7-2013

The Case for Mobile Forensics of Private Data Leaks: Towards Large-Scale User-Oriented Privacy Protection

Joseph Joo Keng CHAN
Singapore Management University, joseph.chan.2012@smu.edu.sg

Kiat Wee TAN
Singapore Management University, kwtan.2010@smu.edu.sg

Lingxiao JIANG
Singapore Management University, lxjiang@smu.edu.sg

Rajesh Krishna BALAN
Singapore Management University, rajesh@smu.edu.sg

The Case for Mobile Forensics of Private Data Leaks: Towards Large-Scale User-Oriented Privacy Protection

Joseph Chan Joo Keng, Tan Kiat Wee, Lingxiao Jiang, and Rajesh Krishna Balan
School of Information Systems, Singapore Management University
{joseph.chan.2012, kwtan.2010}@phdis.smu.edu.sg, {lxjiang, rajesh}@smu.edu.sg

ABSTRACT

Privacy protection against mobile applications on mobile devices is becoming a serious concern as user sensitive data may be leaked without proper justification. Most current leak detection tools only report leaked private data, but provide inadequate information about the causes of the leaks for end users to take preventive measures. Hence, users often cannot reconcile the way they have used an application to a reported leak — i.e., they are unable to comprehend the (il)legitimacy of the leak or make a decision on whether to allow the leak. This paper aims to demonstrate the feasibility and benefits of identifying the causes of leaks from a user's point of view, which we call *mobile forensics of privacy leaks*. Its goal is to correlate user actions to leaks, and report the causes from a user-oriented perspective. To make the case, we have performed a preliminary study that identifies leak causes based on logs of user actions in more than 220 Android applications and corresponding leak reports from a leak detection tool. Our results show that more than 60% of the 105 applications (of the 220 we sampled) that leak private data leak data do so due to user actions on certain in-application GUI widgets. About 44% also leak data right after users launch them, while 32% leak data periodically after launch. We also constructed a database containing leak causes from all tested apps, and demonstrated the use of visual overlays to warn users about potential leaks.

1. INTRODUCTION

We live in a world continually being proliferated by uses of mobile devices, where large amount of private data is accessible by mobile applications (apps), such as phone IMEI, SMS, GPS locations, our contact list, and more [1]. The data becomes a potential treasure trove to malicious users who can sell it to analytics company, advertising companies or even competitors. Balancing mobile privacy and functionality thus becomes an increasingly urgent focus for research.

Various leak monitoring tools have been developed to detect leaks of private data, such as TaintDroid [2] and PiOS [3]. These tools either uses dynamic or static analysis to perform leak detection. However, these tools have common shortcomings that the reported leaks contain limited information about the causes of the leaks: 1) End users cannot easily reconcile their actions to leaks (e.g., did this leak occur because I swiped a screen or tapped a particular button?). 2) User cannot make decisions on the (il)legitimacy of the leaks, and may easily make the same leak-causing actions again.

We present the case for *mobile forensics of privacy leaks* that aims to correlate user actions to leaks. We propose an approach framework for enabling such mobile forensics: user actions in mobile apps are logged through an instrumentation of the mobile system which is Android in our case; privacy leaks for the apps are also logged through a leak detection tool (TaintDroid); then correlation between user actions and leaks are discovered through leak-cause analysis (an association rule mining [4–6] technique). With this approach framework, we demonstrate the feasibility and benefits of mobile forensics with a preliminary study conducted on over 220 Android apps. We find that almost half (47%) of the apps sampled leak various kinds of privacy data. More than 60% of the 105 leaky apps leak data as a result of user actions on certain GUI widgets in the apps. About 44% of the leaky apps also leak data right after users launch them, while 32% leak data periodically after launched.

In addition, we have constructed a database containing leak causes from all tested apps, and further performed manual evaluation of the identified leak causes and find that they have promising accuracies: the causes identified for about 26.6% of the apps have accuracies in the range of 80-100%, and the accuracies of the causes identified for more than 60% apps are higher than 60%. We also have demonstrated the use of visual overlays to warn users about potential leaks so that users may easily identify the potential leak-causing widgets and based on the interface of the widgets have easier time in deciding whether the functionality of the app warrants the leak.

