Unnecessary Information Releases in Mobile Applications

ABSTRACT

Abstract

1. INTRODUCTION

Mobile applications enjoy almost permanent connectivity and ability to exchange information with their own backends, third-party servers and other applications installed on the same device. Recent studies show that during this information exchange, applications often release sensitive information about their users, such as location, phone number or unique device id [3, 2, 5]. With privacy being an increased concern, followup works investigate feasibility of obfuscating or blocking completely such release of sensitive information [4]. Yet, users often deliberately trade their privacy for receiving a desired service from an application, e.g., finding friends within a certain radius from their current location. In such cases, preventing the release of the user's location would render an application privacy-preserving but effectively useless.

Once an application gains access to the user's sensitive information, it can, in addition to utilizing the information for the purpose assumed by the user, also release it to unauthorized third parties. Permission systems of contemporary mobile platforms cannot prevent such scenarios as they only require applications to declare the *type* of information they want to access, not the *purpose* of the access. Existing information leakage detection techniques for mobile applications [3, 2, 1, 5, **?**] also cannot be employed for that purpose: they flag each identified release of sensitive information as a potential privacy breach, regardless of its designated use. The non-trivial task of classifying the purpose of information release is then left to the user.

This paper takes a first step towards performing such classification automatically. Specifically, we focus on identifying information release statements that are *unnecessary* for the application execution, i.e., those that do not contribute to the observable application behavior. We empirically investigate the nature of information releases in ten top-popular Android application from the Google Play store, exploring the impact of connection denial at each executed statement on the overall behavior of the analyzed application. We then design a static application analysis technique for identifying information release sites that can be deemed unnecessary: their failure has no impact on the behavior of the application.

Information Release in Android Applications. We conduct an empirical study whose goal is to explore and quantify the amount of unnecessary connections performed by Android applications. We focus on the three most common connection types: HTTP, socket and RPC. The first two are used to communicate with various backend servers – the application's own and third parties'; the last one is used to communicate with other applications and services running on the same device.

Our study is dynamic in nature and is performed in three phases. In the first phase, we establish a baseline application behavior. Similarly to the approach in [4], we record a script triggering the application functionality via a serious of interactions with the application's user interface. After each interaction, we capture a screenshot of the device to record the application state.

In the second phase, we instrument the application to log information about triggered connection statements. The instrumented apk is then installed and executed on a mobile device using the recorded script.

In the third phase, we investigate the impact of each triggered connection on the overall behavior of the application. Specifically, for each triggered connection, we produce a version of the application with the corresponding connection being disabled while all remaining connections stay intact. Disabling is achieved by replacing the connection statement with the statement that indicated connection failure, e.g., that throws an exception that occurs when the connection fails due to the device being put in airplane mode. We then install the modified application and run it using the previously recorded script. The screenshots documenting the execution of the modified application are compared to those of the original one. We consider executions as equivalent if they result in screenshots that differ only in the content of advertisement information, messages in social network applications such as twitter, and the devices status bar. We also separately note connections that contribute to presenting the advertisement content, if teh analyzed application has any.

Our analysis reveals that an almost 90% of the connection statements exercised by the applications do not lead to any noticeable affect on the observable application functionality. Around 30% of these correspond to HTTP and socket communication. The rest correspond to RPC calls to internal services installed on the device: notably, but not exclusively, google advertising and analytics, which further communicate with external services. Moreover, in applications that present advertisement material, about 60% of the connections that do affect the observable application behavior are used for the advertising purposes only.

A manual byte code inspection of the identified "hidden" connections shows that failures are often either silently ignored by the application, e.g., with an empty exception *catch* block, or written to the log file without being propagated to the user. This behavior is indicative for the connections being *unessential* for the application behavior and, in some cases, even harmful. In fact, the only application out of ten we analyzed that leaked the unique device id to the internet did that via such a hidden connection; identifying and blocking the connection eliminated the information leakage without affecting the application behavior and other information exchange operations that the application performed.

