Security evaluation of glucose monitoring applications for Android smartphones

Research Project 1
Master Security and Network Engineering
University of Amsterdam
Project report
Version: 1.0

Roy Vermeulen `roy.vermeulen @os3.nl`
Edgar Bohte `edgar.bohte@os3.nl`

Supervisors:
Alex Stavroulakis (KPMG)
Vincent de Jager (KPMG)

February 10, 2019

Abstract

Smartphone applications support diabetic patients with glucose monitoring, which improves quality of life. These applications process sensitive medical data, while smartphone applications have previously been proven to likely be insecure.

We selected three popular smartphone applications for glucose monitoring. We tested these applications with well known tools to discover security flaws. The security flaws that were found were categorised by the Open Web Application Security Project mobile top 10 framework.

Data storage was found to be the largest security risk in most applications, as sensitive data was found to be stored in Android system logs and local data was unencrypted. Insecure authentication was another large security risk, as password policies did not enforce the use of strong passwords and authentication tokens were not generated randomly and not invalidated regularly.

There was at least one method found to obtain sensitive data in each of the analysed applications. A way to alter medical data was found in two out of three analysed applications. While the reviewed applications were found to be secure on many aspects, all of them had some security flaws.

Keywords: Diabetes, Security, Android, Glucose monitoring, Application

Contents

1 Introduction

According to the latest data, which is from 2014, 422 million adults worldwide have diabetes and this number is likely to grow [1]. Diabetes can have negative consequences for a patient's health, but with proper disease management, nearly all of these consequences can be avoided. Therefore, it is important for patients to keep their glucose levels under control, a practice called diabetes management. To keep glucose levels under control a patient should log their measured glucose levels. This is most commonly done by catching a drop of blood with a strip and putting it in a device, which measures the glucose level.

In the past, keeping track of these measured values was done with pen and paper. Nowadays, there are smartphone applications (apps) where the patient can fill in the measured glucose levels or even connect the measurement device to the smartphone. In a report from 2016, it was reported that 4.1 million people were using diabetes monitoring apps at the time [2].

Inherent to their function, diabetes monitoring apps process medical data. Medical data is considered sensitive data under the European General Data Protection Regulation (GDPR) and is subject to particularly strict rules [3]. These rules appear to be justified, as medical data can be used for high financial gain on black markets and retains its value over time [4, 5]. The value of this medical data lies in the multitude of ways in which it can be exploited, such as medication fraud or insurance fraud [6]. Some of these ways can be very harmful to the patient, such as identity theft or medical blackmail [5, 7].

Even though diabetes monitoring apps do not contain full medical records, confidentiality of the data they contain and processes is still a concern. Knorr et al. [8] outlined a few theoretical threat actors and their motives to exploit this data to their benefit, which can be detrimental to the users of the apps. An example of this is a health insurance company, who might want diabetes monitoring data. Such a company could use this data to increase the health insurance premium of patients, who have difficulty with diabetes management. These concerns are not only theoretical since 168 out of 1419 major healthcare data breaches reported to the United States Department of Health and Human Services involved mobile devices [9].

The integrity of the data processed by diabetes monitoring apps is also a concern, in this case with regards to the safety of the user. For example, most apps can be used to calculate the amount of insulin that needs to be injected. If a malicious third party can compromise the integrity of the measurement data, a patient could inject the wrong amount of insulin. This could have dangerous consequences for the patient's health [10].

1.1 Research questions

Considering the security issues stated above, we pose the following research question:

What is the current state of security in popular diabetes blood glucose monitoring apps?

To answer this question, we investigated the following sub-questions:

1. Which data can be derived from these apps by an unauthorized third party?
2. How can an unauthorized third party derive data from the glucose monitoring apps?
3. How can an unauthorized third party alter the data in these apps?

Section 2 will describe the academic work we found related to this topic. Section 3 will cover the methods that we used to execute this research. In Section 4, we will present the results of our research and in Section 5, we will discuss the impacts of these results. In Section 6, we will answer our research questions and draw a conclusion.

2 Related Work

In Section 1 we explained the context for our research. In this section, we will discuss some of the work done on this subject by other researchers. We will discuss papers on the topic of general Android security, on the topic of mobile health application security and lastly, we will discuss a research paper on the topic of threat mitigation in mobile health application security.

