Security Testing for Android mHealth Apps

Konstantin Knorr
Trier University of Applied Sciences, Germany
Email: knorr@hochschule-trier.de

David Aspinall
University of Edinburgh, UK
Email: david.aspinall@ed.ac.uk

***Abstract*—Mobile health (mHealth) apps are an ideal tool for monitoring and tracking long-term health conditions; they are becoming incredibly popular despite posing risks to personal data privacy and security. In this paper, we propose a testing method for Android mHealth apps which is designed using a threat analysis, considering possible attack scenarios and vulnerabilities specific to the domain. To demonstrate the method, we have applied it to apps for managing hypertension and diabetes, discovering a number of serious vulnerabilities in the most popular applications. Here we summarise the results of that case study, and discuss the experience of using a testing method dedicated to the domain, rather than out-of-the-box Android security testing methods. We hope that details presented here will help design further, more automated, mHealth security testing tools and methods.**

I. Introduction

Mobile health (mHealth) applications undertake health-related activities, helping users monitor and manage their own health. They can be used to provide data to doctors and other healthcare professionals, improving service, reducing costs and bringing potential advances through data sharing. There is a diverse range of mHealth apps: from physical measurements such as heart rate monitoring, blood pressure and glucose level recording (perhaps assisted through Bluetooth enabled devices or sensors in patches); through to tools for recording sleep patterns, moods, and providing medication reminders.

The number of users of mHealth apps is increasing dramatically with the ubiquity of smartphones and availability on Google's Android and Apple's iOS platforms. Market analysts project that 500 million smartphone users will use mHealth apps this year, rising to 1.7 billion by 2017. The number of Google Play and iOS mHealth apps passed 100,000 in 2014; estimates are that hundreds of million vital parameters are collected each month [1].

This trend raises serious security and privacy concerns. Healthcare data is one of the most sensitive kinds of personal information; yet in the US in 2014, the Identity Theft Resource Center found that over 40% of thefts were in the medical and healthcare category, with 8 million records stolen[1]. Clearly, mHealth apps that collect private data have a responsibility to protect their users against data loss. Unfortunately, research by us and by others has demonstrated that many mHealth apps that collect and transmit private medical data often do a spectacularly bad job of securing it.

Some form of carefully designed, transparent, reproducible security testing method for mHealth apps is therefore highly desirable. Current regulation only marginally addresses security requirements of mHealth apps, and doesn't propose any testing process. The US HIPAA (Health Insurance Portability and Accountability Act) focuses on hospitals, insurance companies and doctors and not developers of mHealth apps. The US FDA (Food and Drug Administration) only regulates a small portion of mHealth apps which qualify as "mobile medical devices" and intentionally excludes apps "that provide or facilitate supplemental clinical care, by coaching or prompting, to help patients manage their health in their daily environment"[2] — this loophole is actually viewed as an advantage in the industry to enable innovation, without the burden to obtain certification.

In this paper, we introduce a threat analysis for mHealth apps, and motivate a security testing method with a case study designed specifically for Android apps that monitor hypertension and diabetes. A main point of our study is that by focusing on a specific sub-category of apps, more precise security (and safety) issues can be addressed, influencing the type of tests performed and their parameters. Our testing method is thus context-sensitive, and also multifaceted: as well as apps we propose testing their associated web servers and evaluate privacy policies. We have demonstrated the method is effective by testing over 150 apps, highlighting a number of flaws. The full results of the case study are reported in a companion paper [2]; here we focus on the design and evaluation of the testing method itself, including the threat model.

Our contribution brings together and extends previous work, which has mainly considered vulnerability types on a point-by-point basis, perhaps inventing new tests (e.g., permissions usage [3], [4] or correct SSL/TLS handling [5]), or has considered threat modelling for mHealth without going down to the detail of testing [6]. What emerges is a testing strategy which is a synthesis of known tools and techniques, with some novel, context-specific extensions.

The rest of the paper is structured as follows: Sect. II considers the threats and vulnerabilities of the mHealth setting, including possible attackers. Sect. III describes the proposed testing method. Results of the case study of testing diabetes and hypertension apps are summarised in Sect. IV, and Sect. V discusses the performance of the tests. Sect. VI concludes.

