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]   **He2014a** p. 645

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

[4]   **He2014a** 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

mHealth apps have been subject of various research studies, that aim at providing insights for developers as well as users into how private information is processed. First of all

---

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

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

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

Khalid et al. (2015) found out, that privacy issues are the number one most impactful user complaint, while using mobile apps.[9] This encourages research to address privacy risks and data leaks. He et al. (2014) analyze the top 160 free Android mHealth apps by potential attack points to users private data. They focus on the technical side of privacy risks, for example the storage of mHealth data in internal Android log files or the storage card used within the phone or tablet.[10] Mcclurg (2012) study Android data leaks on a technical level and apply decompilation tools to automate the leak detection process, rather than reviewing apps manually by hand. They found out that Android apps leak private information, e.g. the device identification to app provider servers.[11] This reveals decompilation to be a feasible assessment technique for privacy risks and data leaks. A similar method is used by Enck et al. (2011), as they develop their own Android decompilation tool to assess misuse of the phone identifier, location data and advertising or analytics networks.[12]. Their insight into the decompilation toolchain will act as a technical foundation for our thesis. Mitchell et al. (2013) step into the analysis of specifically mHealth apps and try to identify private user data that can be accessed unauthorizedly while the application is installed on a mobile device and after it was removed.[13] A detailed privacy leak detection in Android apps is conducted by Kim et al. (2012) by building a static code analysis tool, that is able to trace inputs, such as location data or the phone identifier to the point where this data is sent out to a remote server.[14] Brüggemann, Hansen (2016) identified six factors of privacy risk within mHealth apps and proposed an equation to combine those factors into an equation. Our study to combine the decompilation steps of Enck et al. (2011) with a detailed static code analysis that Kim et al. (2012) proposed and apply this on the automated identification of the privacy risk factors Brüggemann, Hansen (2016) identified.

---

[9]   Khalid et al. (2015), p. 5

[10]   For the previous two sentences, see He et al. (2014), p. 645-646

[11]   Mcclurg (2012), p. 1, 5

[12]   Enck et al. (2011), p. 1

[13]   Mitchell et al. (2013), p. 6-7

[14]   Kim et al. (2012), p. 1-2

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.[15]

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. [16]

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

---

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

[16]   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. [17]

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 [18] an Apple iPhone or iPad, which is a violation of the Apple terms of service and therefore forbidden[19]

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

---

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

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

[19]   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.[20] In order to extract the Java bytecode .class files, we use the command line tool *dex2jar*[21]. The abbreviation DEX stands for Dalvik Executable and refers to the binary collection of compiled Java classes within the APK package file.[22]

Step two includes the actual decompiling phase of the Java bytecode back into .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 of mHealth apps, since *dex2jar* was not functioning at the time they conducted their study.[23] Within their toolchain, they 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. [24] We will use *FernFlower* as the decompiling tool for our study, since it evidentially outperformed

---

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

[21]   Pan (2010)

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

[23]   Enck et al. (2011), p. 16

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

Soot in a further evaluation by Enck et al. (2011).[25] The result of this second step will a directory full of .java files that represents the source code of the APK apps.

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*[26] will be used. *astyle* sets appropriate levels on indenting to the source code, to improve the reading experience.

In the last step of our decompilation process, a tool called *apktool*[27] 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.[28] The XML files are of interest for the automated privacy risk analysis, because Android text input controls can be defined in an external XML file, rather than in the source code itself.

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 processable for further analysis.

Brüggemann, Hansen (2016) identified six potential privacy risk factors that a mHealth app can pose and combined them into a privacy risk index equation. The six factors contain three binary factors. The first binary factor is the question, whether an app requires a login via username/email or a social media login service such as Facebook, Twitter or Google+. The second binary factor asks, if the app uses secured HTTP connections to the servers it is communicating too. While the first two binary factors can be assessed at a reasonable level of difficulty via static code analysis, it is challenging to do so for third binary factor, the 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 for a static code analysis. Our

---

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

[26]   Davidson, Pattee (2006)

[27]   Tumbleson, Wiśniewski (2010)

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

research will find out to what degree it will be possible to evaluate how reasonable the data collection is.

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 the medication intake, symptoms or vital values. The last two privacy risk factors express the target to which the personal data is likely to be sent. Brüggemann, Hansen (2016) refer to target as the host or destination data is sent to. The targets are split into two factors, one for unspecific data targets, which refers to the usage of advertisement (ad) banner services or analytics tools within the app. To what extend the inclusion of ad services or analytics tracks the user data is unclear during the assessment and therefore, the data targets are classified as unspecific. The second data target factor tries to observe immediate data connections after personal data input. 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.[29]

Our study aims at identifying the privacy risk factors identified by Brüggemann, Hansen (2016) by automatically scanning the source code of each app in our working dataset.