Khan et al. [11] used a theoretical approach to demonstrate that the possibilities for Android applications to have security flaws. Khan outlined a few theoretical attacks. One of these attacks, for example, made it possible for apps to transmit sensitive data without the user's consent. Khan also argued that that application developers do not follow best security practice, which can lead to misuse of the apps. Examples of this misuse are battery draining and assisting in a Distributed Denial Of Service attack. Finally, Khan argued that the security controls of Android are limited, which for example do not give users enough control over the behavior of the apps. The limited security controls cannot prevent malicious apps to pose as legitimate apps in the official Google Play store.

Furthermore, Felt et al. [12] created a tool to check whether Android applications had more permissions than they needed. This tool generated the maximum set of permissions needed for the application and compared

them to the set of permissions requested. Felt et al. found that of 940 analyzed Android apps, about one-third of these apps requested more permissions than they needed. Applications were generally overprivileged by a few permissions. This indicates that developers often misuse this security control.

Knorr et al. [8, 13] looked at the privacy, security, and safety of 154 mobile health (mHealth) Android apps, which are used for managing blood pressure or diabetes. To perform this analysis they made a framework [8]. In this framework, there are four parts, namely, static analysis, dynamic analysis, web server security, and privacy policy inspection. From these analyses, they determined that most apps lack encryption of data transmission and storage [13]. Furthermore, in-app adds are also a problem, because they send the package name in clear text in the HTTP header, which can be used by thieves and eavesdroppers for easy access. Also, many apps lacked or have an incomplete privacy policy.

Papageorgiou et al. [14] also did an analysis of security in Android mHealth applications. They did static analyses, dynamic analyses, web server security analyses, and privacy policy inspections. Furthermore, they analyzed unnecessary permissions requested by the apps, assessed the compliance with the GDPR and tracked the developers' response to their findings. Papageorgiou et al. found multiple major security and privacy issues in most of the 20 apps that they analyzed, for example transmitting health data insecurely to the vendor. Minor issues were also found in many apps, for example the transmission of anonymous behavioural data to third parties.

In an attempt to provide a solution to these security risks, Hussain et al. [15] created a framework which could be implemented on Android smartphones to prevent apps on the smartphone to access data when they should not. However, this framework is difficult to implement on a large scale and for now, does not provide a widely implementable solution.

3 Methods

Now that we have explained the context and the state of research on this topic, we will use this section to explain the way we executed our research. We will start by discussing the selection of the Android apps we analyzed. We will then explain our test setup, continued by the categorization of our results. Furthermore, we will discuss the restrictions of this research and we will close off by discussing the tools we used for this research.

3.1 App selection

In this research, we looked at three of the most popular apps measured by rating and number of reviews on the Google Play store. We searched

the Google Play store manually. Due to time constraints, we limited our scope to Android apps only, as Android is the most widely used operating system [16]. We limited the scope further by excluding paid-for apps and paid-for features in free apps. We only selected glucose monitoring apps that allow a user to manually enter blood glucose data. Therefore, apps that obtain their measurements solely by connecting to a device are outside the scope of this research.

3.2 Test setup

For the analysis of these apps, we used an emulated device which ran Android 8.0 Oreo. We used the Genymotion emulator [17] for this emulation. We obtained the Android application package (APK) files from the Google Play store and loaded them onto the emulated device with Android Debug Bridge (adb) [18]. For each app, we manually generated simulated patient data.

3.3 Results categorization

We tested the apps for vulnerabilities described in the ten categories of the Open Web Application Security Project (OWASP) mobile top 10. OWASP is a not-for-profit charitable organization that provides resources for building and maintaining apps [19]. One of these resources is a top 10 mobile risks list. These are updated periodically and the latest version, as of writing this document, is from 2016. This list is updated by the use of a survey and a call for data submission globally. The list was then finalized by giving their community a 90 day feedback period. The categories in the OWASP mobile top 10 are as follows:

1. Improper Platform Usage
2. Insecure Data Store
3. Insecure Communication
4. Insecure Authentication
5. Insufficient Cryptography
6. Insecure Authorisation
7. Client Code Quality
8. Code Tampering
9. Reverse Engineering
10. Extraneous Functionality

3.4 Restrictions

We did not perform tests that would disrupt or slow down the service that these apps provide in any way. Furthermore, we limited our scope to users, who downloaded the apps from the Google Play store. We analyzed the apps as they are found in the Google Play store and we did not explore the option that users can download modified versions of the apps through third parties. Therefore, OWASP category 8: "code tampering" was omitted because it is outside of our scope. Due to time constraints, we cannot analyze code manually and therefore OWASP categories 7: "client code quality" and category 10: "extraneous functionality" were also omitted.