Detecting Unnecessary Connections. Inspired by the findings, we devise a static application analysis technique for detecting cases when connection failures are "silently" ignored by the application, i.e., when information about a connection failure is not propagated back to the end user. Our technique search for cases when exceptions resulting from connection attempts are caught by an applications without giving any visual message to the user. **TDB: designed to be conservative etc.**

We evaluate the technique on the "truth set" produced manually during the above empirical analysis and show that it is able to identify xx% of the "silent" information releases (xxx cases for all analyzed applications), introducing only yy false-positive and zz false-negative results.

Applying the analysis on additional zz top-popular applications from Google Play reveals that ww% of connection sites established by these applications can be deemed unnecessary. **TBD**

Significance of the Work. Our work focuses on begun mobile applications available in popular application stores and installed by thousand of users. By identifying and highlighting application functionality hidden from the user, it aims at improving transparency and, ultimately, quality of these applications.

We are largely inspired by the positive trend of improvement that we recently observed in the application development community. One of its manifestation is the increased awareness to privacy considerations, which is likely affected by multiple studies, including our own, that highlight potential privacy violation in applications from popular application stores [3, 2, 5]. Indeed, we observed that newer versions of the analyzed application in many cases no longer suffer from the previously identified privacy breaches. This improvement trend reassures and further emphasize the significance of this work, as an additional step towards achieving better application quality and increasing trust between developers and their users.

Contributions. The paper makes the following contributions:

1. It sets a new problem of distinguishing between essential and unessential release of information by mobile applications in an automated manner. This problem is orthogonal and complementary to that of identifying sensitive information flow, which was the focus on numerous earlier works. Highlighting unessential information releases in existing mobile applications is expected to improve transparency and contribute to the overall quality of the field.

2. It proposes a dynamic approach for detecting unnecessary releases of information in Android applications which does not does not require access to the application source code. The approach relies on interactive injection of connection failures and identification of cases in which injected failures have no visible affect on the observable application functionality.

3. It provides empirical evidence for the prevalence of such unnecessary connection in real-life applications. Specifically, it shows that 90% of the connections attempted by 10 top-popular free applications on GooglePlay fall into that category.

4. It proposes an static technique that operates on application binaries and identifies unnecessary connection – those where failures are not propagated back to the application's user. The precision and recall of the technique is xx and yy, respectively, when evaluated against the empirically established truth set. When applied on 50 top-popular free applications on Google Play, the technique is able to identify xx% of connections made by these applications as unnecessary.

The remainder of the paper is structured as follows. Section 2 describes the empirical study we conducted for gaining insights into the nature of information releases in mobile applications. Section 3 presents the static analysis technique designed for identifying unnecessary information releases. Section 4 discusses results of its evaluation on real-live examples. Section 5 discusses the limitation of our work. Section **??** presents the related work, while Section **??** the paper and discusses future work.

2. INFORMATION RELEASE IN ANDROID APPLICATIONS

2.1 Connection Sites

List all considered connection types and describe how blocking is done.

2.2 Methodology

Discuss the approach of blocking connections one-by-one, and describe how application equivalence was established.

2.3 Subjects

List all applications we analyzed and describe how these were selected.

2.4 Results

84% of connection sites are not triggered dynamically

Out of those triggered, 90% are deemed unnecessary.

In 3 cases when teh applications contain ad material, xx% of connections that were deemed necessary were use for advertising

3. DETECTING UNNECESSARY CONNECTIONS

4. EVALUATION

5. LIMITATIONS AND THREATS TO VALIDITY

Cannot handle cases when two blocked points depend on each other.

Limited number of subjects.

. only works for cases when the application is design to handle disconnect

6. RELATED WORK

7. CONCLUSIONS AND FUTURE WORK

8. REFERENCES

[1] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'14)*, 2014.

[2] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Proc. of the Network and Distributed System Security Symposium (NDSS'11)*, 2011.

[3] 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 *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, pages 1–6, 2010.

[4] 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 *Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 639–652, 2011.

[5] O. Tripp and J. Rubin. A Bayesian Approach to Privacy Enforcement in Smartphones. In *Proc. of the 23rd USENIX Conference on Security Symposium (SEC'14)*, pages 175–190, 2014.