APSYS '13, July 29-30 2013, Singapore

The rest of the paper is organised as follows. In Section 2, we present potential motivational applications for mobile forensics of privacy leaks. Section 3 describes related work. Section 4 presents our approach framework for mobile forensics. Section 5 shows the empirical study conducted to support the case for mobile forensics. Finally, we discuss the study with future work in Section 6 and conclude in Section 7.

2. MOTIVATION

The key objective for our mobile forensics is to provide user-oriented information for end users to comprehend the linkage between their own actions and leaks so that users can make better decisions in protecting their privacy. We describe two scenarios where leak causes that relate user actions to leaks can aid users.

2.1 Interactive Privacy Leak Reporter

One deployment scenario of leak causes identified by mobile forensics is to enable a real-time privacy leak reporter to track user actions performed in an application so far and interactively informs the user of potential leaks. Different from existing leak detectors (e.g., TaintDroid, PiOS) that report leaks when they happen, such an interactive leak reporter has the knowledge about leak-causing user actions, and can be designed to capture such user actions, and warn users about widgets that may leak privacy in certain follow-up actions, depending on the current mobile screen content. The warning could take the form of real-time highlighting and/or disabling of leaky widgets in the app, or an off-line replay tool that replays the user actions that cause privacy leaks. In this manner, the user will still be able to use a leaky mobile app while side-stepping any user actions that can cause privacy leaks. The user-centric and fine-grained nature of a mobile forensics-based leak reporter also means that a user can potentially better understand the connection between app functionalities and types of data being leaked, and would be better equipped to make an informed choice in utilizing an app. However, such a leak reporter needs to be designed carefully to ensure good usability for the user.

2.2 Enhanced App Stores

Another scenario of using the knowledge of leak causes is through application stores when users download and install an app. Current installation process only informs users of the permissions the app needs, but does not tell whether the permissions are indeed necessary for the functionality of the app. The leak-cause knowledge not only informs what the leak may be, it can also inform how and where it leaks possibly through screen captures or replays of the leak-causing actions in the app. App stores can collect such knowledge through a system working internally in a similar way to our approach framework. For newly uploaded applications, the app stores in the verification phase can employ mobile forensics and collected leak-cause knowledge to reject the app should it contains any unnecessary leaks.

Another interesting possibility would be crowd-sourcing user-perceptions on the app leak-causes from real users of Android apps. Based on the interactive privacy leak reporter in the previous scenario, the app stores could allow users to install a questionable app and make the final choice by presenting the relevant leak-cause knowledge to them. This can also be a more convenient way to engage users in providing feedback on whether a permission requested by an app is really needed for the functionality of the app. This may also provide an interesting angle for research on user-centric permissions analysis.

3. RELATED WORK

In this section, we describe some related work on mobile privacy and discuss their limitations.

Privacy Leak Detection. Various tools, such as TaintDroid and PiOS [2, 3], have been developed for privacy leak detection in mobile applications. These tools are useful in detecting leaked private data, but they are less useful in guiding users to react appropriately with respect to a leak report. For example, a notification of a leak of private data has occurred while the user is on the launcher screen does not indicate why that leak has occurred and what the user should do.

Permissions-Based Framework. The Android platform utilizes a permissions-based framework [7] that runs at install-time and informs the users of the privacy that the app requires (up to 130 application level permissions). It has been shown that most mobile users are unable to comprehend the implications and finer details of each permission [8]. Permissions granting process has become a routine to most mobile users, who would indiscriminately agree to the set of permissions as a matter of habit without much thought.

4. APPROACH FRAMEWORK

We propose the following approach shown in Figure 1 to demonstrate the feasibility of the forensic study of private data leaks and its potential applications (apps). First, we instrument the mobile system to collect user traces with TaintDroid as our leak detector. Second, we perform extensive testing of selected mobile applications to collect the traces and leaks. Lastly, various analyses are carried out to produce the causal relations between user actions and leaks.

