow/deny them based on human managed policies. This approach requires rooting the device and extracting the Android booting partition, which is not practical for most users. In addition, the policy file editing and management are also not user friendly for most non-savvy users. Therefore, FireDroid targets on corporation phone users where the FireDroid can be pre-installed and policies are managed by administrators. Similarly, AppGuards and AurasiumThese require users to define a security policies which is not necessarily doable by inexperienced users. [22], [26], [28], [31], [32], [33] are proposed to rank, detect, and verify privacy risk and resource leakage of mobile apps. Crowdroid [10] and DroidChecker [11] propose a centralized approach to collect data traces from users and then analyze them to classify malware from legitimate apps. This is a complement approach to our except that RecDroid assists users at fine-grained permission level instead of at application level.

Going deeper to the system level, L4Android [27] and Cells citecells isolate smart phone OSes for different usage environments in different virtual machines (VMs). QUIRE [13] proposes a set of extensions addresses a form of attack, called

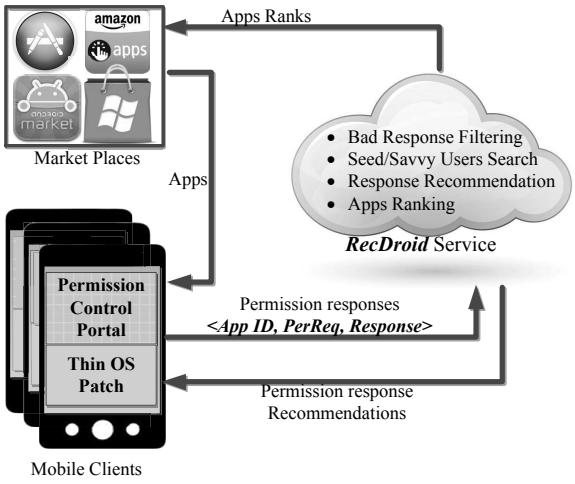


Fig. 1. RecDroid Service Overview

resource confused deputy attacks, in Android. To facilitate that, recently, ASF [5] proposes a security framework with API that allows authors of Android security extensions to develop their own modules. TrustDroid [9] allows applications to be ran at different trust levels (e.g. private vs. corporate). While these approaches are very useful in levitating vulnerabilities at system level, it does not help end-users to make a safer permission granting decisions. In contrast, our proposed approach prompts users adaptively based on the collective permission decisions from other users, and assists users in making decision through RecDroid recommendation system.

III. System Design

Our general approach is to build RecDroid with four functional processes, of which two are on mobile clients and the remaining two are on remote servers. In particular, RecDroid (1) collects users permission-request responses, (2) analyses 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. Figure 1 shows an overview of RecDroid service, 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.

Before going into further details about individual components, we first describe the permission handling procedure during an installation a mobile app with RecDroid. When a user first downloads and installs an application from app stores, the installer requests permission to access resources on the device. Instead of being sent to the operating system's legacy permission handler (e.g. Package Manager Service on Android) , the requests pass through RecDroid checks, which are installed on the mobile client at OS level. Figure 2 illustrates the permission checking and granting flow on RecDroid. In the first installation of an app, RecDroid allows the app to be installed on one of the two modes: *Trusted* and *Probation* (tracking mode), as shown in Figure 7(a). All requested permissions will be permanently granted to the app as specified by the user, if *Trusted* mode was selected. Otherwise, in *Probation* mode, RecDroid will compare the requested permissions against a predefined list of critical permissions that is of concern to users, such as location access, contact access, and messaging functions, etc. Regarding the installation mode selection, RecDroid also recommends an installation mode to the users based on collected data. For example, for new apps and apps that frequently receive rejections on permission requests, a probation installation mode should be recommended to users. With critical permissions, RecDroid client queries the online RecDroid service to get response recommendations for the permissions, specifically for the apps to be installed. Upon receiving the answer from the recommendation service, RecDroid client pops up a request, combined with the suggested response, to the user, as shown in Figure 7(c). Based on the suggested response, the user decide to grant or deny permission to access to certain resource. If a user chooses to deny a request, a dummy data or *null* will be returned to the application. For example, a denied GPS location request could be responded with a random location. That decision is both recorded in RecDroid client and populated back to RecDroid server for app ranking and analysis. Only then, the request is 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 first installed. Later, after collecting users responses and preferences; and having a security and privacy ranking of the app, RecDroid server should decide and notify RecDroid client whether to pop up permission requests or automatically answer them based on prior knowledge. Therefore, RecDroid strives to achieve a balance between the fine-grained control and the usability of the system. In order to realize RecDroid service, four main challenges need to be addressed.