3.5 Tools

We used a number of tools to test for vulnerabilities. Most results were obtained through the use of these tools. Nevertheless, the password policy analysis was done manually. Different passwords were used to evaluate the password requirements, such as the minimum password length, maximum password length, use of password trimming, and the use of numbers and special characters. The tools we used are listed below.

- The Mobile Security Framework (MobSF) [20] is an automated mobile penetration testing framework. We used this framework to perform static analyses to check which permissions the apps use and to obtain the java code.

- adb [18] is a command line tool used to communicate with an Android device. We used it to inspect the logs and the local storage.

- SSLlabs [21] is an online toolkit for testing Secure Sockets Layer, Transport Layer Security, and Public Key Infrastructure. We used SSLlabs to examine the security of the connections to the servers that the apps use.

- BurpSuite [22] is a toolkit for testing web application security. We used BurpSuite to monitor and manipulate the traffic the apps send and receive.

- Drozer [23] is a security audit and attack framework. It was used by us to see which activities are exported. If the activity is exported and the permissions are not set, every app can invoke them.

- AddonsDetector [24] is a smartphone application that is used to detect what add-ons apps use and which apps send push notification advertisements. We used AddonsDetector to check if there were any advertisement add-ons that might collect sensitive data.

4 Results

In Section 3 we discussed the approach to our research. In this section, we will present our findings from following that approach. We start by discussing the apps that were selected for analysis. Next, we list our results, categorized by the OWASP top 10 category they belong to. We close off by rating our results found in each category, which gives us a broader overview of the results.

The apps selected for analysis are shown in Table 1. As we searched the Google Play store manually, it is possible that we overlooked some glucose monitoring apps. Nonetheless, the selected apps appear to be some of the most widely used ones. The results of our analysis are listed below, categorized by the OWASP top 10 vulnerabilities.

4.1 Improper platform usage

This category covers the misuse of platform features and failure to use platform security controls. In the context of our research, the analysis of requested Android permissions falls in this category. Here we mention all permissions that the apps have, but does not need for its functionality. We also analyze all exported activities the applications have that could reveal sensitive data and the permissions needed to invoke them.

4.1.1 Permissions

Applications running on Android do not have access to all data and hardware on the smartphone by default. To access certain sensitive data or system features, applications need to ask the user for permission. Sometimes these permissions are asked on the installation of the app, sometimes they are asked when activating a certain feature of the app which requires a specific permission. The nature of these permissions can range from accessing the users' contacts to using the phones' internet connection.

Most of the permissions that Glucose Buddy requires are needed for the functionalities that it provides. However, it does have permission to turn on the camera flash for an extended period of time, but we could not find a reason why the app required this permission. However, having this

App name	Version number	Average rating	Number of reviews
Glucose Buddy	5.36.3775	4.4	14,386
Diabetes:M	7.0.8	4.6	17,110
Diabetes Connect	2.4.1	4.5	4,186

Table 1: Diabetes monitoring apps selected for analysis with their corresponding version number, average rating, and number of reviews.

permission does not create a security risk.

Most of the permissions that Diabetes:M requires are also necessary for the functionalities that it provides. It does have permission to access the location of the phone, in both coarse granularity and fine granularity, while we could not find a reason that it required this permission. All of the permissions Diabetes Connect required are used for its functionalities.

4.1.2 Exported activities

An exported activity is an activity that other apps can call. Each exported activity has a permission. However, this permission can also be null. This means that every app can call this activity. Glucose Buddy has five exported activities, all of which have a permission of null, but none of them could be abused. Diabetes:M has nine exported activities, all of which also have a permission of null. One of them could potentially be used for the extraction of personal data including medical data. When this certain exported activity is called the user gets asked if another app could see their configuration settings, last logged glucose value, and push logged glucose values to their log. Diabetes Connect has two exported activities and both of them have a permission of null. One of them could potentially be used to extract medical data from the app. We could not prove that this is possible in practice, as the app crashed whenever this exported activity was called.