4.1 App Testing and Log Collection

To identify user actions that may cause private data leaks, we perform extensive testing on the app that uncovers its behaviour to collect as much user actions and leak reports as possible for analysis. In order to generate all possible interactions in a systematic way, we could employ an automated testing approach for mobile apps. There are a number of studies [9–14] for this purpose. However, we have realised that no publicly available tool fits our needs for a scalable automatic test on Android apps. These tools are generally

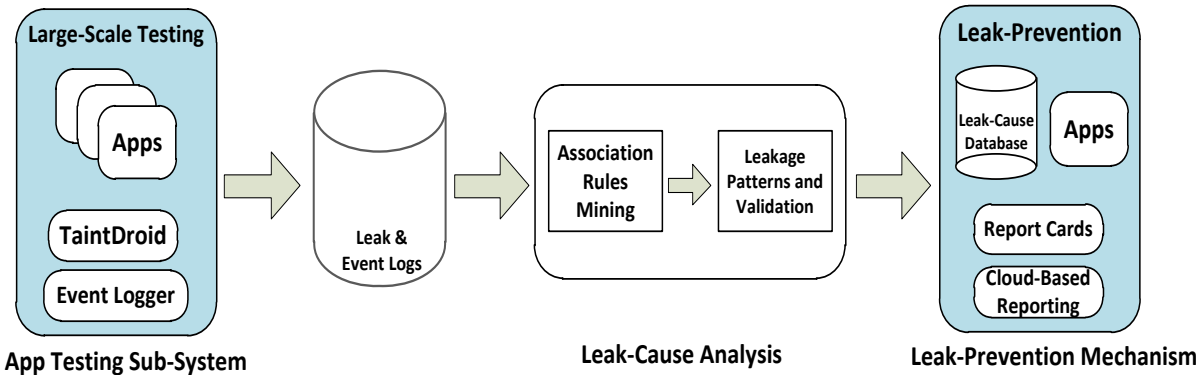

Figure 1: Approach Overview

tested in a graphical environment, one-by-one. Automatic tools like Android Monkey tool [15] are usually use for stress testing which does not provide a systematic way of capturing leak behaviours. Other tools, provide test frameworks for developers to write or record, but not automatically generate test cases [16–18]. For these reasons, we opted to use manual testing for our preliminary study.

We performed manual testing in two phases. In the first phase, we perform extensive testing to ensure high coverage on all GUI components (a.k.a. widgets). We test each app at least three times. We utilized a manual "breath-first" search for each test case. In each iteration, we start off with a 'root' app layout page, and attempt to manually trigger as many on-screen widgets as possible, trying to cover all possible widgets. If a widget click brings a new page layout, we retract to the previous root page if all widgets have not been clicked yet. The root page is shifted to a new app page when all widgets have been covered. There are situations when it might be impossible to click on all widgets on a root page (e.g., multiple individual news items on a news app). For such situations, we click on only a small number (4 to 5) of widgets which are representative of that particular page. Unless there are obvious sequencing requirements for triggering a particular functionality of the application (e.g., we often need to log in first before we can post a comment in the application), our manual testing does not explicitly consider the sequencing of user events that we trigger. TaintDroid [2], is used as a leak reporting tool. However, it only reports what private data is leaked, but does not report what user actions trigger the leak. Thus, we instrumented the mobile system (with TaintDroid) to report user events via the system logs. We also develop a logger to capture all user traces and leak events reported in the logs. In the second phase, we perform more intensive testing based on knowledge from analyzing the leak causes reported from the first phase described in Section 4.2. Using preliminary leak-cause data that identified leaky widgets from phase 1, we further tested the app with an additional three more rounds focusing on these leaky widgets. The logs from these additional runs are also collected to further enhance the cause analysis results in Section 4.2.

4.2 Leak-Cause Analysis

In leak-cause analysis, we establish linkages between user actions and actual leaks. We utilize *association rule mining* [4–6] to identify the *correlation* based on how frequently two concerned events (a particular user action and a particular leak report in our setting) happen together within certain time windows (10 seconds in our preliminary study). A user action can be within multiple overlapping time windows corresponding to different leaks. Our assumption and intuition for applying this technique as follows: the leaking behaviour of an application stays constant with time, similar behaviour is expected to occur repeatedly if sufficient amount of logs are collected. Thus we can extract patterns and identify causes of various leaking behaviour. For example, if a news app is reported to leak IMEI as many times as the "Load More News" button was touched by the user, we may infer that the user action of touching this button is highly correlated with the leak of the IMEI in this app.

The outputs of such analysis are expressed as *association rules* whose left sides indicate the causes of the leaks on the right sides. For example: touch of "Load more news" $\rightarrow$ Leak of IMEI. We use WEKA [19] to perform rule mining on the logs. We call such leaks associated with mined user actions *user-triggered leaks*. For our preliminary study, we utilized the Apriori algorithm available in WEKA to validate such association rules with absolute support and confidence thresholds of 2 and 0.5 respectively. The thresholds were tuned based on a small sample of about 10 apps and fitted reasonably well with our test logs.

