RecDroid: A Resource Access Permission Control Portal and Recommendation Service for Smartphone Users

Bahman Rashidi
Dept. of Computer Science
Virginia Commonwealth University
rashidib@vcu.edu

Carol Fung
Dept. of Computer Science
Virginia Commonwealth University
cfung@vcu.edu

Tam Vu
Dept. of Computer Science and Engineering
University of Colorado Denver
tam.vu@ucdenver.edu

ABSTRACT

The rapid growth of smartphone application market raises security concerns regarding untrusted applications. Studies have shown that most apps in markets request to collect data irrelevant to the main functions of the apps. Traditional permission control design based on one-time decisions on installation has been proven to be not effective to protect user privacy and poorly utilize scarce mobile resources (e.g. battery). In this work, we propose RecDroid, a framework for smartphone users to make permission control in real time and receive recommendations from expert users who use the same apps. This way users can benefit from the expert opinions and make correct permission granting decisions. We describe our vision on realizing our solution on Android and show that our solution is feasible, easy to use, and effective.

Keywords

Permission; Recommendation; Malware; Android; Smartphone

1. INTRODUCTION

The proliferation of mobile apps is the primary driving force for the rapid increase of number of smartphone users. That number is predicted to double in the next 4 years, from 1.4 billion smartphone users world-wide in 2013 to 2.5 billion by 2017 [4]. The number of mobile apps has been growing exponentially in the past few years. According to the official report by Android Google Play Store, the number of apps in the store has reached 1 billion, surpassed its major competitor Apple Apps Store, in 2013 [20]. As the number of smartphone apps increases, the privacy and security problem of users has become a serious problem, especially when the smartphones are used for sensitive tasks and for business purposes. A malicious third-party app can not only steal private information, such as contact list, text messages, and location from the user, but can also cause financial loss of the users by making secretive premium-rate phone calls and text messages [17].

To prevent malicious apps from unauthorized resource access, Android isolate resources controlled by each app and users permission are required to access those resources. When installing a new app, users have to grant all resource access requests before using the app. However, the current permission control mechanism has been proven ineffective to protect users from malicious apps. Study shows that more than 70% of smartphone apps request to collect data irrelevant to the main function of the app [6]. When installing a new app, most users do not pay attention to the irrelevant resource access requests. Only a small portion (3%) of users pay attention and make correct answers to permission granting questions. Current Android permission warnings do not help most users make correct security decisions [12].

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. To help inexperienced users who want to protect their privacy from untrusted apps, we propose a novel solution to allow users to use apps without giving all permissions and receive help from expert users when permission requests appear.

This project is to provide a framework which allows users to install untrusted apps under a *"probation"* 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 any new apps: *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 recommendation system to guide users with permission granting decisions, by serving users with recommendations from expert users on the same apps.

To the best of our knowledge, this is the first framework to (1) guide users to decide on which permission should be granted to a certain app by providing low-risk recommendations from expert system, and (2) use optional probation installation mode, which adaptively prompt users with permission requests which potentially improves the usability of the permission-granting mechanisms; and allows market places to ranking their app based on their security and privacy rating that is implicitly submitted by users.

2. RELATED WORK

Due to its inherent constrains in resources, much effort has been done towards the principles and practices to manage resource usage and privacy protection of mobile applications. The most common

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SPME'14, September 11, 2014, Maui, Hawaii, USA.
Copyright 2014 ACM 978-1-4503-3075-6/14/09 ...$15.00.
http://dx.doi.org/10.1145/2646584.2646586 .

practice for resource access management today is Mandatory Access Control (MAC) mechanism [11, 19], which is found in API from major mobile players such as Apple iOS [3], Android [1], BlackBerry [2], and Windows Phone [7]. 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 [14], 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 [8].

Based on the existing centralized market places (e.g., Android Market), Gilbert et. al. proposed *AppInspector*, an automated security testing and validation system [13]. 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 [10] is a data flow tracking system which allows 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.

AppFence [15], MockDroid [9], and TISSA [5] avoid giving out sensitive data by granting fake permission. While such approaches reduce the risk of leaking private information, it requires users to make decisions on every resource request, which is difficult for inexperienced users. Our proposed approach prompts users adaptively based on the collective permission decisions from other users, and also provides decision recommendation.

A similar work to ours that we are aware is FireDroid [18]. It is a framework of policy-based access control for Android system. In this work, 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.