4.2 Insecure Data Storage

The insecure data storage covers all data that is stored locally on the device in an insecure manner and all data that is leaked. In the context of our research, this category is for the analysis of Android system logs and the security of local files and databases. Furthermore, the leakage of data to third parties through advertisement and analytics service also belongs in this category.

4.2.1 Android system logs

Glucose buddy stores the following items in the Android system logs: settings, queries for food items, profile pictures, and the authentication token. The settings contain among other things the e-mail address, name and time zone, so they are considered personal data. Diabetes:M does not log any personal information in the public accessible logs. Diabetes Connect logs the keystrokes of its own standard keyboard in the public accessible logs when the user enters a measurement. In nearly all cases, this results in all measurements being logged in the Android System logs. These measurements include blood sugar level, carbohydrate intake, heart rate, weight and medication intake.

4.2.2 Local storage

In the local storage of Glucose Buddy, we found a folder named *shared_prefs* in this folder there are multiple Extensible Markup Language (XML) files. In one of these files the id, email, name, birthday, height, weight, profile picture, gender, and timezone can be found. Another file in this folder contains the user id and the session token. In the local storage, there also is a folder named databases, which contains multiple Structured Query Language Lite 3 (SQLite3) databases. One of these databases contains all data that can be logged in the app.

In the local storage of Diabetes:M, we also find a folder named *shared_prefs*, which contains multiple XML files. One of these files contains the access token, user id, security cookie, username, and what the targeted glucose level should be. In the local storage, there also is a folder named databases, which contains multiple SQLite3 databases. One of these databases contains all data that can be logged in the app. Another database contains also the medication the user has added to the app. Lastly, there is a database file which contains all personal data such as username, name, email, birthday, gender, etc.

In the local storage of Diabetes Connect we also find a folder named *shared_prefs*, which contains multiple XML files. One of these XML files contains the gender, device id, password in plaintext, country, name, username, and what the targeted glucose level should be. There also is a folder with multiple SQLite3 databases. One of them contains all data that can be logged in the app.

4.2.3 Advertisement and analytics add-ons

Glucose Buddy has one advertisement add-on and three analytics add-ons. These are the Google Mobile Ads add-on and the Answers add-on, Branch add-on and the Flurry add-on respectively. Diabetes:M has one advertisement add-on and three analytics add-ons. These are Google Mobile Ads add-on and the Answers add-on, Firebase Analytics add-on and the Google Analytics add-on respectively. Diabetes Connect has no advertisement add-ons and one analytics add-on. This analytics add-on is the Firebase Analytics add-on.

4.3 Insecure Communication

The insecure communication category covers all vulnerabilities regarding poor handshaking, incorrect Secure Sockets Layer (SSL) versions and plaintext communication of sensitive data. Regarding our analysis, this category contains the usage and quality of the HyperText Transfer Protocol Secure (HTTPS) and Transport Layer Security (TLS) protocols.

4.3.1 Usage of HTTP and HTTPS

Glucose buddy uses HTTP for the export of measurement data and for changing a password. This data can be intercepted and viewed in plaintext. Diabetes:M and Diabetes Connect use HTTPS for all their communication.

4.3.2 SSL analysis

Analyzing the domains of Glucose buddy, Diabetes:M and Diabetes connect with the SSLlabs tool resulted in many similar results. All three domains did not support TLS 1.3, which is the latest version of the TLS protocol. All domains did support TLS 1.0 through 1.2. Certificate Transparency (CT) is a technique to detect wrongly issued TLS certificates. The domain certificates of Glucose buddy and Diabetes Connect are registered in certificate transparency logs, while the TLS certificate of Diabetes:M is not registered in the certificate transparency logs. Another way to mitigate the use of wrongly issued certificates is through Domain Name Server Certificate Authority Authorisation (DNS CAA). None of the domains of the apps we evaluated supported DNS CAA. Furthermore, none of the apps we evaluated used SSL pinning.