There are also cases when *no* association rules with strong supporting evidences (i.e., occurring enough numbers of times during our testing) are identified for a particular leak. In such case, we further look at the time when such a leak happens to classify it accordingly:

- If the leak only happens once when an application starts, it is still a leak caused by a user action (launching the application), and we classify it as a *start-up* or *one-time* leak.

- If the same leak happens repeatedly during the run of an application, with an approximately constant time interval, we classify it as a *periodic* leak, as it is triggered by a user action (launching the application) but not tied to any user action during the run of the application.

4.3 Leak Prevention

Once we have identified leak causes for an application, we can provide such useful knowledge to users in various ways to help prevent leaks in a more user friendly way. We can generate various kinds of "Report Cards" for an application either in the app store, or during run-time to report possible leaks to users as illustrated in Section 5.4.

5. EMPIRICAL SURVEY

To demonstrate the necessity and benefits of mobile forensics of private data leaks, we perform an empirical evaluation of selected popular applications in the Google Play application store [20] based on our proposed approach framework. In this evaluation, we report various types and distributions of leak behaviors uncovered from the subject applications, and report the cause analysis results based on association rule mining.

5.1 Experimental Setup

Applications. We have sampled the top 10 applications from the 22 categories(From Nov 2012)from Google Play store (Total of 226, 6 additional apps). Note that the number of categories was recently changed to 26 at the start of 2013, but as our experiment was performed in November of 2012, our test captured only the 22 pre-updated categories. The 6 additional applications were added to make up for cases when some apps randomly crashed, possibly due to compatibility issues with the TaintDroid tool.

Testing environment. Our testing has all been performed on a Google Nexus One phone running Android v2.3.4 with our custom ROM that includes TaintDroid and our instrumentation to capture user actions.

Testing duration. For each application, we install and run the app (without other apps in the background) as described in Section 4.1. Each test of an app lasts for about 10-15 minutes.

5.2 Taxonomy of Leak Causes

Ratio of Leaky Applications. Out of the 226 applications studied, 121 applications are not leaking private data (non-leaking apps). Among the 105 apps (46.5%) leaking privacy data, 64 apps (28.3%) were found to leak private data due to user actions on application widgets. With the cause analysis as described in Section 4.2, we notice that an application can leak data in various ways: (1) User-Triggered Leaks (identified by association rules), (2) Start-Up or One-Time Leaks, and (3) Periodic Leaks.

The distribution of the 105 apps with various leak causes is shown in Figure 2. 32.4% (34) of the leaky apps leak

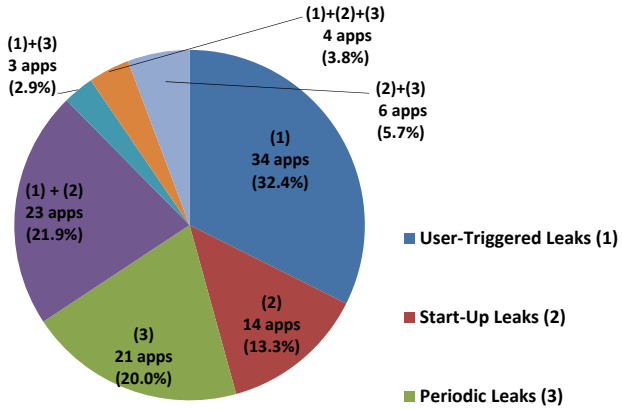

Figure 2: Distribution of Various Leak Causes.

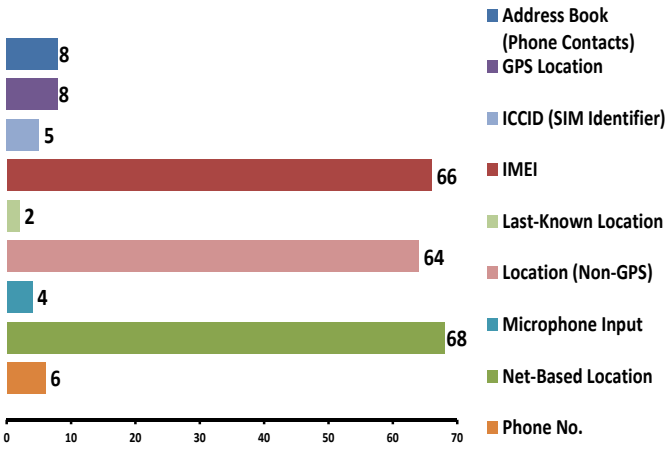

Figure 3: Distribution of types of leaked data.