3. 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

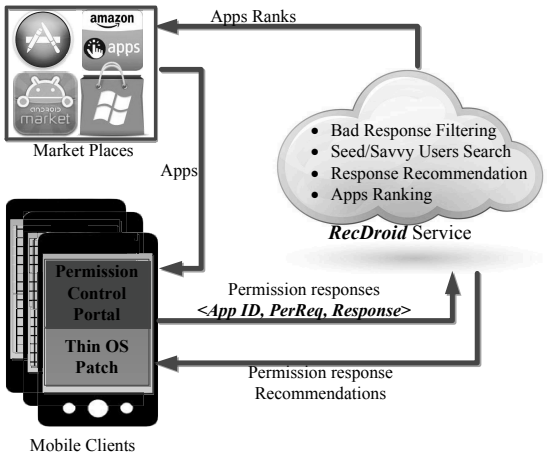


Figure 1: RecDroid Service Overview

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 4(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 4(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.

- 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

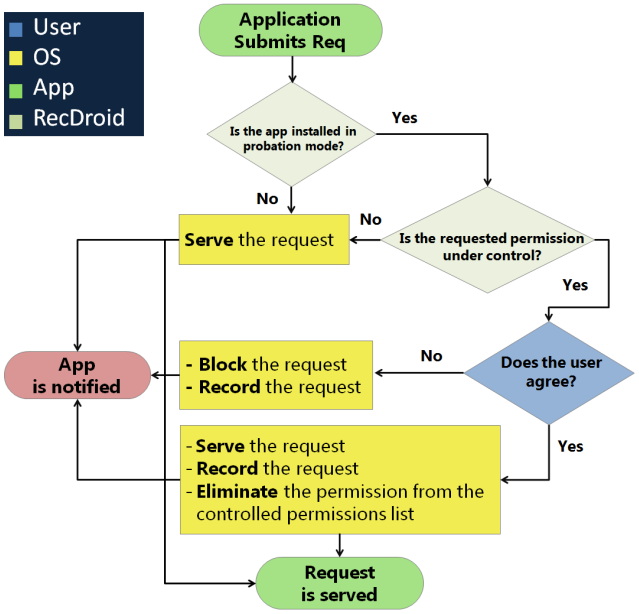


Figure 2: Permission request flow in RecDroid

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.

Next, we will provide our vision on addressing the above mentioned challenges. 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.

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 impractical 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 [16], RecDroid spanning algorithm is sketched as follow.

Algorithm 1 describes our external expert users search method. In its simplest form, if the similarity of unguided (without recommendation) responses from a user and the responses from our seed experts to the same app is high, then we recruit that user into the spanned expert users list. This base method could be further improved by understanding the criticality of a given user by looking at their responses to permission requests across multiple apps. For example, a response from a user that always say "Yes" to all permission requests might not be weighted as high as that of a user that has a diverse set of positive and negative responses. It is certainly true that the more responses our seed expert user send to the system, the more spanned expert users we can find.

Algorithm 1 Seek spanned expert users

Notations :
U :the set of all users
E_i :the set of initial seed expert users
E_s :the set of spanned expert users
for each user u in $U \setminus E_i$ **do**
  if The percentage of matching *unguided* responses from u and E_i is higher than a threshold θ and the number of matching samples is higher than τ **then**
    $E_s = E_s \cup u$
  end if
end for

Table 1 shows the permission decision table for the recommendation system. When a resource access request pops up, our system searches for the response from our seed experts to the same request first. If a match is found then send the response of the expert user as our recommendation and with high confidence. If only the response from a spanned expert user is found, then send the expert response as our recommendation with medium confidence. Otherwise no recommendation is made.

Table 1: Recommendation Decision Table

Top recommender	Recommendation	Confidence
Seed expert	Response from the expert	High
Spanned expert	Response from the expert	Medium
Others	No recommendation	-

4. 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 request at fine-grain level.

Android OS modification: To implement real-time resource permission control, the system needs to capture all resource access requests at running time. To achieve this goal, we created a software patch to modify the operating system, making changes to a few components and methods according to our specific needs.

Figure 3: Patching point to perform RecDroid's permission checking inside PackageManagerSerivce

Our Android OS modification has particularly focused on the Package Manager Service. It plays an important role in installing the application and managing their requested permissions. *PackageManagerService.java* class, for example, is where the most amount of modification is needed. Figure 3 depicts the structure of the main method of this class and the place where we inserted our code.

By design, our implementation is extensible and generic. While our method requires more changes at one place, it doesn't require modification to be made to every permission request handler as it was implemented in earlier works, such as in MockDroid [9] for example.

