Thomas Brüggemann

Master Thesis
im Fach Allgemeine Wirtschaftsinformatik

Automating The Privacy Risk Assessment Of 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 apps 3

3.2 Static code analysis 3

3.3 Decompilation 3

4. Methods 3

4.1 Automating the privacy risk assessment 3

4.1.1 Android decompilation 3

4.1.2 Static code analysis 5

4.2 Evaluation 7

5. Structure 7

6. Expected Results 7

7. Problems 8

References 10

Index of Abbreviations

API	Application Programming Interface
APK	Android Application Package
DEX	Dalvik Executable
DRM	Digital Rights Management
HTTP	Hypertext Transfer Protocol
JAR	Java Archive
JVM	Java Virtual Machine
mHealth	Mobile Health
XML	Extensible Markup Language

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 stores[2] makes it not feasable 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 analysisis used in the field of informatics to analyse application source code to detect faults or vulnerabilities.[4] A static code analysiscould 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: Related Work bei L. Xu (2013) abschauen Previous research in this field revealed...

2. Objectives

The primary objective of this study is to ascertain how or to what degree the assessment of privacy risk factors for mHealth apps can be automated. This will be done by implementing a software tool that downloads, decompiles and analyses the source code of mHealth apps for privacy risk factors. The implementation process might come to a stop at some point due to insurmountable obstacles. These possible obstacles will be documented as a result of this study.

As a secondary objective of this study, an evaluation of the privacy risk assessment tool, including an overview of the users' opinions regarding the potential impact of the tool on their mHealth app decision making, will be given.

3. Definitions

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

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

3.1 Mobile health apps

Mobile health mHealth apps are smartphone or tablet applications that support the users by enabling them to gather information and support about medical or health related issues.[5]

3.2 Static code analysis

Static code analysis refers to the analysis of an applications source code without actually executing the application. This technique is widely used to detect vulnerabilities or to validate the source code at programming time in e.g. the code editor software. [6]

3.3 Decompilation

Compilers transfer human readable programming code into machine code and therefore help humans write software applications in understandable text form. Decompilers retrieve the human readable programming code back from the compiled machine code. They aim at reversing the compilation process. [7]

4. Methods

4.1 Automating the privacy risk assessment

4.1.1 Android decompilation

In order to automate the privacy risk assessment of mHealth apps via static code analysis, it is necessary to gain access 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 on decompiling the Apple app binary back into its source code. These approaches involve unlocking and jailbreaking [8] an Apple iPhone or iPad, which is a violation of the Apple

[5] See Dehling, Gao, Schneider, et al. (2015), p. 1

[6] See Bardas et al. (2010), p. 2-3

[7] This section follows Nolan (2012), p. 1-2

[8] Jailbreaking revers to the action of removing ... Kweller (2010), p. 1

terms of service and therefore forbidden[9]

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 by the Java virtual machine JVM at application runtime.[10] In order to extract the Java bytecode .class files, we use the command line tool *dex2jar*[11]. The abbreviation

[9] Kweller (2010), p. 1

[10] The previous two sentences follow Enck et al. (2011), p. 2

[11] https://github.com/pxb1988/dex2jar, visited 02/03/2016

DEX stands for Dalvik Executable and refers to the binary collection of compiled Java classes within the APK file.[12]

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. [13] 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).[14] 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*[15] will be used. *astyle* sets appropriate levels on indenting to the source code.

In the fourth and last step of our decompilation process, a tool called *apktool*[16] is used to extract all resource files from the APK file. Resource files could be images or XML files that are not compiled into application code.[17]

4.1.2 Static code analysis

In order to analyse potential privacy risks, we are going to use a static code analysis tool. This tool will scan and parse the Java source code files and make them computer readable.

Brüggemann, Hansen (2016) identified six potential privacy risk factors that a mHealth app could pose and combined them into a privacy risk index formula. The six factors con-

[12] See L. Xu (2013), p. 6

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

[14] Enck et al. (2011), p. 6

[15] http://astyle.sourceforge.net/, visited 02/03/2016

[16] https://github.com/iBotPeaches/Apktool, visited 03/02/2016

[17] See L. Xu (2013), p. 5

tain three binary factors. The first binary factor is whether an app requires a login via username/email or a social media login service such as Facebook. The second binary factor is the question if the app uses secured HTTP connections to the servers it is communicating too. While the first ones can be assessed at a reasonable level of difficulty via static code analysis, it is challenging to do so for third binary factor: reasonableness of personal data collection. The reasonableness of personal data collection assessment in the study of Brüggemann, Hansen (2016) was based on usage observation of the app. This might not be feasible in a static code analysis. The non-binary factors include the categories of personal data that users have to input into the mHealth apps. Examples of personal data categories are: Medication intake, Symptoms, Vital values. The last two factors express the target to which the personal data is likely to be sent. This is split in two factors, one for unspecific data targets, which refers to the usage of advertisement banner services or analytics tools within the app. The other data target factor tries to observe immediate data connections after input of personal data. Brüggemann, Hansen (2016) observed that personal data might be sent to a server of the app provider, advertisement or marketing companies, social media networks (such as Facebook) or to research projects.[18]