- When to suggest user to put an app into a probation mode? Leaving that decision to be decided solely by a user is not a viable option because most users assume an app is legitimate and benevolent when they download.
- How do we instrument the operating system to intercept resource requests with the minimum amount of changes to the system such that it does not affect normal and legacy operations of the device? At the same time, how do we make that instrumentation work on both existing legacy apps and coming apps?
- Given that the responses from users are subjective, could be bias, and even malicious, how do we design the recommendation and ranking system based that could detect and then filter out untruthful or bias responses?
- How do we bootstrap and expand the recommendation system? Since this is a participatory service, it is important to have a sustainable and scalable approach that could provide valuable recommendations to all applications. It is certainly a challenging mission given millions of apps out there on different apps market places, and more to come.

At a high level, we propose a participatory framework to collect and analyze users' response to provide recommendations to users and rank apps based on the plausibility of their permission requests. In our proposed system, a *Permission Control Portal* installed on the mobile devices intercepts apps' permission requests, records the requests, and collect users' response to the requests.

Probation mode vs. Trusted mode The very first task of

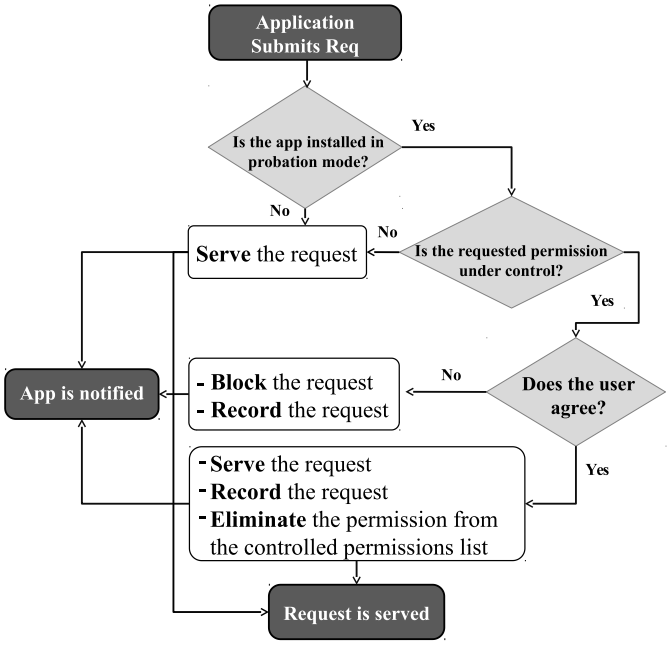


Fig. 2. Permission request flow in RecDroid

RecDroid is to decide whether to suggest an app to be installed in probation mode or in trusted mode. From metadata of the app, RecDroid identify the category to which the app belongs such as education, entertainment, tools, puzzle, music, etc.. It then looks at apps of the same category and same subcategory that it has knowledge about to decide if the set of permission requested by the app to be installed is reasonable. This process also relies on the input from expert users. From the previous selections of experts users for that app, RecDroid makes a suggestion of whether or not to put the app into probation mode.

Intercepting permission 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. One might argue that collecting this permission granting behaviors from users could raise privacy concern. However, since the portal does not collect any actual sensitive information, the data it collects doesn't contain private information. In fact, the portal merely communicates three-tuple data in the form of <*AppID, Permission Request, User's responses*>.

Bootstrapping the service: In order to suggest plausible responses to users, RecDroid starts from a set of *seed expert users* and make recommendation based on their responses. However, it is practical to have our expert users to provide plausible responses to hundreds of thousands of apps available on the market. 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 [38], RecDroid spanning algorithm is sketched as follow.

IV. 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.

A. 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,

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

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 seed experts 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 random variable $X \in \{0, 1\}$ to denote an random variable that a user answers the

permission requests correctly or not. Where $X = 1$ represents that user responds to a request correctly, vice verso. Therefore, we have $p = \mathbb{P}(X = 1)$. Given a sequence of observations on X, a beta distribution can be used to model the distribution of p.