2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
6th International Workshop on Security Testing (SECTEST 2015)
978-1-4799-1885-0/15/$31.00 ©2015 IEEE

[1]http://www.idtheftcenter.org/images/breach/DataBreachReports_2014.pdf

[2]http://www.fda.gov/downloads/MedicalDevices/.../UCM263366.pdf

II. Threat and vulnerability analysis

First we define the context. We consider the category of mHealth apps concerned with *monitoring and recording biological measurements* which might be entered manually or via a connected device. Being more specific, we focus on apps for hypertension and diabetes, two of the most common long-term health conditions suffered in the developed world. Self-monitoring is believed to be a useful intervention in both cases.

Blood pressure apps record the numeric measurements of systolic and diastolic blood pressure (in mmHg), and sometimes heart rate (bpm). Diabetes apps record blood glucose concentration (mmol/L or mg/dL). In general, there are ranges of values (that may vary across the day) which are *normal* (good) in the population as a whole, *abnormal* which may be a symptom of the condition being monitored. Other values may be considered *critical* or outright *invalid*.

We consider these data items, together with the fact that they are being monitored at all, to be confidential information which should be treated in a privacy-respecting way (i.e., if data is shared with a third party, that third party themselves has a duty of care for the data). Moreover, in case the data is being used to raise alarms or make treatment decisions, the integrity and validity of the data is important to maintain.

A. Threat modelling

A complete threat and risk analysis is beyond the scope of our work, but we introduce some initial thinking that provides a framework. We consider three types of threat:

- **Unauthorised learning of health data.** Somebody discovers the health information belonging to an individual, including the fact that an individual might be using a health-monitoring app.
- **Tampering with health data.** An attacker alters the health data that is recorded or reported by an app.
- **Reporting invalid health data.** An app reports wrong information to the user or healthcare professional.

The first type of threat represents a breach of confidentiality or privacy, the second of integrity, and the second and third raise concerns of safety (incorrect medication is issued) or availability (legitimate cases are denied service due to false alarms).

What is the data specifically and how do we place a value on it? A simple approach is to rank it by information content:

Type of data	Value
pseudonymous usage of app	low
user of app + public identity	medium
aggregated measurements	high
full detailed measurements	highest

Of course, this assignment belies that some patients value their privacy more than others. Some may like to publish details of their blood pressure tracking on social media (e.g., pseudonymously on Twitter or against a public profile on Facebook): a feature provided by many apps. Others users may avoid fine-grained sharing but happily reveal their usage of apps, e.g., by app store reviews. We don't separate these cases, but a fuller analysis might introduce different personas.

Next we consider threat agents. Some examples are:

- **Health insurance companies** who may seek to gain advantage by learning health information which is not normally part of their review procedures.
- **Employers** who wish to vet potential recruits, or monitor existing employees performance or work absence.
- **Data intelligence companies** who accumulate and sell demographic information about individuals.
- **Investigators** who seek to compromise privacy of celebrity individuals and other public figures.

We won't go further here, but a deeper analysis could consider particular scenarios and a risk analysis based on judgements of likelihood of specific threats by specific actors, etc.

B. Attack surface and vulnerabilities

Fig. 1 shows a smartphone with mHealth apps installed. Medical data are inserted via a GUI or using external medical devices via Bluetooth or NFC. The smartphone stores the app's data internally in a database or the file system. The SD card is used for database backups and restores, and to export reports. Mobile communication networks are used for emergency SMS and calls.

This picture shows several types of servers that may be connected. The App Shop is used to search for, download, rate and possibly pay for apps. The Ad and Analytics Infrastructure allows apps to display ads provided by ad servers (e.g., `AdMob` by Google) and tracks users' interaction with the apps. Many mHealth apps provide a gateway to a dedicated App Web Server, to allow upload, backup and synchronization of data; such web sites may also interface to other parties like insurance companies, doctors, or hospitals. Some apps connect to Social Media servers such as Facebook or Twitter. Finally, apps provide connectors to External Storage services like Dropbox or Google Drive. Apart from this, apps may use other services, e.g., sending e-mails containing medical data.

This architecture suggests an attack surface for realising the threats described above, and suggests where an attacker may look for vulnerabilities. The attacker kinds are:

A1. **Eavesdropper**: He will be able to see any unprotected network traffic.

A2. **Active attacker on the network**: He will be able to delete, detour, modify data in transit and brute force user accounts.

A3. **Man in the Middle**: He is able to leverage improperly implemented SSL protocols by diverting the SSL tunnel over his own server thereby being able to read the formerly encrypted data in clear-text. Note that A1-A3 can attack Internet, short range and mobile communication networks.

