Short Paper: Enhancing Users' Comprehension of Android Permissions

Liu Yang, Nader Boushehrinejadmoradi, Pallab Roy, Vinod Ganapathy, Liviu Iftode
Department of Computer Science
Rutgers,The State University of New Jersey
110 Frelinghuysen Road
Piscataway, New Jersey 08854-8019, USA
{lyangru@cs,naderb@cs,pkroy@eden,vinodg@cs,iftode@cs}.rutgers.edu

ABSTRACT

Android adopts a permission-based model to protect user's data and system resources. An application needs to explicitly request user's approval of the required permissions at the installation time. The utility of the permission model depends critically on end users' ability to comprehend them. However, a recent study has shown that Android users have poor comprehension on permissions.

In this paper, we propose to help Android users better understand application permissions through crowdsourcing. In our approach, collections of users of the same application use our tool to help each other on permission understanding by sharing their permission reviews. We demonstrate the feasibility of our approach by implementing a proof-of-concept of our design. Our case study shows that the tool can provide helpful information of permission usage. It also exposes the limitations of the current implementation, and the challenges need to be addressed in our next step.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles

General Terms

Human Factors

Keywords

Android permission, Mobile applications, Permission understanding, Crowdsourcing, Record and replay.

1. INTRODUCTION

Android leads the market share in mobile application downloads since 2011 [13]. As an open source platform, Android supports third party application development with extensive APIs that provide access to the phone hardware, user data, and settings. To protect system resources and user's data, Android adopts an application-based permission model for security, where each application is required to explicitly indicate at installation time all resources needed for its funtionality at installation time. An application is installed only if the user approves all requested permissions.

The application-based permission model improves upon the traditional user-based permission model [8]. However, this model works as expected under the assumption that users could correctly understand the permissions they grant [1]. Unfortunately, recent studies have shown that Android phone users have poor understanding of permissions [9, 12]. In particular, Felt et al. found that only 17% of phone users paid attention to permissions during application installation, and only 3% of users could correctly answer all three permission comprehension questions [9]. These findings clearly indicate that the current permission warnings are not very effective in helping users make correct security decisions. As the number of applications are rapidly growing, there is an urgent need to find alternative approaches that help users better understand their application permissions.

In this paper, we propose to enhance user comprehension of application permissions using crowdsourcing. Our approach is to allow collections of users of the same application use our tool to help each other on permission understanding by sharing their permission reviews. Central to our approach is the assumption of *observability of permission change*. That is, we assume that changes in the permissions granted to an application reflect in changes in the application behavior that can be observed, and possibly understood by the user. Examples of observable and understandable changes in application behavior due to permission changes are given in Section 3. In this paper, we explore a particular case, namely when changes in the application behavior are **visible** to the user. We show that behavior changes of application due to permission suppression can provide helpful information for users to understand the purposes of certain permissions. To this end, we developed a tool called Droidganger.

Droidganger is constructed using two techniques: *record/replay* and *permission suppression*. It works in two steps. In the first step, Droidganger records the execution of an application with all requested permissions granted. During the second step, it replays the execution multiple times with certain requested permissions suppressed. When a permission suppression causes an application to deviate from its normal behavior, i.e., different than the execution during the record stage, Droidganger presents the differences to the user. The user is asked to translate the observed difference into a meaningful explanation on why the suppressed permission is needed. Droidganger uploads a user's permission comments to a Comments Processing Server (CPS), where permission comments

[1]A second assumption is that the system has no capability leaks on permissions, but Grace et al. has shown that it is not true [10].

SPSM'12, October 19, 2012, Raleigh, North Carolina, USA.

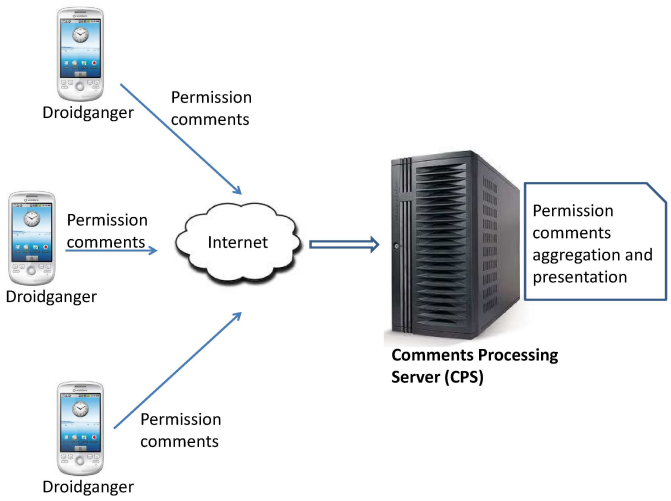

Figure 1: Design overview.