In Bayesian inference, posterior 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,

$$\begin{aligned} \alpha_n =& \sum_{k=1}^{n} q^{n-k} x_k + q^n C_0 \qquad (4)\\ =& x_n + q x_{n-1} + ... + q^{n-1} x_1 + q^n C_0 \\ \beta_n =& \sum_{k=1}^{n} q^{n-k}(1 - x_k) + q^n C_0 \qquad (5)\\ =& (1 - x_n) + q(1 - x_{n-1}) + ... + q^{n-1}(1 - x_1) + q^n C_0 \end{aligned}$$

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.

Equation 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,

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

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.

We can see that 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.

B. 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 Algorithm1.

In the qualification step, only responses from qualified users are included into the voting process. Initially the ballot count for reception decision and rejection decision are equally initialized to D_0. For each qualified voter, the weight of the cast ballot is the ranking score of the voter. After the voting process finishes, the average ballot score is used to make a final decision. If the average ballot score exceeds a decision threshold, then corresponding recommendations are made. Otherwise, no recommendation is made.

V. Evaluation

To evaluate the performance of RecDroid, we conducted a set of simulation experiments to measure the accuracy, reliability, and effectiveness of the system.

Algorithm 1 Weighted Voting for Recommendation Decision

This algorithm is to decide whether to make recommendation, and what recommendation to make given the response from other regular users.

Notations :

M :the set of users who have responded to the permission question
s_i :the ranking of the i_{th} user
x_i :the response of the i_{th} user
τ_e :the minimum required ranking score to be classified into expert users
τ_d :the recommendation threshold
a, b :the cumulative ballots for reception and rejection decision
D_0 :the initial ballot count for both decisions

```
//initialize voting parameters
a = b = D_0
for each user u in M do
    if s_u > τ_e then
        //only qualified users responses are considered into the voting
        if x_u = 1 then
            a+ = s_u
        else
            b+ = s_u
        end if
    end if
end for
//decision making based on final ballots result
if a/(a+b) > 1 − τ_d then
    Recommend to accept the request
else if a/(a+b) < τ_d then
    Recommend to reject the request
else
    No recommendation
end if
```

A. 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.

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

B. Expertise Rating and the Impact of Parameters

The remembering parameter q (Equation 4) and conservation factor t (Equation 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 Figure 3(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 Figure 3(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. Figure 3(c) 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.

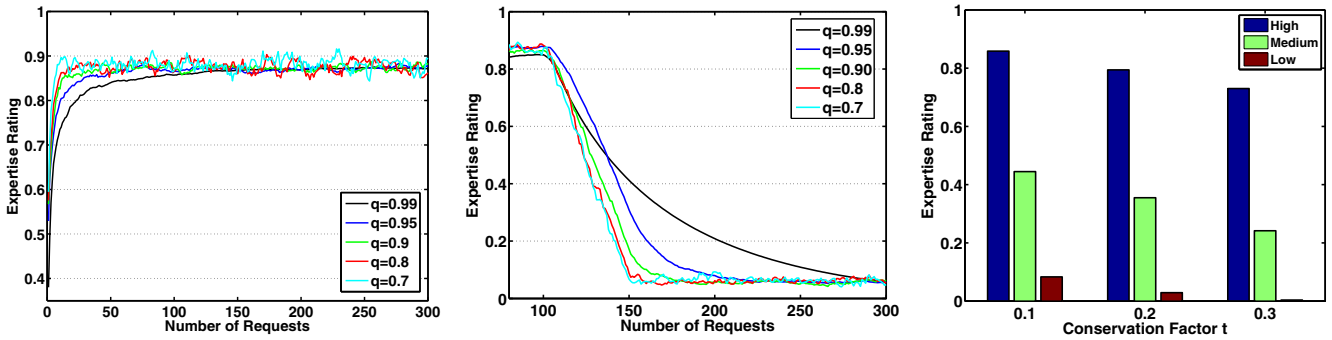
Figure 4 shows the expertise rating of the three types of users (with expertise $0.1, 0.5$, and 0.9). The blue boxes represents the central 50% of the expertise rating data while the red bars are the medium values of the samples. The vertical whiskers indicate the range of all data except outliners, which are represented by red crosses.

C. Coverage and Accuracy of RecDroid Recommendation

In this experiment we evaluate the performance of RecDroid 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.