4.4 Insufficient cryptography

The insufficient cryptography category contains all risks that involve cases where cryptography was applied but not implemented correctly. In this research, we found no cases in which cryptography was applied. When there is a lack of encryption, the OWASP guidelines state that it falls under the category of insecure storage, which is treated in subsection 4.2. Therefore there are no results to report in this section.

4.5 Insecure Authentication

The insecure authentication category encompasses failure to identify the user or maintain the users' identity when required. It also involves weaknesses in session management. For our analysis, this category is for the weaknesses found in the use of authentication tokens and the password policy.

4.5.1 Authentication tokens

Glucose buddy generates authentication tokens which stay valid for an extended period of time. Due to time constraints, we could not test for exactly how long the token remains valid, but we observed that a token remained valid for four weeks. The token also did not become invalidated whenever the user logged out. We did not find improper usage of authentication tokens in Diabetes:M

Diabetes Connect generated the authentication code by combining the e-mail address and the password, separated by a colon, and encoding them with base64. Consequently, if this token is obtained, the e-mail address and the password can be found and vice versa. This also means that the token will remain valid for as long as the password and the e-mail address remain unchanged.

4.5.2 Password policy

The results of our password policy analysis are shown in Table 2. Glucose buddy and Diabetes:M employ the same password policy. There is only a requirement on minimum length, which is 8 characters. There are no further rules in the password policy. The password policy of Diabetes Connect requires the password to be at least 6 characters and to contain at least 1 letter and 1 number. There are no further rules in the password policy. Passwords are not converted into lowercase in any of the analyzed apps. In one app the password was trimmed if it was too long. Diabetes:M trimmed the password to a length of 72 characters whenever the password exceeded 72 characters.

4.6 Reverse Engineering

This category is for all vulnerabilities that can be found through decompiling the apps and analyzing them. For our research, we make a quick assessment of how much code obfuscation is used. We found that all apps used some code obfuscation. Often classes or methods that should have had meaningful names were given random or arbitrary names.

4.7 Overall rating of results

We rated the apps on each OWASP category a rating from one to three. One means that we found no security issues, or only minor security issues. Two means that we found security issues that were hard to exploit or had low impact. Three means that we found vulnerabilities that had a large impact. These results are displayed in Table 3.

	Glucose Buddy	Diabetes:M	Diabetes Connect
Length	minimum 8	minimum 8	minimum 6
Letters	no rules	no rules	at least one letter
Numbers	no rules	no rules	at least one number
Special characters	no rules	no rules	no rules

Table 2: Password policies of the analysed apps

OWASP category	Glucose Buddy	Diabetes:M	Diabetes Connect
1: Improper platform usage	1	2	2
2: Insecure data storage	3	2	3
3: Insecure communication	3	1	1
4: Insecure Authentication	2	2	3
6: Insecure authorisation	2	1	3
9: Reverse engineering	1	1	1

Table 3: Results rated from 1 to 3, 1 being no vulnerabilities found, 2 being minor vulnerabilities found or vulnerabilities found that are hard to exploit and 3 being major vulnerabilities found

5 Discussion

In Section 4 we presented the vulnerabilities that we found in our research. In this section, we discuss how these vulnerabilities could impact the users of the diabetes apps. We also discuss the options for developers to solve these vulnerabilities. Moreover, we suggest solutions for the users to mitigate these risks for when the developers have not (yet) solved these vulnerabilities.

5.1 Discussion of results per app

5.1.1 Glucose Buddy

In Glucose Buddy, the app logged the authentication token in the Android system logs. These logs are accessible for all apps. A malicious app or a malicious actor with physical access to the device could, therefore, extract the logs and obtain the authentication token. This authentication token could be used to read and modify the data stored serverside through Application Programmer Interface (API) calls.
This security risk is increased by the fact that authentication tokens stay valid for at least four weeks and are not invalidated by a logout. It is not clear how long these tokens stay valid exactly, but it could be possible that a malicious actor can gain access by obtaining a relatively old token. The likelihood of installing a malicious app might be small, as many users might not install apps from third parties. Yet, a malicious app could appear in the play store for a period of time or a user could be tricked into installing a malicous app through a phishing campaign.