A4. **App shop owners**: typical app stores maintain records of who has downloaded which apps, moreover they

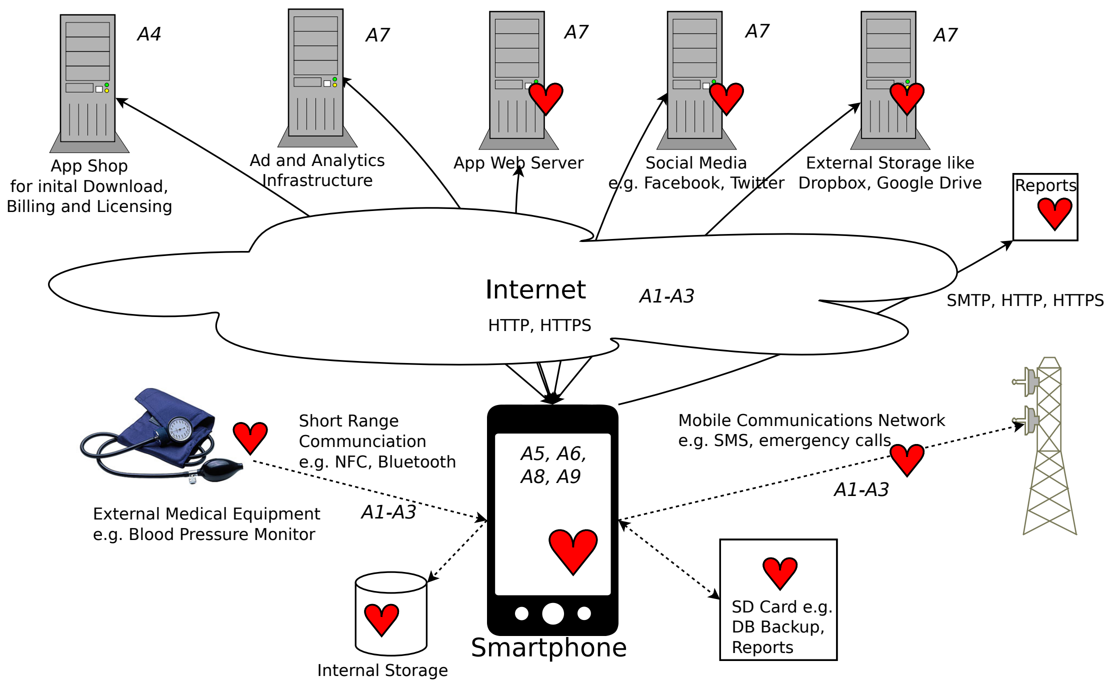

Fig. 1. An mHealth app in context. The heart indicates fine-grained medical data. *A1* – *A9* indicate possible attackers

may publish user ratings which explicitly reveal which apps users are (likely) to be using.

A5. **App developers**: they may insert trackers into their application to observe user behaviour, make mistakes that leak information, or even include malicious code.

A6. **Malware developer**: there may be separate malware on the phone which targets medical apps and transfers data over the Internet. Sensitive data on the phone (like logs or exports) can be accessed.

A7. **Third parties** like ad and analytics server, web server, social media sites, and external storage typically provide privacy policies that state how user data is handled. However, these policies can be missing, intrusive, or incorrect.

A8. **Attacker with physical access to smartphone**: we suppose he is able to extract the SD card and access unprotected data; he might also circumvent security mechanisms like PINs and encryption.

A9. **The user** of the app or the owner of the smartphone, who may wittingly or unwittingly release data.

Our testing method described next considers these attack modes. Others possible modes (side channels, protocol attacks, etc.) are currently not included.

III. Testing Method

Our testing method is illustrated in Fig. 2. It has four parts: (A) static analysis; (B) dynamic analysis; (C) web server security and (D) privacy policy inspection. Table I shows how our testing methods addresses three aspects: security, privacy and safety. Some tests help ensure more than one aspect: e.g., encryption may help provide confidentiality as well as privacy in case encrypted data is uploaded. Safety checking inputs for valid ranges can also help ensure that vulnerabilities like SQL injection or XSS are impossible. Also, it is indicated whether manual testing or tools were used. To underline the need for all four parts and three categories of testing, we give sample vulnerabilities found.