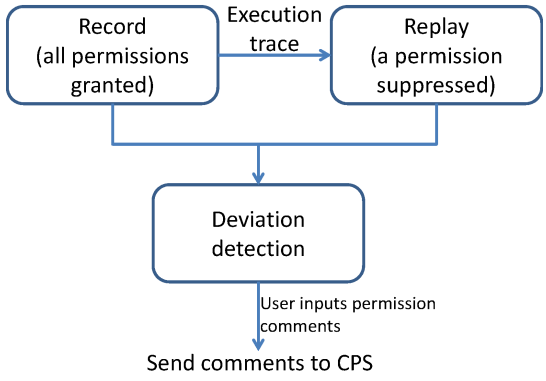

Figure 2: The structure of Droidganger.

are aggregated and presented for public access. Droidganger does not interfere with users while they are using an application.

An individual user often uses a portion of functionalities of an application. Thus, her usage may only cover a portion of permissions requested by an application. We expect to increase permission coverage through *crowdsourcing* by allowing collections of users of the same application to use Droidganger to help each other on permission understanding by sharing their permission reviews on CPS. Crowdsourcing is an efficient way to cope with the huge number of applications in Android market. It also makes the review burden of individual users small.

We demonstrate the feasibility of our approach by implementing a proof-of-concept of Droidganger, which is capable of recording the user input events, and replaying the execution of an application with the recorded trace. Our case study shows that even with a simple visual detection of the difference in output, our approach can help users understand about 40% of application permissions. It also exposes the limitations of the current implementation, and the challenges need to be addressed in our next step.

2. ARCHITECTURE

Figure 1 depicts our approach. The envisioned system employs a client-server model. The clients of the system are Android *Application Communities*. An application community is a collection of Android users of a specific application who use Droidganger as a tool to interpret permissions requested by that application. To be a client, an Android user simply installs Droidganger on her device, and uses her applications as usual. Users' comments on permission usage of applications are submitted to a Comments Processing Server (CPS), which aggregates the permission comments for different applications. The aggregated permission comments can then be accessed by any Android user, who can use the comments to gauge the benefits and risks of installing an application.

2.1 Droidganger

Droidganger is designed to give users helpful hints on how permissions are associated with an application's functionality. The design is based on the assumption that every requested permission, if not redundant, serves one or more purposes in an application. Depriving a requested permission may cause an application to be less functional or deviate from its normal behavior (where all requested permissions are granted). We expect that some behavior deviations due to permission suppression may provide meaningful clues for users to understand *why a permission is needed* or *what purposes it serves* in an application. Droidganger has three components: a *Record* module, which records the execution of an application when all requested permissions are granted; a *Replay* module, which replays the application using the recorded traces with certain requested permissions suppressed; a *Deviation Detection* module, which compares the snapshots of recorded and replayed executions and detects the deviations, and presents the deviations to users. Figure 2 shows the structure of Droidganger.

Record.

The Record module runs as a background service. When the user interacts with the application under study, the Record module records the execution of the application. To replay an application, we record all nondeterministric inputs and events that influence the execution. These inputs and events include user's keystroke, screen touches/drags, hardware lock, network traffic, location data from the GPS sensor, etc. These data can be obtained by monitoring the system calls made by an application. In addition, we also need to record the start state of the execution, including the application configuration, and a snapshot of the file system, etc. To allow for a later comparison, some snapshots, e.g., screenshots, system logs, of an execution are also recorded. All this data is stored in a trace file on the phone itself.

Replay.

The replay module replays a recorded trace after suppressing one of the permissions requested by the application. During the replay, snapshots corresponding to the same phases as in the record stage are taken for deviation detection. To observe the effects of each requested permission, the Replay module can be configured to replay a recorded trace multiple times, each time with a different permission being suppressed. However, we envision that each user in an application community will only replay the application with one permission suppressed. A large application community will assure that all permissions of an application will be fully covered.

Deviation Detection.