private data solely due to (1) User-Triggered actions on widgets. 13.3% (14) leak solely on (2) Start-Up. 20% (21) leak data solely in a (3) Periodic fashion. 21.9% (23) can leak data by either (1) User-Triggers or (2) Start-Up. 2.9% (3) leak data either by (1) User-Triggers or (3) Periodically. 3.8% (4) applications leak data by any of the three means. 5.7% (6) have no User-Triggered leaks, but leak data on either (2) Start-up or (3) Periodically.

5.3 Distribution of Various Private Data Leaked

Figure 3 shows the distribution of various types of private data that is leaked by our subject applications. 9 different leaked data types have been found: {Phone Contacts, GPS Location, ICCID (SIM card identifier), IMEI, Last-Known Location, Location (Non-GPS), Microphone Input, Net-based Location, Phone Number}. It is noticed that a majority of leaks are of 3 types: IMEI, Location (Non-GPS), and Net-Based location. As most of the processes in the applications required an active internet connection, it is unsurprising perhaps that much of the private data leaked is of

their Net-Based location, which is the coarse-grained geographical location obtained from mobile users's IP addresses. However, it is surprising that many applications are leaking IMEIs and non-GPS location, which is the user location obtained from cellular towers, to external servers. From our observation, IMEIs are often used (but possibly unnecessary) as a unique identifier to link mobile users to activities such as user feedback, comments, and record-keeping for access when requesting server-based information by the applications.

5.4 Leak-Cause Database and Applications

This section presents our ideas on using leak causes in a user-oriented way to help reduce unwanted leaks.

5.4.1 Creation of A "Leak-Cause" Database

Establishing a database that contains all leak causes for various applications can help to enhance the usability and trustworthiness of an application store, as discussed in Section 2.2. We thus have constructed a leak-cause database for all of the Android applications under test. Table 1 shows sample causes from this database for 4 applications: 'Dictionary', 'HungryGoWhere', 'Gmail', and 'MessengerWithYou'. The database contains relevant information pertaining to the leak behaviours of the apps: type of private data leaked, and widgets identified by our association rule mining step that may be triggered by user actions to leak data. For example, The rule 4) for 'Dictionary.com' indicates that touching the 'ImageView #3' widget after touching 'Button #1' would leak users' locations. The leaky widgets can be uniquely identified by 8-digit Java hash code identifiers and (x,y) layout coordinates, with their corresponding leak type associations. A total of 647 leak causes were mined from the 64 apps with user-triggered leaks.

By far, we have not observed any leak that is only triggered by a sequence of more than one user actions. Intuitively, such cases should exist. For example, a user may need to touch a series of buttons in an app to trigger the display of a widget and the mere display of the widget would leak data. Our manual testing process may also be a cause of the no-show of such cases in our study since, as mentioned in Section 4.1, we did not consider the ordering among widgets. We plan to further investigate such cases in our future work when automated testing tools can help to exercise an apps more extensively.

The accuracy of mined leak causes would impact the usability of such a database, and thus we perform further validation of the leak causes to improve our confidence on the preliminary analysis results. The validation testing was a separate phase on top of the manual test-cases described in 4.1. For all the 64 applications with User-Triggered leaks, we verify each of the leak causes by rerunning the application and performing the user actions stated in the cause and observing whether the leak monitoring reports the leak or not. We define the *accuracy* to be the percentage of leak causes that have been verified to be correct, out of the total number of leak causes.

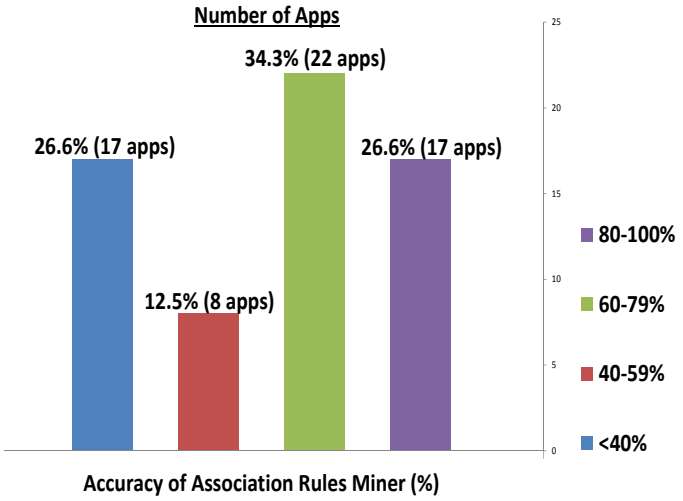

Figure 4: Distribution of Leak-Cause Accuracies.