We now list the test schedule. For each test, we indicate the attacker who is being tested against (A1–A9 in the previous section). The initial download (or subsequent rating) of an app immediately allows the App Shop to learn information (A4).

A. Static analysis

Static analysis is based on the information contained in the APK file, including the manifest and the compiled code (in the file `classes.dex`). Tests are performed on a workstation and the device.

As the first step we extract the apps's permissions, which tell what actions (like Internet, Bluetooth, NFC usage or access to the contacts in the address book) are allowed. The permissions decide on the applicability of certain tests (no INTERNET usage, no MITM attack). The tool `AndroGuard` [7] is used for this. Then we test:

- Proper SSL usage (A3): Faulty SSL implementations or failure to check the entire certification path results in potential MITM attacks, lacking encryption, or missing replay and integrity protection which would allow access all medical data transmitted and other attacks. Faulty usage of SSL can be identified with `MalloDroid` [5].
- Debug flag (A8): The debug flag for Android allows dynamic inspection of the app, even when running in user mode. This can reveal medical information. We check this with `Drozer` [8].

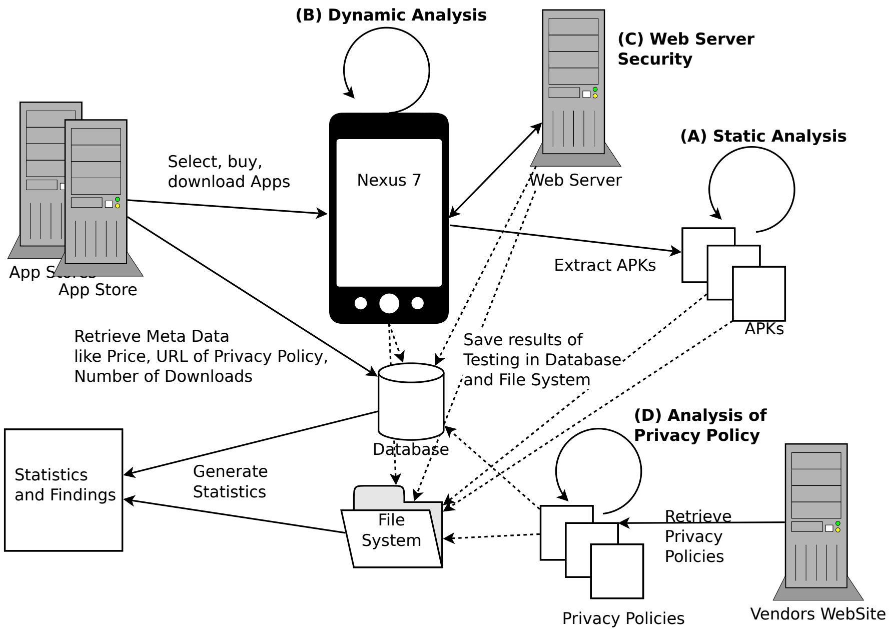

Fig. 2. Testing Method

TABLE I. STRUCTURE OF SECURITY TESTING

	What is tested?	How is it tested?	Sample vulnerability
Security	Static code analysis off and on phone (A)	Tools e.g. `MalloDroid`, `Addons Detector`	Faulty SSL implementation → MITM attacks Outdated third party libraries used
	Dynamic analysis (B)	Manual with tool support e.g. `tpacketcapture`, `adb`	Password or medical data in cleartext Unencrypted medical data in backup
	Web security (C)	Manual	Empty/simple password
Privacy	Analysis of privacy policies (D)	Manual. Checking policy for 1. base info, 2. completeness based on OECD, 3. invasiveness	Policy missing or outdated (1.) Description of user rights missing (2.) Medical data shared with 3rd parties (3.)
Safety	Input validation of sel. medical data (B)	Manual. Checking input fields in GUI	Lethal values accepted Junk input accepted → SQLi, XSS, XSRF