To observe the effects of a permission, the snapshots of the replay where the permission was suppressed are compared pairwise with the snapshots taken in the record stage. If there is a significant difference between a pair, the user is shown the snapshot pair and she is asked to decide whether the difference might be due to the

suppressed permission. If so, the user inputs her comments on what purposes this permission serves in the application. For example, in a game application, the difference between a pair of screenshots may be that the advertisements did not appear in the screenshot when the INTERNET permission was suppressed. Then the user may input her comments saying that one purpose of using INTERNET in the game is to retrieve advertisements. Droidganger submits users' comments to the Comments Processing Server (CPS).

2.2 Comments Processing Server

The comments processing server (CPS) is responsible for analyzing and presenting the permission comments received from users. Users' comments are anonymized and grouped by application names. Comments for an application are grouped by permission names. Comments for a permission are aggregated and presented in a ranked manner. If a permission has too many comments, the top k are presented. The ranking of comments may change over time since the CPS continuously receives comments from users. Techniques commonly used in natural language processing can be used for comments aggregation and ranking. Also, the CPS can use heuristics to discard spam submissions.

3. FEASIBILITY STUDY

We have implemented a proof-of-concept of Droidganger on Android 4.0.3 and deployed it on the Android emulator. In our implementation, the Record module is capable of recording user input events, e.g., keystrokes, touches, drags, etc. The Replay module replays executions by reading the recorded input events. We modified Android Application Framework to allow us to suppress any permission of an application. Since this study is focused on visual effects of permission changes, we implemented the Deviation Detection module as an image comparator, which compares screenshots of the record and replay stages pairwise. We evaluated the feasibility of our approach using two applications downloaded from Android market. Our experiments showed that even with a simple visual detection of the differences in output, our approach can help user better understand about 40% of application permissions.

3.1 Implementation

Record.

We made minor changes to the Android Application Framework by inserting code to record user input events while a user interacts with an application. The record module records user's keystrokes, touches, and drags by monitoring the KeyEvents and MotionEvents in the InputManger of the Framework. We also record the time of each event, and screenshots for certain events.

Replay.

The Replay module reads a recorded trace, i.e., a sequence of input events and times, translates the trace into `Monkeyrunner` scripts. It then calls `Monkeyrunner` to execute the scripts translated from a trace. We instruct `Monkeyrunner` to take screenshots corresponding to the same events as those in record stage. We modified the PackageManager of the Framework to allow users to selectively suppress permissions requested by an application.

Deviation Detection.

We implemented an image comparator to detect the differences between screenshots taken in the record and replay stages. The Deviation Detection module reads two sequences of screenshots and compares them pairwise. For a pair of images, it computes a score ranging from 0 to 1 measuring the degree of their similarity, where 1 denotes two images are equal. Two images are considered same if the score of comparison is higher than a threshold. In our experiments, we use a threshold of 0.95. Screenshot pairs with scores lower than a threshold are presented to user for review.

3.2 Case Study

Data Sets.

We downloaded two applications from Android market: AngryBirds Rio and Antivirus. The numbers of permissions requested by these applications are 6 and 39 [2]. Together the two applications request 40 unique permissions.

Methodology.

We conducted an internal user study within our research group to evaluate the feasibility of our approach. Three students (one undergraduate and two graduates) were invited to use Android emulator equipped with our Droidganger implementation to observe the permission usage of the two applications. Each student used Droidganger to record traces of her application usage, and replayed the traces at a later time. During the record stage, all requested permissions were granted. While in the replay stage, one of the requested permissions was suppressed. For an application with n requested permissions, a recorded trace was replayed n times, with a different permission being suppressed each time. In a real deployment, we envision that different permissions could be suppressed during the replay stage for various members of the application community. Screenshots of the recorded and replayed stages are compared pairwise to find execution deviation caused by permission suppression. Three students cross-validated their observations of the effects caused by permission suppression.

Findings.

The results of the user study are summarized in Figure 3, where we categorize the side effects of permission suppression into four types:

- *Meaningful* effects: an execution deviation due to a permission suppression provides meaningful clues to the purposes of a permission. For example, in a pair of AngryBirds screenshots shown in Figure 4, we found that advertisement did not appear after the INTERNET permission was suppressed, while a Goolge Play advertisement appeared on the top right screen when all permissions were granted. Such an effect can be considered as an indication that one purpose of the INTERNET permission in AngryBirds is to access advertisements.

- *Crashed* effects: an application crashed or failed to start when a permission was suppressed. We observed two types of crashes due to permission suppression: (1) failed to start, and (2) crashed in the middle. Case (1) often does not provide meaningful information to common users. In case (2), reviewing the snapshots before the crash often gives helpful information to a user: from that a user can learn which task he was performing before the crash happened.

