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, März 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 phone and tablet applications (apps) grows extensively since 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 affected 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 remedy health-related information deficiencies. Receiving health-related information does not come at no costs for the users. Users are asked to expose their personal health-related information in order to receive a tailored app that fits their preferences.[2] Users reveal vulnerable information about their health status while it remains mostly unclear how and where the data is sent, processed and stored.[3]

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.[4] Processing these privacy policies requires a higher level of education and time to read through large bodies of text, in order to find the relevant information. Additionally, the important information is hidden in legal language or is insufficiently addressed, if at all. Aside from data usage beyond the control of the users, 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 comparability between apps. When users are looking for specific functionality in an 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 of similar functionality, the high volume of apps available in the app stores[5] makes it laborious to review all of them by hand. One way to assess privacy risks of the large amount of mHealth apps is to automate the review process of each individual app. The

---

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

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

3 See He et al. (2014), p. 652.

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

5 See Enck et al. (2011), p. 1.

assessment automation can be done by downloading and analyzing the source code of each app and by tracing data leaks. Static code analysis is used in the field of informatics to analyse application source code and detect faults or vulnerabilities.[6] It is yet 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. A static code analysis could, in theory, be used to automatically assess some of the privacy risks that mHealth apps pose. Previous research has not shown if the combination of static code analysis is feasible in the field of mHealth app privacy risk assessment and therefore the aim of this study is to explore the possibilities of static code analysis for privacy risk 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 furthermore helps to drastically reduce the costs of reviewing each individual app and enhances the information experience users receive while looking for 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.

## 2. Related Work

mHealth apps have been examined in various research studies that aim at providing insights for developers as well as users into how private information is processed. Privacy issues are the most impactful user complaint while using mobile apps.[7] This encourages research to address privacy risks and data leaks.

The technical side of data leaks has been extensively studied. An analysis of mHealth data storage in internal Android log files or the memory card used within the phone or tablet has been conducted.[8] Technical evaluation of mobile apps even goes further into the topics of decompilation to analyze device identification or geolocation data leaks.[9] Decompilation reveals to be a feasible assessment technique for privacy risks and data

---

6 See Baca, Carlsson, Lundberg (2008), p. 79.

7 See Khalid et al. (2015), p. 5.

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

9 See Mcclurg (2012), p. 1, 5., Enck et al. (2011), p. 1. and Mitchell et al. (2013), p.6-7.

leaks.

In informatics and software development contexts, static code analysis has been used to analyze source code and provide feedback on coding styles to the users while programming or ”to find defects in programs”[10]. Static code analysis provides a fast way to analyze source code[11], which makes it suitable to automate the analysis on large datasets. Static code analysis has been used in the context of mobile apps to trace the leakage of device features, such as the location data and phone identifier.[12]

Our study will 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 which Brüggemann, Hansen (2016) identified. Brüggemann, Hansen (2016) identified six factors of privacy risk within mHealth apps and proposed a formula to combine those factors into an equation. These privacy risk factors are less of a technical risk but rather focus on the user data input and where this data is sent. It is unclear if static code analysis is a viable tool to analyze the user input data privacy risk.

## 3. Objectives

The main objective of this study is to ascertain how and to what degree the assessment of privacy risk factors for mHealth apps can be automated. In order to reach this objective, the following sub-objectives have to be met.

The first sub-objective is to download and decompile as many Android apps as possible. This step is necessary to gain access to the app source code for further evaluation. After decompilation, the second sub-objective is to apply static code analysis to identify privacy risk factors within the apps' source code. As a third sub-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.

---

10 Bardas et al. (2010), p. 1.

11 See Bardas et al. (2010), p. 5.

12 See Kim et al. (2012), p. 1-2.

## 4. Definitions

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

### 4.1 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 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. [13]

### 4.2 Mobile Health Apps

Mobile health (mHealth) apps are smartphone or tablet applications that support users by enabling them to gather health related information and support the consumer in medical or health related issues.[14]

### 4.3 Privacy Risk Factor

A privacy risk factor of an app is a factor or a circumstance that increases the risk of private user data being leaked. An example for a privacy risk factor is the usage of an unencrypted HTTP data connection.

### 4.4 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. [15]

---

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

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

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

# 5. Methods

## 5.1 Automating the Privacy Risk Assessment

### 5.1.1 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 to decompiling the Apple app binary back into its source code. These approaches involve unlocking and jailbreaking[16] an Apple iPhone or iPad, which is a violation of the Apple terms of service and therefore forbidden.[17]

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, 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.

To get an overview of the APK files available on the Google PlayStore, we use the repository of app listings by W. Xu, Liu (2015) that was extracted from the Google PlayStore from 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 our study. 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 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

---

[16] Jailbreaking refers to the action of unlocking the iOS device firmware to gain unrestricted access to the bootloader. See Kweller (2010), p. 1.

[17] See Kweller (2010), p. 1.

dataset. We will download as many of these 3,180 apps as possible. Downloading 3,180 apps in a short period of time triggers Googles security mechanisms, since a normal user could never download that many apps in the same timeframe from the PlayStore. We will apply a careful downloading technique and allow for pauses in between app downloads to not over-use the Google API.

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

Step two includes the actual decompiling phase of the Java bytecode back into .java files. A .java file contains exactly one Java *class* in human-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.[21] Within their toolchain, they used a tool called *Soot*. *Soot* optimizes the decompiled code to improve the readability by humans. 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).[22] The result of this second step will be 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*[23] will be used. *astyle* sets appropriate levels on indenting to the

---

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

19 Pan (2010).

20 See L. Xu (2013), p. 6.

21 See Enck et al. (2011), p. 16.

22 See Enck et al. (2011), p. 6.

23 Davidson, Pattee (2006).

source code, to improve the reading experience.

