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

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 smartphone and tablet applications (apps) is growing extensively in recent years. It is increasingly easier for companies or even single developers to create unique apps that reach millions of users around the planet via digital app stores.[1] This market growth effected mobile health (mHealth) apps as well. More and more mHealth apps are available that support the users in resolving their health related issues and that try to patch information deficiencies. The data tools and information systems that have been used and maintained in hospitals for decades are recently carried out to distributed cloud services for virtually everyone to consume.[2]

Users are asked to expose their personal health related information, in order to receive a tailored app that fits their custom preferences.[3] The users reveal vulnerable information about their health status, while it remains mostly unclear how and where the data is send, processed and stored.[4]

The information about these privacy related practices of app providers and their offered apps should be stated in the privacy policy document, provided by the app provider. Processing these privacy policiesrequires a higher level of education and time to read through large bodies of text, to find the relevant information. Additionally, the important information is hidden in legal language or insufficiently addressed, if at all.[5]

Aside from the data usage beyond users control, it is also challenging for users to assess what kind of private information an app asks for, prior to the app usage. Users have to download the apps of interest and try them out, before it becomes clear what health related information is processed by the app and in what way. 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 pursue the task of finding and comparing mHealth apps

---

1 Previous two sentences following Enck et al. (2011), p. 1

2 He et al. (2014), p. 645

3 Chen et al. (2012), p. 2

4 He et al. (2014), p. 652

5 This paragraph follows Dehling, Gao, Sunyaev (2014), p. 11

of similar functionality, the high volume of apps available on the app stores[6] makes it overbearingly hard to review all of them by hand.

Resolving the challenges in evaluating the privacy risk of large volumes of mHealth apps, before usage, will result in an improved decision making process for users. Users will be able to make app usage decisions based on an actual privacy risk measure that empowers comparability. It also reduces the danger of exposing vulnerable information. A way to assess privacy risks of mHealth apps would be to automate the review process of each individual app.

A way to overcome the challenge of reviewing large amounts of mHealth apps is by automating the assessment. Automating the review process for large scale app assessments even has the potential to grow new privacy evaluation service markets.[7] The assessment automation can be done by downloading and analyzing the source code of each app and by tracing data leaks. Static code analysisis used in the field of informatics, to analyse application source code and detect faults or vulnerabilities.[8] A static code analysiscould be used to automatically assess the privacy risks that mHealth apps pose. It is unclear if, and to what degree, the concepts of static code analysisand privacy risk assessmentcan 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?

The automated process helps to reduce the costs of reviewing each individual app drastically 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.

1.1 Related Work

TODO: Related Work bei L. Xu (2013) abschauen Previous research in this field revealed...

---

6 Enck et al. (2011), p. 1

7 Enck et al. (2011), p. 14

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

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 to the furthest degree possible. The implementation process might come to a stop at some point due to insurmountable obstacles.

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:

3.1 Mobile health apps

Mobile health mHealth apps are smartphone or tablet applications that support the users by enabling them to gather health related information and support the consumer in medical or health related issues.[9]

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 during development in the sourcecode editor software. [10]

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. The

---

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

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

compilation process of an application is non reversible. Therefore, decompiling is a reverse engineering technique that outputs source code, similar to the original source code of the application, but with the same functionality. [11]

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 existing approaches on decompiling the Apple app binary back into its source code. These approaches involve unlocking and jailbreaking [12] an Apple iPhone or iPad, which is a violation of the Apple terms of service and therefore forbidden[13]

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 make use of an undocumented Google API that reveals access to the APK files from the Google PlayStore. The API can be queried by sending an Android device id, pretending to be a requesting Android device. It is used by the Google PlayStore internally and the result of the query is the binary APK file.

W. Xu, Liu (2015) provision a repository of app listings that were 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 5,379 Android apps. We exclude paid apps from further proceeding. Downloading paid apps would charge the credit card of

---

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

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

[13] Kweller (2010), p. 1

the authors, since we use the same API that the Google PlayStore itself uses to purchase apps on Android devices. Downloading all 5,379 apps would result in a purchase value of 19,904.24 US-dollars. Therefore, we reduce the dataset to the 3,180 free apps, which is still 59.1% of the apps from the initial dataset. We will download as many of these 3,180 apps as possible. Download limitations could arise from the fact that Google blocks requests on the API as soon as the system detects heavy use or misuse. Downloading 3,180 apps in a short period of time will likely trigger these security mechanisms, since a normal user could never download that many apps in the same timeframe from the PlayStore. Careful download rate implementation is necessary.

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.[14] In order to extract the Java bytecode .class files, we use the command line tool *dex2jar*[15]. The abbreviation DEX stands for Dalvik Executable and refers to the binary collection of compiled Java classes within the APK file.[16]

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. [17] 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).[18] The result of this second step is a directory full of .java files that represent the source code of an APK app.