- Content providers (A5, A6, A8): Android's `ContentProvider` class allows sharing data between applications, with its own access control model. When sharing a provider's data with other apps, access control should be carefully implemented to prohibit unauthorized access to sensitive data. Again, `Drozer` helps check this.
- Use of encryption (A1, A8): To find use of encryption, we look for (standard) cryptographic functions by examining the disassembled `smali` code produced with the `apktool` [9].
- Poor use of certificate parameters (A1, A3): Android uses X.509 certificates to sign apps. To find certificates with poorly chosen parameters, `OpenSSL` [10] can extract certificate information from the APK files.
- Code quality (A5, A6, A8): `FindBugs` [11] can be used as a rough measure by counting the number of likely-bug patterns; high numbers indicate a likely poor code quality, which suggests possible unreliable behaviour, giving security and safety risks. FindBugs analyses a JAR file, `dex2jar` [12] converts.
- Add-ons (A7): advertising and analytics pose a threat: information shared may reveal a medical condition or more specific data. The tool `Addons Detector` [13] can identify and classify add-on libraries.
- Malware and Privacy scanners (A5): a range of dedicated apps from security vendors can be used to detect vulnerable, malicious or privacy-leaking apps. Specifically, we used `Snoopwall Privacy App` [14], `Clueful` [15], `AVG Antivirus Security` [16], `AVAST` [17], `McAfee Security & Antivirus -- FREE` [18] and `Recap vulnerability scanner` [19].

B. Dynamic analysis

Dynamic analysis is mostly manual; the process is difficult to automate (see discussion in Sect. V). As a preparatory step, we set up accounts on Facebook, Twitter, Dropbox, and Gmail for a dummy patient for whom some values are out of the physiologically normal range (to act as clear signals and boundary checks).

We then investigate whether abnormal and illegal inputs are accepted, and how exported data is stored or transmitted.

The tests are:

- Input validation (A9): Does the app accept abnormally high input values? With warning? Can junk values (non-numeric) be entered?
- Data leakage (A1, A6, A8): Test all available export routes (e-mail, SD card, web server, social media) to see if data was stored or transmitted unencrypted. Network is sniffed with `tPacketCapturepro` [20] (which can filter network traffic using process ids) and `DroidWall` [21][3]. Traces are inspected with `Wireshark` [22].
- Data wipe (A8): Check if the app includes a feature to erase all stored medical data?
- Privacy policy (A7): Is (a reference to) a privacy policy given in the app?
- Reasonable permissions (A5): Comparing permissions in the manifest with app features can reveal if an app seems over-privileged: a sign of risky programming (or possible backdoors).
- Secure backup and logging (A6, A8): We test whether backups or log data contained unencrypted medical data. This is done via `adb` [23].

C. Web server connection

For apps that interface with a dedicated web server to backup or synchronise medical data, we analyse the following security issues related to web security. As a preparatory step, a user account is needed on the web site corresponding to the dummy patient.

- Web Server connection (A1–A3): Plain HTTP allows an eavesdropper to read the user's medical data; by recording URLs we can see if a secure transport (`https:`) is used. We record traffic to see if passwords or medical data in textual or graphical form could be sniffed in clear text.
- Web Server authentication (A2): We investigate the effectiveness of the server authentication, for example by checking password policies. If weak passwords are allowed, an attacker may guess or brute force entry to the server.

Web server tests can be performed with any web browser, and network traffic analysis with `tPacketCapturepro` and `Wireshark`.

D. Inspection of privacy policies

Storage of medical data on an external server further stresses privacy concerns (and enables attacker A7). To check for weaknesses here, we propose inspecting privacy policies that are (or ought to be) provided.

- Collect basic privacy policy information: URL to privacy policy, number of words (indicating coverage), version and country of origin (jurisdiction).
- Completeness: We pose questions to test OECD privacy principles [24], answering "Yes", "No", "Partly":
    1) Accountability: Is the data controller named or detailed contact data given?
    2) Security safeguards: are safeguards described?
    3) Openness: are types of data collected described?
    4) Purpose specification: is the purpose and usage of the data selected described?
    5) Individual participation: are the rights of the individual described?
- Invasiveness: We pose three questions answered on the scale "Yes", "No", "Partly":
    1) Can medical data be used for other purposes like marketing or research?
    2) Is the data stored by a third party?
    3) Is it passed on to 3rd parties generally, or in case of company mergers/acquisition?

These checks are performed manually.

[3]requiring a rooted device

IV. Case Study

This section gives an outline of our test results for a hypertension and diabetes case study; more detailed and comprehensive discussion is in our companion paper [2].