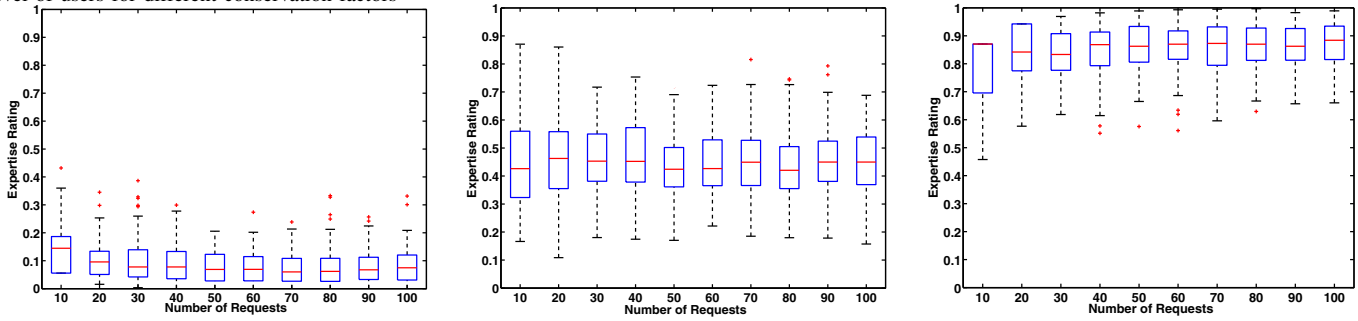
In the first experiment we investigate the scenario that 100 requests receive responses from all 100 users, except seed experts. RecDroid uses Algorithm 1 to determine whether to make a recommendation to new users or not and what recommendation it should make. Note that we assume all 100 users have received expert rating scores previously. We plot in Figure 5(a) and Figure 5(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.

Figure 5(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



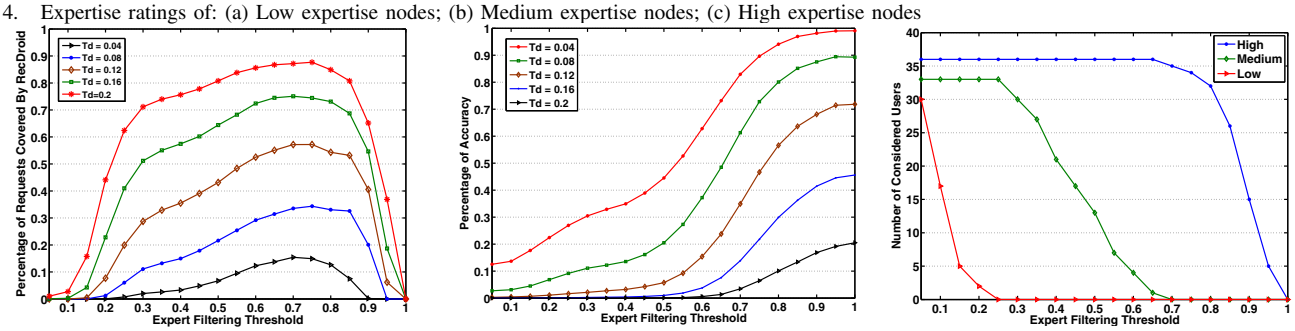
(a) (b) (c)

Fig. 3. Forgetting and Conservative factor: (a) Expertise level of users with different forgetting factor; (b) 100 percent of requests are answered; (c) Expertise level of users for different conservation factors



(a) (b) (c)

Fig. 4. Expertise ratings of: (a) Low expertise nodes; (b) Medium expertise nodes; (c) High expertise nodes



(a) (b) (c)

Fig. 5. 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

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 Figure 6, 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 simulation parameters are $\tau_e = 0.5$, $\tau_d = 0.2$, q=0.9, and t=0.1.