Permission Control Portal: The users of RecDroid have the options to install apps under a *probation mode*. We use the app "Line" (a popular chat application) as our example. The first screenshot (Figure 4(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 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 (Figure 4(c)). In addition, the system also gives a recommendation with a level of confidence to assist users to make decisions. If the user choose 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 the decision recommendation to users are essential to RecDroid. In our system, we create a remote server to record the responses and also pre-compute and store recommendations, which could become handy when access to Internet is not available to the client.

5. OPEN ISSUES AND FUTURE WORK

While the major challenges for RecDroid have been discussed, many other ones remain. In this section, we discuss some potential issues in RecDroid and our future plan to build such system.

Since RecDroid can be context-aware, some extra information will to be collected. For example, what time the response is issued and what request the user just sent before receiving a permission request. In some cases the collected data can be sensitive. The design of the RecDriod will protect the privacy of users by using encrypted communication and hashed identity in database. The data will be used internally and will not be disclosed to any third party.

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. 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 a method that could integrating context in without compromising users privacy.

Since the bootstrapping step relies on seed expert users, malicious seeds may cause RecDroid to provide misleading recommendations. We are aware of this potential threats and plan to add resilience design to RecDroid to filter dishonest or biased responses in our next phase. Furthermore, we also consider enhancing RecDroid in which recommendations are adapted according to each user's different security and privacy needs.

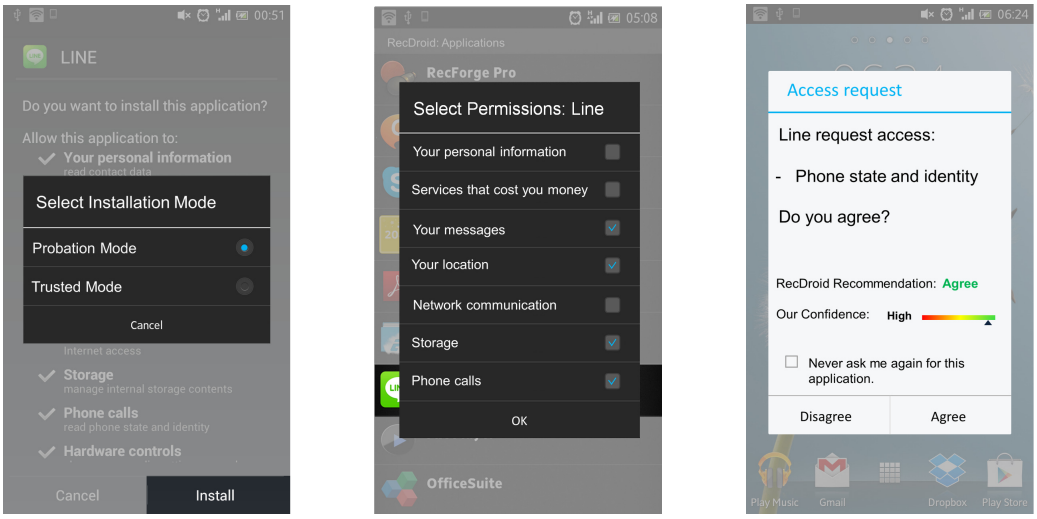
Similar to any other participatory services, RecDroid should have an incentive model to motivate users to participate. Benefit users getting from the recommended permission responses might not be enough to justify the inconvenient of reporting their decision to a centural server all the time. Incentivizing users however is out of scope of this paper.

6. CONCLUSION

We presented our vision for implementing RecDroid, a smartphone 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. RecDroid also provides expert recommendations on permission granting decisions to reduce the user's risk of making false decisions. We implemented our system on Android phones and demonstrate that the system is feasible and effective.

7. REFERENCES

[1] Android project. http://www.android.com.
[2] Blackberry 10. http://global.blackberry.com/blackberry-10.html.
[3] The iphone operating system. https://www.apple.com/ios/.
[4] Smartphone users worldwide will total 1.75 billion in 2014. http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536.
[5] Tissa. http://www.cs.ncsu.edu/faculty/jiang/pubs/TRUST11.pdf.
[6] What is the price of free. http://www.cam.ac.uk/research/news/what-is-the-price-of-free.
[7] Windows phone 8. http://www.windowsphone.com/.



(a) (b) (c)

Figure 4: 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

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

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These arenŠt the droids youŠre looking for. *Retrofitting Android to Protect Data from Imperious ApplicationsŤ. In: CCS*, 2011.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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`.