Our study aims at identifying the privacy risk factors from Brüggemann, Hansen (2016) by scanning the source code of each app in our working dataset. In order to identify whether an app required the users to login, the source code will be scanned for text input fields that are labelled with the substrings ”login”, ”register”, ”password”. Additionally, the source code will be scanned for social network login buttons. They are identified by the button name ”com.facebook.login.widget.LoginButton” for Facebook login, ”SignIn-Button” for Google login and ”com.twitter.sdk.android.core.identity.TwitterLoginButton” for Twitter login functionality. Android uses a *RequestQueue* to dispatch HTTP requests from the main thread. We will scan the source code for requests that are added to the *RequestQueue* and trace the url, that is the request target, back, until we can check if the URL contains the HTTPS prefix. This indicates a secure connection. The URL targets that requests are made to also indicate the personal data target factor. Especially if variables that were assigned through user input fields can be traced until they might be sent

[18] This section follows Brüggemann, Hansen (2016), p. 1-99

out in a request. In order to identify the individual text fields that ask for user input, two source code scanning methods need to be used. The Android *EditText* classes can either be declared within the layout Extensible Markup Language (XML) file or added via Java code during application runtime. In both cases the label and ID properties of the *EditText* instance can indicate the semantics of the personal data that is requested from the users. As soon as we got an overview of the *EditText* labels, we are going to cluster them into the personal data categories proposed by Brüggemann, Hansen (2016). At this point it would be possible to train a naive Bayes classifier, to judge the personal data categories from the individual data input text field labels. To identify if analytics or advertisement services are used within the app, we can simply scan for the inclusion of corresponding libraries. For example the Google Analytics[19] library is used within Android source code by including the package *com.google.android.gms.analytics.GoogleAnalytics*.

After the implementation attempts of all privacy risk factors are completed

4.2 Evaluation

5. Structure

6. Expected Results

As a result of this master thesis, we expect the implementation process of the automated privacy risk assessment to be at least partially feasible and implemented. Automating the downloading process of APK app files from the Google PlayStore is possible to a certain amount, and will be provided. The decompilation process is set to gain at least a 94 to 98% code recovery rate.[20] Therefore we expect the majority of APK files to be decompilable. We expect the static code analysis for privacy risk factors to be success for the majority factors and will provide insights into the possible degree of detail.

Another expected result is an overview of the users' options on the usage of the tool. Depending on the implementation success it will reveal to be either useful or rather not useful to the users decision making process regarding mHealth apps.

[19] https://www.google.com/analytics/, visited 02/05/2015

[20] Enck et al. (2011), p. 5-6

7. Problems

After downloading the first 1081 apps via the undocumented Google API function, my Google PlayStore account was suspended and I was automatically logged out of my Google account in the PlayStore app on my tablet. It might be possible to circumvent this by creating another Google account and using that one on my tablet. The tablet is important, since the undocumented Google API is queried with the unique Android ID of the tablet. As long as the suspension is linked to my Google account and not the Android ID, a realistic chance exists to download more APK files.

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

Bardas et al. (2010)

Alexandru G Bardas et al.: Static code analysis. In: Journal of Information Systems & Operations Management. Nr. 2, Jg. 4, 2010, pp. 99–107

Brüggemann, Hansen (2016)

Thomas Brüggemann, Joel Hansen: “A Privacy Index for mHealth Apps”. Manuscript 2016

Dehling, Gao, Schneider, et al. (2015)

Tobias Dehling, Fangjian Gao, Stephan Schneider, Ali Sunyaev: Exploring the Far Side of Mobile Health: Information Security and Privacy of Mobile Health Apps on iOS and Android. In: JMIR mHealth and uHealth. Nr. 1, Jg. 3, 2015, e8

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 Octeau, 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

Nolan (2012)

Godfrey Nolan: Decompiling Android. 1. Edition, New York 2012

L. Xu (2013)

Liang Xu: “Techniques and Tools for Analyzing and Understanding Android Applications” 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