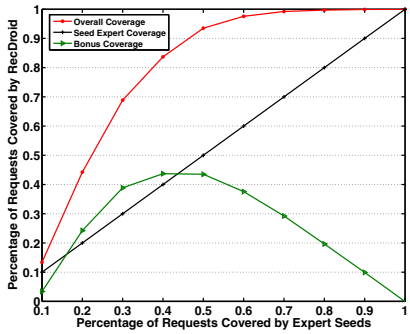
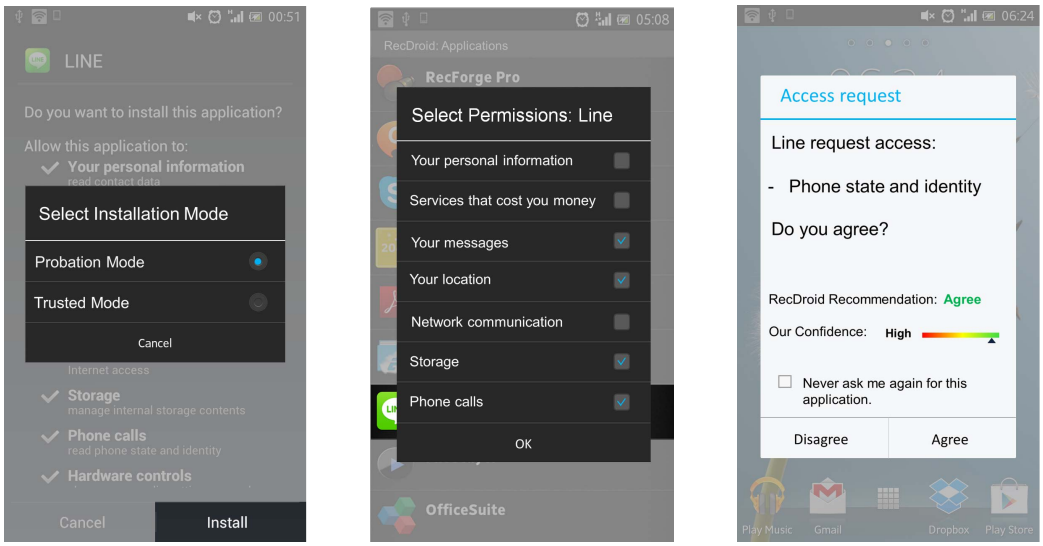


Fig. 6. Coverage of overall requests vs. coverage of seed experts



(a) (b) (c)

Fig. 7. An example of RecDroid on Line 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

VI. Preliminary Implementation

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 also provide users with an Android application to monitor and manage resource access permissions at fine-grain level.

Android OS modification: To implement a real-time resource permission control, RecDroid is designed to capture all resource access requests 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. The core of the modification is the *Package Manager Service*, which plays an important role in the installation of applications and their requested permissions management. The *ActivityManagerService* and the *ApplicationContext* classes, for example, are the places where the majority of modifications occur.

Our implementation was designed to be extensible and generic. While our implementation requires multiple changes in one place, it doesn't require modifications on every permission request handler, as it was the case on some previous works, such as in MockDroid [8]. 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.

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 (Figure 7(a)) displays two options when installing the app on the smart phone. 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. If the user click on an application in the list, a set of its requested permissions (see Figure 7(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 (Figure 7(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.

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 and also compute recommendations according to the recorded responses from users. The RecDroid clients request recommendations from the server when needed.

VII. Conclusion

In this paper we present RecDroid, a 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] Tissa. http://www.cs.ncsu.edu/faculty/jiang/pubs/TRUST11.pdf.

[2] What is the price of free. http://www.cam.ac.uk/research/news/what-is-the-price-of-free.

[3] Gartner: 1.1 billion android smartphones, tablets expected to ship in 2014, 2014.

[4] J. Anderson, J. Bonneau, and F. Stajano. Inglorious installers: Security in the application marketplace. In *WEIS*, 2010.

[5] M. Backes, S. Bugiel, S. Gerling, and P. von Styp-Rekowsky. Android security framework: Enabling generic and extensible access control on android.

[6] M. Backes, S. Gerling, C. Hammer, M. Maffei, and P. v. Styp-Rekowsky. AppGuard ï£¡ enforcing user requirements on android apps. In N. Piterman and S. A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 7795 in Lecture Notes in Computer Science, pages 543–548. Springer.

[7] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In *SPSMD*, SPSM '12, pages 81–92, New York, NY, USA, 2012. ACM.

[8] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: trading privacy for application functionality on smartphones. In *HotMobile'11*, pages 49–54.

[9] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Practical and lightweight domain isolation on android. In *Proceedings of the 1st ACM SPSMD*, SPSM '11, pages 51–62. ACM.

[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st ACM SPSMD*, SPSM '11, pages 15–26. ACM.