A. App selection and preparation

In November 2014 we selected mHealth apps for diabetes and blood pressure which: (1) had an English or German user interface; (2) would run on a Google Nexus 7 test device running Android 4.4.2; (3) had over 10,000 downloads (for free apps) or over 1,000 (for paid). Most apps stem from the medical category of the stores and where specifically developed for diabetes and blood pressure monitoring. The most popular apps are `Blood Pressure (My Heart)` and `Blood Pressure (BP) Watch` which each have between one and five million downloads in Google Play.

We ended up with 154 apps (55% diabetes, 35% blood pressure, 10% both). The static analysis tests (A) were applied to all these apps. For dynamic analysis (B), we took the 72 most frequently downloaded apps. Finally, 20 apps were addressed in parts (C) and (D).

For our dummy patient, we invented a persona for someone 170 cm tall, weighed 99 kg, had dangerously high blood pressure of 200 mmHg/120 mmHg, a physiologically improbable heart rate (333 bpm), and a blood glucose level of 111 mmol/L, which is lethal and perhaps suggests a user confused mg/dL with mmol/L.

B. Test environment, database and other tools

Our test device was a Nexus 7 running Android 4.4.2. We use a laptop running Ubuntu Linux 12.04 to connect to the device and to run most of the tools. We created several `Python` (v2.7.3) scripts for data extraction, conversion and transferring. The `Google Play Unofficial Python API` [25] was used to extract meta data like prices, download numbers, ratings and permission from Google's Play Store.

For the sake of reproducibility and transparency we devised a file directory structure and database schema to store the intermediate and final results of our testing method. The file directories stores for each app (identified via its package name) the results of the tools (one directory per tool) and the manual analysis, e.g., log files, backup files, screen shots, network traces and privacy policies.

The major findings are extracted from these files (mainly using `Python`) or inserted via a web based GUI directly into the database, cf. Fig. 2. We used a `MySQL` database and `phpMyAdmin` running on an `Apache` web server. We make a database dump available for inspection[4].

C. Results of testing

Our main in-the-large findings were:

- **Encryption of health data is hardly ever provided.** Only one of the tested apps allows to encrypt reports, charts, and tables of medical data. As most apps allow to send e-mail or store data on SD card this means that the medical data remains unprotected when exported or sent (or relies on other Android security mechanisms).
- **Validation of health data is sketchy.** We found that many apps failed to make bounds checks in health data being input, recording invalid (or fatal) values.
- **Advertising is common and leaks package identities.** Of the apps tested, 74 include advertisement addons like `AdMob`. These addons often transmit the app's package name in clear text in the HTTP header which discloses the usage of this app (which is, per se, sensitive) to any passive eavesdropper.
- **Privacy policies are often missing or inadequate.** The majority (over 80%) of apps do not link to their privacy policy in the corresponding app stores. A potential user therefore has no or only limited possibility to check the privacy issues prior to installation and usage of the app. Even when privacy policies are provided, we found that they were often inadequate.

Beyond these high-level findings, the specific tests produced a wealth of detailed information about failings of specific applications, available in our database and described in [2].

V. Discussion

The effectiveness of our testing method is demonstrated by unveiling several critical security, safety, and privacy issues. We discuss aspects of the testing experience here.

Automated vs. manual testing. Most steps of the testing in parts (B), (C) and (D) in the dynamic analysis were manual. Automated dynamic analysis tools like `Droidbox` [26], `Tracedroid` [27], `CopperDroid` [28] either use random GUI inputs or require user interaction. Even with a uniform category of apps, it is difficult to automate the process reliably because of the diversity of GUIs and input designs (spinner wheel vs. free text, etc.) used by the apps.

[4]http://tinyurl.com/mhealthapps

Several tests in (C) like systematically checking different passwords should only be done with the web applications owner's consent and could be interpreted as illegal actions in some countries. We therefore only performed carefully selected, manual, non invasive "soft" checks and refrained from using tools.

Reading and interpreting the privacy policies in (D) needed to be done manually, since standards for machine-readable (or human comprehensible) policies like P3P (Platform for Privacy Preferences) have not been successful so far. Interpreting privacy policies is the most subjective stage of our testing; besides being open to interpretation, we must also take it on faith that policies are being complied with by their owners.

Coverage. A rough estimate of the coverage of our testing parts (A) and (B) can be given by comparing with established secure coding and general testing guides. Compared to the CERT secure coding guidelines for Android [29] our method covers 9 of 27 test cases, 8 are uncovered (5 of which are related to Android Intents), 10 (like OAuth, NDK, or Geolocation API test cases) are not relevant for our test set.