- *Syslog only*: an execution deviation due to permission suppression did not appear as screenshot difference, but it was

[2]We only consider Android defined permissions and ignore permissions defined by third parties.

App \ Effects	Meaningful	Crashed	Syslog only	None
AngryBirds	2	1	1	2
Antivirus	14	4	8	13

Figure 3: Observed effects of the two applications.

captured in the system log. In our study, around 20% of effects by permission suppression could only be captured by the system log.

- *None* denotes that no visual effects or system log information were observed when a permission was suppressed. There are three cases in which a permission suppression may have no effects to the execution: (1) the permission is redundant, or (2) the APIs associated with the permission are unusable on an emulator, e.g., location service, etc., or (3) the APIs that related to the permission were not invoked. To test our conjecture, we uploaded the Antivirus application to Stowaway [7] and the returned results indicated that this application is overprivileged, where 7 of the 39 requested permissions are redundant. Assuming the results returned by Stowaway are relatively accurate, our current implementation provides meaningful clues to $\frac{14}{39-7} = 44\%$ of permissions requested by Antivirus.

Figure 5 is a screenshot pair taken from Antivirus application. It can be observed that the application failed to wipe personal data after the WRITE_SYNC_SETTINGS permission was suppressed during the replay stage. Such a deviation indicates that one purpose of requesting the WRITE_SYNC_SETTINGS permission is to provide the ability to “wipe personal data" in the application. Due to space limits, we could not list screenshots of other permission suppressions.

Among the 40 requested permissions, 6 of them are not observable on an emulator because emulator does not provide services, e.g., location-based, associated with these permissions. We expect that our current prototype of Droidganger may provide meaningful clues to a higher percentage of permissions on real devices.

Also, by increasing code coverage of execution, e.g., asking a large number of users to use an application (as would be expected in a real application community), we expect that some permission suppressions that had no effect in our study may provide meaningful clues on permission usage.

4. CHALLENGES

There are a number of challenges to be addressed in order to successfully build the envisioned system as described in Section 2.

Network Proxy.

Many applications access Internet during execution. A survey by Felt et al. showed that 86.6% of free and 65.0% of paid applications request INTERNET permission [8]. To ensure a replay has same network traffic inputs, we need to build a network proxy to record the incoming traffic to the device. During the replay, the replay module accesses the proxy and retrieves the traffic data needed for replay.

Randomness.

Some applications introduce randomness for unpredictability, e.g., in card games, cards are shuffled before being dealt. For these applications, same inputs and events may produce different outputs, thus, an execution may not be replayed well if the status

(a) Record stage

(b) Replay stage

Figure 4: A pair of screenshots of AngryBirds during record and replay stages. During the record stage, all permissions were granted; while in the replay stage, the INTERNET permission was suppressed. It can be observed that advertisements did not appear in (b).

(a) Record stage

(b) Replay stage

Figure 5: A pair of screenshots of Antivirus during record and replay stages. During the record stage, all permissions were granted; while in the replay stage, the WRITE_SYNC_SETTINGS permission was suppressed.

of the Pseudo Random Generator (PRG) is unknown. Recording an execution in VM level allows full replay. However, VM level record/replay is too expensive for a smartphone. This calls for a light-weight record/replay technique.

Non-repeatable Executions.

Executions related with online purchases are often hard to replay. For example, if a user makes an online payment during a record stage, then replaying the execution will make another payment, which is often not desired by the user. For these applications, we need to find an alternative way to help users understand the purposes of permissions.

Application Failure.

Since application developers may not capture exceptions triggered by permission failures, an application may crash if one of the requested permissions is suppressed. In that case, we may need to identify the failed API calls, and combine context information in order to derive a permission usage.

User Incentives.

Since Droidganger uses crowdsourcing, its effectiveness is proportional to the number of users who actively use it. Using incentives can increase the number of active users and the quality of submitted comments. Finding the incentive that maximizes the number of Droidganger users and the quality of their submissions is an important challenge.

5. RELATED WORK

Android's application permission-based security model improves upon the traditional user-based permission model [8]. Even so, researchers found that large number of applications have potential to misuse users' private information [11, 4, 5].