In the last step of our decompilation process, a tool called *apktool*[24] 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.[25] 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.

### 5.1.2 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.

The static code analysis tool will not be able to identify new privacy risk factors by itself, but rather has to be individually programmed to scan the source code for privacy risk factors. Brüggemann, Hansen (2016) identified six potential privacy risk factors that a mHealth app can pose and combined them into a privacy risk index equation. We will focus on identifying these privacy risk factors with our static code analysis tool. 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. Since this is not feasible to be detected by a static code analysis, we will leave the reasonableness assessment to the users of our tool.

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

---

24 Tumbleson, Wiśniewski (2010).

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

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 extent 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.[26]

Our study aims at exemplary 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 sent to the app providers server (usually the company name, or alike) or e.g. to a click analytics service. These web requests are of special interest if user input data is transmitted. 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

---

[26] This section follows Brüggemann, Hansen (2016), p. 7-9.

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[27] library is used within Android source code by including the library package *com.google.android.gms.analytics.GoogleAnalytics*.

After the implementation 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 a user interface.[28] This enables the mHealth apps to be compared, regarding their privacy risk.

### 5.2 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 obstacles detected in implementing the automated privacy risk assessment. We will outline which privacy risk factors can and can not be detected automatically, as well as the reasons for this. Furthermore, we will evaluate for which privacy risk factors the automated static code analysis is superior to using manual assessment techniques.

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. The 15 users will be interviewed in terms of their experience using the privacy risk assessment tool. The interview questions will ask the users about their perception on the impact such a tool, and the information it provides, has on the app store usage. We will then 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.

---

[27] https://web.archive.org/web/20160205060303/https://www.google.com/analytics/, visited cached version on 02/05/2016.

[28] See Brüggemann, Hansen (2016), p. 10, 15-16.

## 6. 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 Objectives** | This section describes the primary objective of the thesis and the subsequent secondary objectives, that we hope to reach. |
| **1.3 Structure** | The formal structure of the thesis, used to provide the reader with an overview of the thesis. |
| **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 |
| **3 Method** | This chapter describes the main implementation phase of the automated privacy risk assessment tool. |
| **3.1 Privacy Risk Factors** | Introduction of the privacy risk factors identified by Brüggemann, Hansen (2016) and a further explanation of their relevance to mHealth app assessments. |
| **3.2 Implementation** | Details on the implementation of the automated mHealth privacy risk assessment tool itself. |
| **3.2.1 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. |

| | |
|---|---|
| **3.2.2 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. |
| **3.2.3 Static Code Analysis Phase** | Information on the different approaches to gather information on 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. |
| **3.3 User Interviews** | Describing how we interview the 15 users regarding their opinions on the automated privacy risk assessment. |
| **4 Results** | 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? |
| **4.1 Privacy Risk Factors** | Results on what privacy risk factors are identifiable via static code analysis. |
| **4.2 Implementation** | Explaining the outcome of the implementation step. |
| **4.2.1 Download Phase** | Results on how many mHealth apps could be downloaded and how the download phase went. |
| **4.2.2 Decompilation Phase** | Results on how many mHealth could be decompiled and at what degree of quality. |
| **4.2.3 Static Code Analysis Phase** | Results on the outcome of the static code analysis |

| | |
|---|---|
| **4.3 User Interviews** | 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. |
| **5 Discussion** | |
| **5.1 Evaluation of Implementation** | Information on the feasibility of implementing an automated mHealth privacy risk assessment tool, the challenges and obstacles. |
| **5.2 Limitations** | Explanations in what way the research approach and execution was limited or constrained during this thesis. |
| **5.3 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. |
| **5.4 Conclusion** | A critical reflection on the thesis. Were all the objectives reached and are the methods used coherent? |

## 7. Expected Results

The results of this master thesis will be the implementation of the automated privacy risk assessment tool and an evaluation of the users' options on the usage of the automated privacy risk assessment tool. Part of the expected results will be the answer to the question, if static code analysis is a viable tool to assess privacy risk factors of mHealth apps.

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 extent, and will be provided. The decompilation process is expected to gain at least a code recovery rate of above 90%.[29] Therefore, we expect the majority of APK files to be assessable. We expect the

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

static code analysis for privacy risk factors to be successful for the majority of factors and will provide insights into the possible degree of detail.

## 8. Problems

So far, there are no problems.

**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. No. 2, Vol. 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. No. 4, Vol. 14, 2012, e112

Davidson, Pattee (2006)

Tal Davidson, Jim Pattee: Artistic Style. https://web.archive.org/web/20160228181142/http://astyle.sourceforge.net/, visited cached site on 02/28/2016

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. No. 1, Vol. 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: Proceedings of the 20th USENIX Conference on Security. No. August, Vol. SEC'11, 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. Vol. 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. No. 3, Vol. 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. No. 8, Vol. 27, 2010, pp. 480–523

Mcclurg (2012)

Jedidiah Mcclurg: Android Privacy Leak Detection via Dynamic Taint Analysis. In: . 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. Vol. 3, 2013, pp. 2166–2174

Nolan (2012)

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

Pan (2010)

Bob Pan: dex2jar. https://web.archive.org/web/20160222112240/https://github.com/pxb1988/dex2jar, visited cached site on 02/22/2016

Tumbleson, Wiśniewski (2010)

Connor Tumbleson, Ryszard Wiśniewski: Apktool. https://web.archive.org/web/20160222172049/https://github.com/iBotPeaches/Apktool, visited cached site on 02/22/2016

L. Xu (2013)

Liang Xu: "Techniques and Tools for Analyzing and Understanding Android Applications". 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. No. 1, Vol. 3, 2015, e28