The number of leak causes (association rules) found and the number of causes verified for the 4 example applications are also shown in Table 1. In the 'Dictionary' app for example, 18 rules were found, out of which 13 were verified, and thus its accuracy is 72%.

The *leak cause accuracies* for the 64 applications are illustrated in Figure 4. We observe that our leak-cause analysis is very accurate (80-100%) for 26.6% of the applications. Overall, the miner achieves an average accuracy of about 60% across the 64 applications.

Our preliminary study does indicate that false negatives do exist. However, this study does not investigate false negatives since the full search space of leaky widgets is unknown to us. We plan to complement our approach with static analysis to obtain a better search space of leaky widgets in our future work.

Causes of Mining Inaccuracies: We highlight a number of causes for the inaccuracies in our current rule mining which we can improve in future work. We observed that multiple non-leaky widgets are often situated in close proximity to a leaky widget on the screen layouts. This causes false positives due to these non-leaky widgets showing up as false candidates in the Association Rules mining. Co-occurrences of start-up/periodic leaks together with user-triggered leaks can confuse the rule mining and cause false positives. Also, there are situations in which widgets might leak infrequently (e.g. only once before stopping), and this contributes to false negatives. It is also known that TaintDroid may generate false positives due to the usage of tag aggregation at various points in the system to reduce storage costs. All these reasons contribute to mining inaccuracies.

5.4.2 Creation of Visual Leaky Layouts

For easier notification and illustration of leak causes and

No.	App Name	Leaks Found	Association Rules(Sample)	# of Rules Found	# of Verified Rules
1	Dictionary.com	1) Location	1)(ImageView #1) → (Location)+(Net-based Location)	18	13
		2) Net-based Location	2)(ImageView #2) → (Location)+(Net-based Location)		
			3)(TextView #1) → (Net-Based)+(Location)		
			4) (Button #1) + (ImageView #3) → (Location)+(Net-based Location)		
2	HungryGoWhere	1) ICCID (SIM Card Identifier)	1)(TextView #1) → (ICCID)+(IMEI)	10	10
		2) IMEI	2)(LinearLayout #1) → (ICCID)+(IMEI)		
			3)(LinearLayout #2) → (ICCID)+(IMEI)		
			4) (RelativeLayout #1) → (ICCID)+(IMEI)		
3	Gmail	1) Address Book (Phone Contacts)	1)(LinearLayout #1) → (Address Book)+(Phone Book)	6	5
		2) Phone No.	+(GPS Location)+(Net-based Location)		
		3) GPS Location	2)(Button #1) → (Address Book)+(Phone Book)		
		4) Net-based Location	+(GPS Location)+(Net-based Location)		
4	Messenger WithYou	1) IMEI	1)(Button #1) → (IMEI)	6	4
			2)(Button #2) → (IMEI)		
			3)(Button #3) → (IMEI)		
			4)(Button #4) → (IMEI)		

Table 1: Sample Association Rules in Our Leak-Cause Database

potential uses of the rules for end users, we create visual presentations of the leak causes by overlaying semi-transparent shaded-boxes on the top of leaky widgets in applications. The semi-transparent overlays may be shown at run-time to warn users about potential leaks. Screen captures could also be available through app stores for users' reference whenever they want to download, install, upgrade the application. Such information is much more user friendly and easier for users to digest and help them to be better informed and make better decisions. These visual overlays provide a preview on what can be implemented for permission analysis & user-studies as well as potential user notification methods as described in section 2.