---

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

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

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

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

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

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*[19] 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*[20] 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.[21]

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 contain 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

---

19 http://astyle.sourceforge.net/, visited 02/03/2016

20 https://github.com/iBotPeaches/Apktool, visited 03/02/2016

21 See L. Xu (2013), p. 5

(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.[22]

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, ”SignInButton” 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 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[23] 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

---

22 This section follows Brüggemann, Hansen (2016), p. 1-99

23 https://www.google.com/analytics/, visited 02/05/2015

4.2 Evaluation

The evaluation phase starts with an overview of the degree of privacy risk assessment automation that has been accomplished. The evaluation will be based on the feasability and obstacles detected in implementing the automated privacy risk assessment. The explored privacy risk factors will be combined into the privacy risk index weighted-sum equation, proposed by Brüggemann, Hansen (2016) and will be displayed within their user interface.[24] Using the Brüggemann, Hansen (2016) user interface allows the users to explore and compare mHealth apps in a table view format. It can be used in addition to downloading apps from the Google PlayStore.

In order to further evaluate the automated privacy risk assessment, we will expose 15 potential users to the user interface and let them discover our detected privacy risks. We will give an overview of the users options towards the presence of enhanced privacy risk information, that might change the users decision making process regarding the selection of mHealth apps.

5. Structure

Es folgt eine Erläuterung der Gliederung der Bachelorarbeit. In Klammern steht jeweils der geschätzte bzw. anvisierte Umfang des jeweiligen Kapitels.

1 Introduction	The introduction provides an overview of the problem statement, the objectives, the methods used and the structure of the thesis.
1.1 Problem Statement	Description of the research cycle of the master thesis to further stress the practice problem and its relevance, as well as the research problem and its relevance.

[24] Brüggemann, Hansen (2016), p. 99

1.2 Related Work	Overview of previous research and related work that has covered the topic of analysing mHealth privacy risks or even using automated techniques on gathering privacy risk information
1.3 Objectives	This section describes the primary objective of the thesis and the subsequent secondary objectives, that we will aim at reaching.
1.4 Methods	An overview of the research approach of this master thesis that reflects the main part of the thesis.
1.5 Structure	The formal structure of the thesis to provide the reader with an overview of the thesis.
2 Automated Privacy Risk Assessment	This chapter describes the main implementation phase of the thesis.
2.1 Privacy Risk Factors	Introduction of the privacy risk factors identified by Brüggemann, Hansen (2016) and a further explanation on their relevance to mHealth app assessments.
2.2 Implementation	Details on the implementation of the automated mHealth privacy risk assessment tool.
2.2.1 Download Phase	Description of the APK file discovery and download phase of the tool. While free Android apps are available for everybody with an Android smartphone, we want to feed mHealth apps into our automated system and therefore need to download the APK files automatically.
2.2.2 Decompilation Phase	This section provides information on the decompilation phase of the APK files and the methods used to regain the source code of Android apps. This is a crucial step, since the source code is our main and only resource to perform the static code analysis on, in chapter 2.2.3.

2.2.3 Static Code Analysis Phase	Information on the different approaches to gather information on the individual privacy risk factors, explained in chapter 2.1. We will try to identify via a static code analysis, if the mHealth app requires a login to tailor the user experience or not. Furthermore we want to find out if personal health data is collected and if so, what kind of data. Via source code analysis we aim at identifying the targets, where data is sent and if advertisement or click-analytics services are used.
2.3 Results	Explaining the outcome of the implementation step. How many apps could be automatically assessed? How many factors could be automatically identified? Was it possible to compute information on all of the privacy risk factors from chapter 2.1?
3 Evaluation	Evaluating the outcomes of this thesis by critically appreciating the results of the implementation and the opinions of potential users.
3.1 Evaluation of Implementation	Information on the feasibility of implementing an automated mHealth privacy risk assessment tool, the challenges and obstacles.
3.2 Overview of Users' Opinions	Further evaluation of the automated privacy risk assessment tool is given by presenting an overview of the users' options on the tool. We will expose 15 users with our novel tool and share their option on the implications this tool has on mHealth app decision making.
4 Conclusion	A critical reflection on the thesis. Are all the objectives reached and are the methods used coherent?
4.1 Limitations	Explanations in what way the research approach and execution was limited or constrained during this thesis.

4.2 Future Research	This section will provide hints to future researchers on how to continue or extend the current study in this thesis and further develop the implemented tool and its implications.

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.[25] 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.

7. Problems

So far there are no open questions or problems. Though, a problem could arise from the fact that the core of this master 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 analysisas it currently is. It might be possible to circumvent the usage of undocumented Google APIs by downloading the APK files manually or semi-automated on an Android smartphone. This will be a fallback solution.

---

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

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

Chen et al. (2012)

Connie Chen, David Haddad, Joshua Selsky, Julia E Hoffman, Richard L Kravitz, Deborah E Estrin, Ida Sim: Making sense of mobile health data: an open architecture to improve individual- and population-level health. In: Journal of medical Internet research. Nr. 4, Jg. 14, 2012, e112

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 An-

droid Application Security. In: USENIX security .... Nr. August, 2011, pp. 21–21

He et al. (2014)

Dongjing He, Muhammad Naveed, Carl A Gunter, Klara Nahrstedt: Security Concerns in Android mHealth Apps. In: AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium. Jg. 2014, 2014, pp. 645–54

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”. PhD thesis. Davis: PhD thesis, 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