Developers can mitigate this security risk by not storing sensitive data in the Android system logs and invalidating old authentication tokens after a reasonable amount of time and after every logout. Users can mitigate this threat by preventing the installation of malicious apps and implementing strong physical security. Physical security in this context means securing the

device with a strong authentication method and preventing loss or theft as much as possible. It is also helpful to implement a remote locking mechanism that can be activated in case of loss or theft. Preventing installation of malicious apps can be done by installing only apps from a trusted source through the Google Play store and by being aware of phishing dangers.

Another security risk is that some functionality of the app is handled through a web viewer and this data is sent over HTTP. This functionality includes changing passwords, logging in and out, and exporting measurement data. While this functionality is most likely used infrequently, it is possible for a malicious actor who is intercepting the traffic to obtain the e-mail address and password belonging to an account whenever the feature is used.

Developers can mitigate these risks by implementing secure communication channels. Users can mitigate these risks by connecting their phone to the internet through a Virtual Private Network (VPN) and only connected to properly secured WiFi networks.

In Glucose Buddy it is possible to share measurement data through links. These links are not generated randomly but incremented by 1 every time a link is generated. This way, anyone can know which links have been generated in the past. When following such a link, anyone with an account can log in and view this data. This combined with the possibility to guess share links makes it possible for unauthorized third parties to view measurement data. While this can hardly be used for a targeted attack, it is a case of data leakage nonetheless.

Developers can mitigate this risk by creating random share links with a large character space. This way it becomes harder to guess links containing shared data. A better mitigation would be that the share links require a stronger authentication method. Users can only mitigate this risk by not sharing sensitive data through share links.

5.1.2 Diabetes:M

In Diabetes:M exported activities can be invoked to see the last logged measurement and configuration settings. This risk can be mitigated by developers by requiring permissions for the exported activities.

5.1.3 Diabetes Connect

Diabetes Connect had its *android:allowBackup* tag set to "true". This can be leveraged to obtain e-mail addresses and passwords. These are not hard to obtain, as this data is stored unencrypted. Authentication tokens in this app consist of the base64 encoded e-mail addresses and password, separated by a colon. Since a malicious actor can obtain the e-mail address and password as described in this paragraph, the actor can generate the

authentication token. Having the authentication token, the malicious actor can invoke API calls to both read and modify the data stored server-side. The malicious actor does need physical access to the device or root access to the device.

This attack is possible because of multiple security flaws. The most desirable mitigation for the developers to implement is to encrypt any sensitive data that is stored locally and store passwords as a hash whenever it is necessary to store these. Furthermore, it is necessary to generate authentication tokens randomly and change them for every login, as the authentication tokens can be used to obtain a password in their current state. Finally, it is desirable to set the *android:allowBackup* to false if possible.
Users can mitigate this risk by implementing strong physical security.

For this next attack, it is worth noting that the app crashed when we performed this attack, and therefore we cannot prove that this attack can be successful in practice. While we could not investigate the nature of the crash completely due to time constraints, we do believe that the attack is possible. We believe that it is possible to use the exported activity to store all of the medical data of this app elsewhere on the device. A malicious app could do this and subsequently read the medical data. The most desirable mitigation for the developers to prevent this attack is to require a permission in order to invoke this activity. Users can mitigate this attack by preventing the installation of malicious apps.

5.2 Overlapping issues

5.2.1 Local storage

None of the apps used encryption for the data in local storage. In all apps, this data contains some form of personal data and medical data. Accessing locally stored data does require root privilege, but there are ways in which a malicious app, for example, can root a phone[25]. Therefore, the sensitive data that is stored locally would be safer if it were encrypted.

5.2.2 Password Policy

None of the apps enforced a strong password policy. Glucose Buddy and Diabetes:M required 8 characters and Diabetes Connect required 6 characters and that at least one letter and one number would be used in the password. Not forcing a user to create a longer password makes users likely to use short passwords [26]. Passwords of only 8 characters in length are fairly easy to crack in case the password hashes are leaked in a data breach. Furthermore, it is likely possible to guess passwords of 6 to 8 characters when trying to log in with a dictionary attack or a credential stuffing attack, which is an attack where passwords from previous data breaches are used to attempt a login into other accounts. Requiring at least one letter

and one number to be used makes a dictionary attack or credential stuffing attack slightly less effective but does not mitigate the risk completely.