Comparing to OWASP [30] our method entirely or partly covers 7 of 8 static categories and 3 of the 7 dynamic categories. The OWASP list is by far too comprehensive to be applied to a larger number of apps and would need to be filtered for test cases relevant for our Android mHealth apps.

Regarding coverage for privacy policies, we performed only a superficial check. Possible extensions are given in Sect. VI-B. However, we claim this provides a first insight into the completeness and invasiveness of the privacy policies.

A fundamental challenge of app testing is the dynamic nature of the apps and the Android OS. New versions of apps appear regularly, often adding new features and changing permissions which would require new testing. Also, Android is evolving. This includes changing interpretation of permissions, new security features, fixing of bugs and possibly the inclusion of new vulnerabilities. We used Android version "KitKat", which currently has around 40% market share. For a specific app, one could test on a range of Android versions, but this is infeasible for a large collection.

Test effort. Most time consuming besides designing the method is the manual testing. A lot of time is spent understanding GUIs, setting correct units, setting up user accounts and inputting medical data in the correct fields. Among the tools, `FindBugs` and `MalloDroid` are costly (up to several minutes per app on our test machine), but the total time is still dwarfed by the manual effort. We tried to keep the test effort low by sticking to automated tests wherever possible. E.g., we tested SSL MITM attacks using `MalloDroid` instead of dynamically testing using the `Burp Suite` [31].

False positives in tool output. Findings of static analysis tools can have false positives and (ideally) should be manually verified. For example, when `MalloDroid` indicates a "naive TrustManager" or "broken SSL certificate validation" an exploitation would require to set up a corresponding MITM test environment for each app. Similarly, when Drozer identifies a unprotected Android content provider, a precise security test should investigate whether medical data can actually be accessed. In practice, if testing would become part of an acceptance criteria, modifying the app to conform to good practice will be cheaper.

A. Related work

Apps for mHealth in general have been examined by others in the last few years. Avancha et al. provide a literature review [32] while Martinez-Perez et al. [33] present general frameworks for privacy in mHealth and a detailed discussion of the existing privacy regulations.

Schulke [34] warned that mHealth apps for Android and iOS are currently largely unregulated and pose health risks to patients when healthcare professionals are not involved in development. The FDA [35] proposes a generic process to address "cybersecurity" for medical devices starting from assets, threats and vulnerabilities and then deriving security controls, but without giving details about security testing. Other standards, such as the Common Criteria, are similarly heavyweight and will not be used for mHealth apps at scale. At the level of individual apps, a wealth of tools can be used for checking particular security aspects, penetration testing, forensics, etc. Useful pointers are [36]–[38].

Work closest to our contribution stems from [39]–[42]. Njie [39] manually analyses 43 popular iOS and Android health and fitness apps with a focus on network analysis; he mentions a need to restrict future studies to a specific category of apps. His tests fall into our parts (B) and (D). He et al. [40], [41] analyse 47 randomly selected iOS and Android mHealth apps in two studies regarding Internet usage, logging, content provider, SD cards, and usage of cloud services and Bluetooth. Their tests fall into our part (B) and are also based on a threat analysis. Adhikari et al. [42] examine 40 popular iOS and Android mHealth apps by answering 9 privacy questions yielding a privacy score; their tests fall into our parts (B) and (D).

Regarding part (C) we base our analysis on the well established OECD guidelines rather than devising new schemas like proposed by Adhikari et al. [42] and Sunyaev et al. [43].

In contrast to other studies, our work focuses (1) on apps for specific purposes, rather than selecting randomly from the broad category of mHealth apps and (2) only on Android apps. This gives us a more homogeneous population which can be closely tested and compared, in particular, it allowed us to check input validation for specific blood pressure and glucose reading ranges. Our testing goes further than previous work: while others have applied dynamic testing methods, we additionally apply static analyse techniques in part (A) and examine the web security aspects in part (C) which is novel for mHealth apps.

In part (B) the safety checking and creation of a virtual patient along with its electronic identity in the security testing context is novel. We are also the first in a mHealth security scenario to store our test results in a database and to make it accessible to other researchers.