Researchers have developed a number of techniques to amend Android permission-based model. Nauman et al. proposed Apex, a framework allowing users to selectively granting permissions to applications [16] on their devices. CyanogenMod is an open source replacement firmware distribution based on Android system [3]. One of the features offered by CyanogenMod is permission suppression. Beresford et al. proposed Mockdroid, a tool allowing phone users to selectively provide fake information to applications which request sensitive information [1]. Enck et al. proposed Taintdroid, a dynamic taint tracking and analysis tool that allows users to track the usage of their sensitive data by third-party applications. Hornyack et al. built AppFence, which allows users to protect their sensitive data by data shadowing and exfiltration blocking [11]. AppFence originated the concept of "visibility of privacy control". The authors studied how privacy control affects the functionalities of applications.

An underlying assumption of tools like Apex, CyanogenMod, Mockdroid, and AppFence is that the user has a good understanding of Android permissions, otherwise she could not properly use or configure the tools. However, researchers have shown that Android users have poor understanding of permissions [9, 12]. In [6], Felt et al. showed that application developers may also misunderstand permissions. Their study found that around one third of Android applications are overprivileged [6], partly due to application developers' confusion on permission understanding.

Crowdsourcing provides the opportunities to delegate tasks typically performed by expensive in-house teams to mass of Internet users. The concept of collaborative security has been applied to malware detection [2], SMS spam filtering [18], and software self-healing [15], etc. Concurrent with our work, Lin et al. presented a study on *privacy as expectations*, a crowdsourcing model used for studying users' expectation of what sensitive resources mobile apps use [14]. In their study, users were shown application descriptions and developer-selected screenshots and were asked whether and why certain privacy related resources are used by applications. Our approach also employs crowdsourcing, but we enhance users' understanding of general permissions (not limit to privacy related ones) by showing them deviations of app executions when certain permissions are suppressed. Both [14] and our work leverage the concepts of crowdsourcing and collaborative security. We propose an architecture that allows Android users to collaboratively label the usages of permissions requested by applications. In this way, application community users help each other and thus improve their understanding of permissions.

Doppelganger is a tool that helps users to perform privacy preserving cookie configuration on their Internet browsers [17]. Both Doppelganger and our work leverage the execution deviation caused by policy configurations. However, Doppelganger aimed for cookie configuration of Internet browsers, while our work focuses on improving users' comprehension of Android permissions. Doppelganger employs parallel execution, while our approach uses record/replay to achieve our goal.

6. CONCLUSION

In this paper, we propose to use crowdsourcing to enhance user comprehension of Android permissions. We design a tool, Droidganger, which provides information of permission usage to users using two techniques: record/replay and permission suppression. Our experiments showed that even with a simple visual detection of the differences in output, our approach can help users better understand about 40% of application permissions.

Acknowledgments

This work was supported in part by NSF grant CNS-1117711 and US Army STIR grant W911NF-12-1-0018. We thank the anonymous SPSM reviewers for useful comments on an earlier draft of this paper.

7. REFERENCES

[1] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.

[2] J. Cheng, S. H. Wong, H. Yang, and S. Lu. Smartsiren: virus detection and alert for smartphones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 258–271, New York, NY, USA, 2007. ACM.

[3] CyanogenMod. `http://www.cyanogenmod.com/`, Retrieved in Aug 2012.

[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[5] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[6] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th*

ACM conference on Computer and communications security, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[7] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Stowaway: A static analysis tool and permission map for identifying permission use in android applications. `http://www.android-permissions.org/`, 2011.

[8] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[9] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[10] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic detection of capability leaks in stock Android smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, Feb. 2012.

[11] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

[12] P. Kelley, S. Consolvo, L. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: Installing applications on an Android smartphone. In *Proceedings of the Workshop on Usable Security (USEC)*, Feb. 2012.

[13] J. Kendrick. Latest smartphone market share numbers: Apple is flat, google going strong. `http://www.zdnet.com/blog/mobile-news/latest-smartphone-market-share-numbers-apple-is-flat-google-going-strong/2387`, May 2011.

[14] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expecation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 14th ACM International Conference on Ubiquitous Computing*, Sep 2012.

[15] M. E. Locasto, S. Sidiroglou, and A. D. Keromytis. Software self-healing using collaborative application communities. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA*. The Internet Society, 2006.

[16] M. Nauman, S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[17] U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 154–167, New York, NY, USA, 2006. ACM.

[18] K. Yadav, P. Kumaraguru, A. Goyal, A. Gupta, and V. Naik. Smsassassin: Crowdsourcing driven mobile-based system for sms spam filtering. In *Proceedings of the 12th Workshop on Mobile Computing Systems & Applications*, HotMobile '11. ACM, 2011.