[11] P. P. Chan, L. C. Hui, and S. M. Yiu. DroidChecker: Analyzing android applications for capability leak. In *Proceedings of the Fifth ACM SPWMN*, WISEC '12, pages 125–136. ACM.

[12] J. Crussell, R. Stevens, and H. Chen. MAdFraud: Investigating ad fraud in android applications. In *12th CMSAS*, MobiSys '14, pages 123–134, New York, NY, USA, 2014. ACM.

[13] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight provenance for smart phone operating systems. In *USENIX Security Symposium*, 2011.

[14] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, volume 10, pages 1–6, 2010.

[15] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *16th ACM CCS*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[16] W. Enck, M. Ongtang, P. D. McDaniel, et al. Understanding android security. *IEEE Security & Privacy*, 7(1):50–57, 2009.

[17] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *18th CCS*, pages 627–638. ACM, 2011.

[18] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *UPS*, SOUPS '12, pages 3:1–3:14. ACM.

[19] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *SOUPS'12*, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[20] A. Gember, C. Dragga, and A. Akella. ECOS: Leveraging software-defined networks to support mobile application offloading. In *Eighth ACM/IEEE ANCS*, ANCS '12, pages 199–210, New York, NY, USA.

[21] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung. Vision: automated security validation of mobile apps at app markets. In *ACM MSC'11*, 2011.

[22] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. RiskRanker: Scalable and accurate zero-day android malware detection. In *MSAS*, MobiSys '12, pages 281–294. ACM.

[23] S. Guha, M. Jain, and V. N. Padmanabhan. Koi: A location-privacy platform for smartphone apps. In *NSDI*, NSDI'12, pages 14–14. USENIX Association.

[24] J. Hildenbrand. Android app permissions - how google gets it right. http://www.windowsphone.com/.

[25] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These arenï£¡t the droids youï£¡re looking for. *Retrofitting Android to Protect Data from Imperious Applicationsion. In: CCS*, 2011.

[26] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu. RiskMon: Continuous and automated risk assessment of mobile applications. In *DASP*, CODASPY '14, pages 99–110. ACM.

[27] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. L4android: A generic operating system framework for secure smartphones. In *SPSMD*, SPSM '11, pages 39–50, New York, NY, USA, 2011. ACM.

[28] L. Li, A. Bartel, J. Klein, and Y. Le Traon. Detecting privacy leaks in android apps.

[29] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *2012 CUC*, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM.

[30] B. Liu, S. Nath, R. Govindan, and J. Liu. DECAF: Detecting and characterizing ad fraud in mobile apps. In *11th USENIX CNSDI*, NSDI'14, pages 57–70, Berkeley, CA, USA, 2014. USENIX Association.

[31] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. CHEX: Statically vetting android apps for component hijacking vulnerabilities. In *CCS*, CCS '12, pages 229–240. ACM.

[32] W. Luo, S. Xu, and X. Jiang. Real-time detection and prevention of android SMS permission abuses. In *SESS*, SESP '13, pages 11–18. ACM.

[33] S. Matsumoto and K. Sakurai. A proposal for the privacy leakage verification tool for android application developers. In *CUIMC*, ICUIMC '13, pages 54:1–54:8. ACM.

[34] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *18th CMCN*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

[35] M. Nauman, S. Khan, and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *5th ACM SICCS*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[36] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps. In *10th CMSAS*, MobiSys '12, pages 267–280, New York, NY, USA, 2012. ACM.

[37] O. R. E. Pereira and J. J. P. C. Rodrigues. Survey and analysis of current mobile learning applications and technologies. *ACM Comput. Surv.*, 46(2):27:1–27:35, Dec. 2013.

[38] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer US, 2011.

[39] W. Rothman. Smart phone malware: The six worst offenders. http://www.nbcnews.com/tech/mobile/smart-phone-malware-six-worst-offenders-f125248.

[40] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark. Firedroid: Hardening security in almost-stock android. In *ACSAC'13*, pages 319–328, New York, NY, USA, 2013. ACM.

[41] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. *IEEE Security & Privacy*, 8(2):35–44, 2010.

[42] H. Victor. Android's google play beats app store with over 1 million apps, now officially largest. http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

[43] R. Xu, H. Saï£¡di, and R. Anderson. Aurasium: Practical policy enforcement for android applications. In *21st CSS*, Security'12, pages 27–27. USENIX Association.