

Thomas Brüggemann

Master Thesis
im Fach Allgemeine Wirtschaftsinformatik

Automating the privacy risk assessment for mHealth apps

Themensteller: Jun.-Prof. Dr. Ali Sunyaev

Vorgelegt in der Masterprüfung
im Studiengang Information Systems
der Wirtschafts- und Sozialwissenschaftlichen Fakultät
der Universität zu Köln

Köln, Februar 2016

Contents

Index of Abbreviations	III
1. Problem Statement	1
2. Objectives	2
3. Definitions	2
3.1 Mobile health applications	2
3.2 Static code analysis	2
3.3 Decompilation	2
4. Methods	2
4.1 Automating the privacy risk assessment	2
4.2 Evaluation	5
5. Structure	5
6. Expected Results	5
7. Problems	5
References	6

Index of Abbreviations

API	Application programming interface
APK	Android application package
DRM	Digital rights management
JAR	Java archive
JVM	Java virtual machine
mHealth	Mobile health

1. Problem Statement

The market for mobile smart device applications (apps) is growing extensively in recent years. It is increasingly easier for small companies or even single developers to create unique apps that reach millions of users via app stores. This market growth also effected mobile health (mHealth) apps. More and more apps are available that support the users in solving their health related issues and information deficiencies. Users are asked to input their personal information in order to tailor the app to their custom needs. The users are asked to expose vulnerable information about their health status while it remains mostly unclear how and where the data is processed, stores or tossed along.

The information about privacy practices of the app providers should be found in the privacy policy document provided by the app provider. Processing these privacy policies- requires a higher level of education and time to read through large documents of text to find the relevant information. Additionally, the important information is hidden in legal language or insufficiently addressed, if at all.¹

Aside from the data usage beyond users control, it is also challenging for users to assess what kind of privat information an app asks for, in order to tailor the app experience. Users have to download the apps of interest and try them out, before it becomes clear what information is used. This leads to low inter comparability between apps. If users are looking for specific functionality in a mHealth app, it is challenging to find the app that offers the desired functionality at an acceptable privacy risk. Even if users would persue the goal to find and compare mHealth apps of similar functionality, the high volume of apps available on the app stores² makes it not feasible to review all of them by hand.

Resolving the challenges in evaluating the privacy risk of mHealth apps, before usage and of large volumes, will result in an improved decision making process for users. It also reduces the danger of exposing vulnerable information. A way to assess privacy risks of mHealth apps would be to automate the review of each individual app. Automating the review process for large scale app assessments has the potential to grow new privacy evaluation service markets.³ Automating the app assessment for potential privacy risk

¹ Dehling, Gao, Sunyaev (2014), p. 11

² Enck et al. (2011), p. 1

³ Enck et al. (2011), p. 14

factors can be done by downloading and analyzing the source of each app to trace data leaks. Static code analysis is used in the field of informatics to analyse application source code to detect faults or vulnerabilities.⁴ A static code analysis could potentially be used to assess the privacy risks that mHealth apps pose.

The automated process of assessing the privacy risk helps to reduce the costs of reviewing each individual app and enhances the information experience users get while researching mHealth apps. Additionally, it exposes new possibilities for research in the privacy risk area. The research could be conducted on providing solutions and best practices for minimizing the privacy risk of apps. It is unclear if, and to what degree, the concepts of static code analysis and privacy risk assessment can be combined in order to automate the app assessment. This leads to the research question of this master thesis. How and to what degree can the privacy risk assessment of mHealth apps be automated?

TODO: PREVIOUS RESEARCH Previous research in this field reveals...

2. Objectives

3. Definitions

Certain terms are used in the remainder of this thesis that need to be defined:

3.1 Mobile health applications

3.2 Static code analysis

3.3 Decompilation

4. Methods

4.1 Automating the privacy risk assessment

In order to automate the privacy risk assessment of mHealth apps via static code analysis, it is necessary to gain access to the source code of the apps. While uploading a new app to the Apple AppStore, Apple's digital rights management (DRM) system encrypts the binary file in a way that makes recovering the source code difficult. There are approaches

⁴ Baca, Carlsson, Lundberg (2008), p. 79

on decompiling the Apple app binary back into its source code. These approaches involve unlocking and jailbreaking ⁵ an Apple iPhone or iPad, which is a violation of the Apple terms of service and therefore forbidden⁶

The Google PlayStore on the other hand hosts Android application in APK containers that are non encrypted and allow for decompilation back into the original source files. In order to automate the download process of APK files to our local computer, we found a hidden Google API that reveals access to the APK files from the PlayStore. The API can be queried by sending an Android device id along, effectively pretending to be a requesting Android device. The result of the query is a binary APK file.