5.2.3 Connection security

None of the domains to which the apps connected implemented DNS CAA, but two of the three domains had their TLS certificate registered in CT logs. These technologies are used to prevent the use of a wrongly issued certificate. The likelihood of an attack occurring, which involves a wrongly issued certificate is low, but the impact could be great [27]. DNS CAA is not a widely used technology, however, as of January 7th, 2019, only 4,3% of websites indexed by SSL labs used DNS CAA [28]. This indicates that DNS CAA is currently not standard practice in security. For two of the apps, this is also not a big security risk, as they can identify wrongly issued certificates through CT. Mitigating attacks involving wrongly issued certificates can also be mitigated through SSL pinning, yet none of the evaluated apps implement this technique.

All servers the apps are communicating with support a few weak encryption cipher suites. This makes TLS communication vulnerable to downgrade attacks. These attacks make it possible to view and modify the data being sent.

5.2.4 Add-ons

All of the analyzed apps use analytics add-ons and two of the apps used advertisement add-ons. Although analyzing which data is sent through these add-ons was not possible due to time constraints, it is likely that these add-ons collect and send sensitive data about the app usage without the user's consent. This leads to third parties being in possession of this data without the user's knowledge or consent. Furthermore, having more parties in possession of this data leads to a higher chance of data being compromised in a data breach.

5.3 Future work

Many subjects were left outside of the scope of this research. This research focused on apps created for the Android operating system. It could be interesting to look at apps created for another smartphone operating system to see if security is implemented differently in those apps. iOS is another widely used smartphone operating system [16], which makes it an interesting candidate for this topic of research.

In this research, only a few apps were evaluated. It could be interesting to expand this research and evaluate more apps and to include apps that have paid-for features.

We did not do invasive testing of the server-side security of these apps. Comprehensively testing the server side security of diabetes apps could be another interesting subject of research. This kind of research would require the permission of each app vendor.

Lastly, we also did not include sensors that connect to smartphone apps. The fact that the measurement data is transferred automatically to the smartphone poses additional security issues in relation to medication safety, which makes it an interesting topic for further research.

6 Conclusion

We examined the current state of security in glucose monitoring apps in this paper. From the research we performed, it can be seen that some kind of data can be obtained for all the tested apps. In all cases, this is glucose levels, but in some apps, you are also able to log your blood pressure, heart rate, and medications. Other personal information, such as e-mail address, full name, gender, etc., can also be obtained.

We found for the tested apps this data can be extracted via the use of improper platform usage, insecure data storage, insecure communication, and insecure authentication. Insecure data storage and authentication pose the largest risks. Furthermore, the authentication vulnerabilities could be used in such a way that we were able to modify the data, which was the case for two out of the three apps. This can be done via abusing insecure authentication tokens. Additionally, to exploit the vulnerabilities found in this research, physical access to the device is needed or the vulnerabilities can be exploited via a malicious app. The app developers should make glucose monitoring apps as safe and secure as possible, yet users can also take action to some extent in order to secure the data processed by these apps.

References

[1] Margaret Chan. "Global report on diabetes." In: *World Health Organization* 58.12 (2014), pp. 1–88. ISSN: 1098-6596. DOI: 10.1128/AAC.03728-14. URL: http://apps.who.int/iris/bitstream/handle/10665/204871/9789241565257_eng.pdf?sequence=1 (visited on 01/08/2019).

[2] Ralf-Gordon Jahns. *The addressable market for diabetes apps in 2016 has increased to 135.5M potential app users (diagnosed diabetics), out of which 4.1M are active users.* URL: https://research2guidance.com/the-addressable-market-for-diabetes-apps-in-2016-has-increased-to-135-5m-potential-app-users-diagnosed-

diabetics-out-of-which-4-1m-are-active-users/ (visited on 01/16/2019).

[3] *Health Data in the workplace*. URL: https://edps.europa.eu/data-protection/data-protection/reference-library/health-data-workplace_en (visited on 01/25/2019).

[4] *The Real Threat Of Identity Theft Is In Your Medical Records, Not Credit Cards*. URL: https://www.forbes.com/sites/forbestechcouncil/2017/12/15/the-real-threat-of-identity-theft-is-in-your-medical-records-not-credit-cards/ (visited on 01/26/2019).