In order to identify whether an app requires 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 can be identified by the button classname *com.facebook.login.widget.LoginButton* for Facebook login, *com.twitter.sdk.android.core.identity.TwitterLoginButton* for Twitter login functionality and *SignInButton* for Google login. 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 target web address contains a secure HTTPS prefix. This indicates an encrypted connection. The targets web addresses, that requests are made to, also indicate the personal data target factor. The web address domain name usually points out if the web request is made to the app providers server (usually the company name, or alike) or e.g. to a click analytics service. Especially if variables that were assigned through user

---

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

input fields can be traced until they are sent out in web request. In order to identify the individual text fields that call for user input, two source code scanning strategies 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 get 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 segment the individual data input text field labels to the personal data categories. To identify if analytics or advertisement services are used within the app, we can scan the source code for the inclusion of corresponding libraries. For example, the Google Analytics[30] library is used within Android source code by including the library package *com.google.android.gms.analytics.GoogleAnalytics*.

After the implementation of is completed, 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.[31] This enables the mHealth apps to be compared, regarding their privacy risk.

4.2 Evaluation

First, we want to evaluate the degree of privacy risk assessment automation that has been accomplished. The evaluation will be based on the feasibility and the obstacles detected in implementing the automated privacy risk assessment. We will outline which privacy risk factors can be detected automatically and which factors can not and why.

Using the Brüggemann, Hansen (2016) user interface allows 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 and its impact on users, we will expose 15 potential users to the user interface and let them discover our detected privacy risks. We will then give an overview of the users options towards the

---

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

[31]   SeeBrüggemann, Hansen (2016), p. 99

presence of enhanced privacy risk information, that might change the users decision making process regarding the selection of mHealth apps.

5. Structure

In this section we would like to introduce the structure of the final thesis chapters.

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.
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 will be used in 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 automated privacy risk assessment tool.
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 itself.

2.2.1 Download Phase	Description of the APK file discovery and download phase of the tool. Free Android apps are available for everybody with an Android smartphone. We want to feed mHealth apps into our automated assessment system and therefore need to download the APK files automatically.
2.2.2 Decompilation Phase	This section provides information on the decompilation 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 resource to perform the static code analysis on, as outlined 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 the 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 results 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 the 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 the tool and share their option on the implications this tool has on mHealth app decision making.
4 Conclusion	A critical reflection on the thesis. Were 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 expected to gain at least a code recovery rate of above 90%.[32] Therefore, we expect the majority of APK files to be assessable. 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 anticipated result is an overview of the users' options on the usage of the automated privacy risk assessment tool. Depending on the implementation success it will reveal to be either useful or rather not useful to the users decision making process

---

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

regarding mHealth apps.

7. Problems

So far, only one problem arose from the fact that the core of this master thesis relies on an undocumented Google API for downloading the APK files of the apps. After testing the APK download API, it became clear that Google is limiting the request rate to the API. The extend to which requests are limited and blocked is unknown. After exceeding the limit on one day, it was possible to resume downloading on another day. 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 could be a fallback solution.

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

Davidson, Pattee (2006)

Tal Davidson, Jim Pattee: Artistic Style. http://astyle.sourceforge.net/, Abruf am 02/10/2016

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 Pol-

icy 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

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

Khalid et al. (2015)

Hammad Khalid, Emad Shihab, Meiyappan Nagappan, Ahmed E. Hassan: What Do Mobile App Users Complain About? In: IEEE Software. Nr. 3, Jg. 32, 2015, pp. 70–77

Kim et al. (2012)

Jinyung Kim, Yongho Yoon, Kwangkeun Yi, Junbum Shin: Scandal: Static Analyzer for Detecting Privacy Leaks in Android Applications. In: IEEE Workshop on Mobile Security Technologies (MoST). 2012,

Kweller (2010)

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

Mcclurg (2012)

Jedidiah Mcclurg: Android Privacy Leak Detection via Dynamic Taint Analysis. In: Unpublished. 2012,

Mitchell et al. (2013)

Stacy Mitchell, Scott Ridley, Christy Tharenos, Upkar Varshney, Ron Vetter, Ulku Yaylacicegi: Investigating privacy and security challenges of mhealth applications. In:

19th Americas Conference on Information Systems, AMCIS 2013 - Hyperconnected World: Anything, Anywhere, Anytime. Jg. 3, 2013, pp. 2166–2174

Nolan (2012)

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

Pan (2010)

Bob Pan: dex2jar. https://github.com/pxb1988/dex2jar, Abruf am 10/02/2016

Tumbleson, Wiśniewski (2010)

Connor Tumbleson, Ryszard Wiśniewski: Apktool. https://github.com/iBotPeaches/ Apktool, Abruf am 11/02/2016

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