6. DISCUSSION & FUTURE WORK

In this paper, we presented our case for a user-centric privacy protection mechanism using leak analysis in mobile forensics. Although, we have shown that our approach demonstrates the impact that it can have to users there are still some considerations. Firstly, our approach relies on TaintDroid as a leak detector. However, TaintDroid uses dynamic analysis through data-flow and does not track privacy data leaking through control-flow. Hence, static analysis can also be utilized to improve the accuracy of leak-cause associations, as well as mitigate blind spots in the dynamic analysis. Secondly, the usage of association mining and the data does not represent the full scope of data-mining techniques. We plan in future work to adopt other approaches to leak-cause generation. Lastly, Our taxonomy and testing was performed on an older version of TaintDroid (running on Android v2.3.4). A new version of TaintDroid running on v4.1 has been released by its developers, this new version allows us to test a larger number of apps (including games) from the Google Play. However, we deferred testing with this version to the future.

For the leak-cause database in user notifications, difficulties in issuing widget-level warnings to users might exist. Such warnings might be too fine-grained or numerous to be effective or pleasant to the user. This is especially true for apps with complex UIs or a large number of leaky widgets. We plan to conduct system-level user-studies in the future to investigate and resolve this issue. In addition, some leaks may be legitimate for the functionality provided by the app, but our current database cannot tell the legitimacy of a leak. AppProfiler [21] has analyzed users' opinions about how applications affected their privacy. We also plan to carry out a similar study on the (il)legitimacy of data leaks in conjunction with the study on the usability of our system.

In our future work, we planned to realised the motivational applications mentioned in section 2, the Enhanced App Store and the Interactive Privacy Leak Reporter with a comprehensive user study to validate its usefulness.

7. CONCLUSION

In this paper, we present the case for mobile forensics of private data leaks in mobile applications. Through our empirical study of more than 220 Android applications from Google Play based on our proposed approach framework, we show that there is a high percentage (46.5%) of applications that leak various types of private data, that leaks may be triggered by different conditions (user actions on some in-app widgets, start-up or one-time, or periodic), and that our leak-cause analysis can identify the actual leak causes with reasonably promising accuracies.

With the construction of a leak-cause database, we also demonstrate that the information accumulated there is easier

for users to understand than existing leak monitoring tools, and can be used to develop more user-friendly mechanism for privacy protection. Such mobile forensics of leaks can also be enabled in a large scale, with appropriate testing infrastructures and/or user-feedback channels, to identify leak causes from various application, either in the application stores or in the mobile systems, before installation or during runtime, to aid better privacy protection.

8. ACKNOWLEDGEMENTS

This work is supported in part by the National Research Foundation Singapore under its IDM Futures Funding Initiative, and administered by the Interactive & Digital Media Program Office, Media Development Authority. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency, or Singapore Management University.

9. REFERENCES

[1] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the 5th international conference on Trust and Trustworthy Computing (TRUST)*, pp. 291–307, 2012.

[2] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *OSDI*, pp. 1–6, 2010.

[3] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting privacy leaks in iOS applications," in *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, (Santiago, Chile), pp. 487–499, Sept. 1994.

[5] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3 ed., 2011.

[6] L. Dehaspe and H. Toivonen, *Discovery of Relational Association Rules*. Springer-Verlag, 2000.

[7] T. Vidas, N. Christin, and L. Cranor, "Curbing android permission creep," in *Proceedings of the Web*, vol. 2, 2011.

[8] A. 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*, p. 3, ACM, 2012.

[9] C. Hu and I. Neamtiu, "Automating GUI testing for android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test (AST)*, pp. 77–83, 2011.

[10] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *FSE*, pp. 59:1–59:11, 2012.

[11] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using GUI ripping for automated testing of android applications," in *ASE*, pp. 258–261, 2012.

[12] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "SmartDroid: An automatic system for revealing UI-based trigger conditions in android applications," in *Proceedings of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, pp. 93–104, 2012.

[13] K. Lee, J. Flinn, T. Giuli, and C. Peplin, "AMC:verifying user interface properties for vehicular applications," in *MobiSys*, 2013.

[14] W. Yang, M. Prasad, and T. Xie., "A grey-box approach for automated GUI-model generation of mobile applications," in *FASE*, 2013.

[15] "Android application exerciser monkey." `http://developer.android.com/tools/help/monkey.html`.

[16] "monkeyrunner." `http://developer.android.com/tools/help/monkeyrunner_concepts.html`.

[17] "Robotium." `http://code.google.com/p/robotium/`.

[18] "Robolectric." `http://pivotal.github.com/robolectric/`.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[20] "Google play." `https://play.google.com`.

[21] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 221–232, ACM, 2013.