W. Xu, Liu (2015) provision a repository of apps listings that are extracted from the Google PlayStore within the categories "Health & Fitness" and "Medical". Due to the obstacles of gaining access to the Apple iOS binary files, we restrict the dataset to the available Android apps and conduct our automated privacy risk assessment on these apps. The W. Xu, Liu (2015) dataset contains 5379 Android apps. We exclude paid apps from further proceeding, since downloading paid apps would charge the credit card of the authors, since we use the same API that the Google PlayStore itself uses to purchase apps from the store on Android devices. Downloading all 5379 apps would result in purchasing costs of 19,904.24 US-dollars. Therefore we reduce the dataset to the 3180 free apps, which is still 59.1% of the initial dataset. We will download as many apps of these 3180 as possible. Potential limitations in downloading could arise from the fact that Google blocks requests on the API as soon as the system detects overly use or misuse. Downloading 3180 apps at a time will likely trigger these security mechanisms of Google, since a normal user could never download that many apps in the same timeframe from the PlayStore.

As soon as the APK files are available offline, the decompiling phase of the study will begin. Our decompiling process consists of four steps.

The first step is to recover a java archive (JAR) file from the APK file. JAR files contain a collection of .class files. These .class files hold Java bytecode that can be interpreted

⁵ Jailbreaking revers to the action of removing ... Kweiler (2010), p. 1

⁶ Kweiler (2010), p. 1

by the Java virtual machine JVM at application runtime.⁷ In order to extract the Java bytecode .class files, we use the command line tool *dex2jar*⁸. The abbreviation **DEX!** stands for *Dalvic executable* and refers to the binary collection of compiled Java classes within the APK file.⁹

Step two includes the actual decompiling phase of the Java bytecode back into readable .java files. A .java file contains exactly one Java *class* in humanly readable Java code. Enck et al. (2011) developed their own Java decompiling toolchain in order to assess security issues and used within their toolchain used a tool called *Soot*. *Soot* optimizes the decompiled code to improve the readability by humans. Aside from using *Soot*, Enck et al. (2011) propose the idea to use different Android decompiler tools, such as *JD* or *Fernflower* in future research, since *Soot* did not perform well in all cases.¹⁰ We will use *Fernflower* as the decompiling tool for our study, since it evidentially outperformed *Soot* in a further evaluation by Enck et al. (2011).¹¹ The result of this second step is a directory full of .java files that represent the source code of an APK app.

The decompiling process delivers source code files that lack any formatting. For the case of manually validating the output of the automated privacy risk assessment we will have to take a look into the source code files. To make manual code inspections easier to read, a tool called *astyle*¹² will be used. *astyle* sets appropriate levels on indenting to the source code.

⁷ The previous two sentences follow Enck et al. (2011), p. 2

⁸ <https://github.com/pxb1988/dex2jar>, visited 02/03/2016

⁹ See L. Xu (2013), p. 6

¹⁰ For the previous three sentences, see Enck et al. (2011), p. 5

¹¹ Enck et al. (2011), p. 6

¹² <http://astyle.sourceforge.net/>, visited 02/03/2016

4.2 Evaluation

5. Structure

6. Expected Results

7. Problems

So far there are no open questions or problems. Though, a problem could arise from the fact that the core of this thesis relies on a undocumented Google API for downloading the APK files of the apps. If this API is shut down or somehow secured from open usage, it would not be as easy to gather the needed APK files for the static code analysis as it currently is.

References

Baca, Carlsson, Lundberg (2008)

Dejan Baca, Bengt Carlsson, Lars Lundberg: Evaluating the cost reduction of static code analysis for software security. In: Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security - PLAS '08. 2008, p. 79

Dehling, Gao, Sunyaev (2014)

Tobias Dehling, Fangjian Gao, Ali Sunyaev: Assessment Instrument for Privacy Policy Content: Design and Evaluation of PPC. In: WISP 2014 Proceedings. 2014,

Enck et al. (2011)

William Enck, Damien Ocate, Patrick McDaniel, Swarat Chaudhuri: A Study of Android Application Security. In: USENIX security Nr. August, 2011, pp. 21–21

Kweller (2010)

Abby Kweller: Jailbreaking and Unlocking the iPhone: The legal implications. In: Polymer Contents. Nr. 8, Jg. 27, 2010, pp. 480–523

L. Xu (2013)

Liang Xu. “Techniques and Tools for Analyzing and Understanding Android Applications”. PhD thesis. Citeseer, 2013.

W. Xu, Liu (2015)

Wenlong Xu, Yin Liu: mHealthApps: A Repository and Database of Mobile Health Apps. In: JMIR mHealth and uHealth. Nr. 1, Jg. 3, 2015, e28