[5] *WHY ARE HACKERS TARGETING YOUR MEDICAL RECORDS?* URL: https://luxsci.com/blog/hackers-targeting-medical-records.html (visited on 01/26/2019).

[6] *Hundreds arrested for $900 million worth of health care fraud*. URL: https://edition.cnn.com/2016/06/23/health/health-care-fraud-takedown/index.html (visited on 01/26/2019).

[7] *Hackers stole photos from top plastic surgery clinic in London, threaten to distribute them*. URL: https://www.foxnews.com/tech/hackers-stole-photos-from-top-plastic-surgery-clinic-in-london-threaten-to-distribute-them (visited on 01/26/2019).

[8] Konstantin Knorr and David Aspinall. "Security testing for Android mHealth apps". In: *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. IEEE. 2015, pp. 1–8.

[9] B Hewitt, D Dolezel, and A McLeod Jr. "Mobile Device Security: Perspectives of Future Healthcare Workers". In: *Perspectives in health information management* 14.Winter (2017). ISSN: 15594122. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85045048263&partnerID=40&md5=974ec6b5e4790bad1d85e9992d1f5275.

[10] *Insulin Overdose: Signs and Risks*. URL: https://www.healthline.com/health/diabetes/insulin-overdose#od-symptoms (visited on 01/26/2019).

[11] Sohail Khan et al. "How secure is your smartphone: An analysis of smartphone security mechanisms". In: *Proceedings 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic, CyberSec 2012* (2012), pp. 76–81. ISSN: 01676636. DOI: 10.1109/CyberSec.2012.6246082.

[12] Adrienne Porter Felt et al. "Android Permissions Demystied.pdf". In: (), pp. 627–637.

[13] Konstantin Knorr, David Aspinall, and Maria Wolters. "On the privacy, security and safety of blood pressure and diabetes apps". In: *IFIP International Information Security Conference*. Springer. 2015, pp. 571–584.

[14] Achilleas Papageorgiou et al. "Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice". In: *IEEE Access* 6 (2018), pp. 9390–9403. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2799522.

[15] Muzammil Hussain et al. "A security framework for mHealth apps on Android platform". In: *Computers and Security* 75 (2018), pp. 191–217. ISSN: 01674048. DOI: 10.1016/j.cose.2018.02.003. URL: https://doi.org/10.1016/j.cose.2018.02.003.

[16] *Mobile Operating System Market Share Worldwide - December 2018*. URL: http://gs.statcounter.com/os-market-share/mobile/worldwide (visited on 01/26/2019).

[17] *Genymotion Emulator home page*. URL: https://www.genymotion.com/ (visited on 01/24/2019).

[18] *Android Debug Bridge (adb)*. URL: https://developer.android.com/studio/command-line/adb (visited on 01/29/2019).

[19] *Mobile top 10 2016*. URL: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10 (visited on 01/26/2019).

[20] *Mobile Security Framework github page*. URL: https://github.com/MobSF/Mobile-Security-Framework-MobSF (visited on 01/10/2019).

[21] *SSLlabs home page*. URL: https://www.ssllabs.com/ (visited on 01/28/2019).

[22] *Burp Suite home page*. URL: https://portswigger.net/burp (visited on 01/10/2019).

[23] *Dozer home page*. URL: https://labs.mwrinfosecurity.com/tools/drozer (visited on 01/10/2019).

[24] *Addons Detector home page*. URL: https://public.addonsdetector.com/ (visited on 01/26/2019).

[25] *Dvmap: the first Android malware with code injection*. URL: https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/ (visited on 02/09/2019).

[26] Troy Hunt. *The science of password selection*. URL: https://www.troyhunt.com/science-of-password-selection/ (visited on 02/04/2019).

[27] Nicole van der Meulen. "DigiNotar: Dissecting the First Dutch Digital Disaster". In: *Journal of Strategic Security* 6.2 (2013), pp. 46–58. ISSN: 1944-0464. DOI: http://dx.doi.org/10.5038/1944-0472.6.2.4.

[28] *SSL Pulse*. URL: `https://www.ssllabs.com/ssl-pulse/` (visited on 01/29/2019).