VI. Conclusions and future work

A. Conclusions

Our security testing method found clear evidence of privacy, safety, and security concerns for the majority of the apps we analysed in the case study. Some issues, such as the pervasive lack of encryption, suggest security is not a priority for developers. Reports, charts, and tables of medical data are often stored without any protection, giving thieves and eavesdroppers easy access. Another important threat is advertising. Of the 154 apps tested, 74 include advertisement addons. These addons often transmit the app's package name in clear text in the HTTP header which discloses the usage of this app (which is, per se, sensitive) to eavesdroppers. We also pointed out that current malware and privacy scanners fail to identify privacy issues in mHealth apps. On data entry, we found that input data was often not validated or badly validated. This can result in reporting invalid data with healthcare professionals and possible false alarms.

App developers also rarely provide privacy policies. Although most users may not read such policies [44], they ought to provide accurate summaries for people who do review them and can raise awareness for others, including security auditors. Most policies we analysed fell far short of this goal.

Our testing method can be seen as a compromise regarding effort and depth between pure pentesting and the more formal security analysis as followed by FDA and the Common Criteria. For app populations comparable in size to ours, the testing could be helpful for consumer protection organisations or emerging app shops. Consumer protection organisations could systematically compare app groups regarding security, privacy and safety and give corresponding recommendations. Custom, curated app stores might provide selected sets of mHealth apps which have been reviewed by experts.

Our work shows that comprehensive security testing for Android mHealth apps is expansive due to the large amount of manual work. Some tools for automated testing are available but they cover only a small area and by far not the entire spectrum required. Fully automated checking is desirable but unrealistic today.

B. Future work

Our method is sensitive enough to uncover many privacy and security concerns, and it could be extended to apps for other kinds of conditions. We advocate examining the app's purpose and data collected in detail: while we considered limits on simple numerical data collections and direct data leakages, a more nuanced investigation might examine ways that detailed data can leak indirectly (for example, through summaries).

Additional tests can be easily added. E.g., the automatic identification of unused permission as proposed by Felt et al. in [45] or the automatic detection of common cryptographic errors (like using ECB mode) as proposed by Egele et al. in [46] could augment part (A). Other extensions are the test cases in [29] and [30] currently not included in our method. Part (D) can be extended by addressing further OECD categories and asking additional questions or by adding the privacy questions given in [39], [42], [43].

As apps become increasingly relied up for mHealth, other security aspects beyond the way they treat data will be important, such as their resilience to denial-of-service attack, or their failure modes when they become unavailable simply because of power loss. Usability (and the potential for accidental misuse) is another vital concern: there are documented cases where regulated, expensively certified professional medical devices have caused mistreatment because of user-interface confusion. This is a concern because the advertised self-monitoring role of many current mHealth apps is only a small step away from self-medication (and arguably, medical reminder apps already perform that role). Future security and safety testing for mHealth apps will need to consider the full spectrum of issues.

References

[1] 500m people will be using healthcare mobile applications in 2015. [Online]. Available: http://research2guidance.com/500m-people-will-be-using-healthcare-mobile-applications-in-2015/

[2] K. Knorr, D. Aspinall, and M. Wolters, "On the privacy, security and safety of blood pressure and diabetes apps," in *Proceedings of IFIP SEC 2015, International Conference on ICT Systems Security and Privacy Protection*, 2015.

[3] M. Frank, B. Dong, A. P. Felt, and D. Song, "Mining permission request patterns from android and facebook applications," in *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM)*, 12 2012, pp. 870–875.

[4] Y. Y. Pern Hui Chia and N. Asokan, "Is this app safe? a large scale study on application permissions and risk signals," in *Proceedings of the WWW 2012*, Lyon, Apr. 2012, pp. 311–320.

[5] S. Fahl *et al.*, "Why eve and mallory love Android: An analysis of Android SSL (in) security," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 50–61.

[6] D. Kotz, "A threat taxonomy for mHealth privacy," in *3rd International Conference on Communication Systems and Networks*, 2011.

[7] [Online]. Available: https://code.google.com/p/androguard/

[8] [Online]. Available: https://www.mwrinfosecurity.com/products/drozer/

[9] [Online]. Available: https://code.google.com/p/android-apktool/

[10] [Online]. Available: https://www.openssl.org/

[11] [Online]. Available: http://findbugs.sourceforge.net/

[12] [Online]. Available: https://code.google.com/p/dex2jar/

[13] [Online]. Available: https://play.google.com/store/apps/details?id=com.denper.addonsdetector

[14] [Online]. Available: https://play.google.com/store/apps/details?id=com.snoopwall.privacyapp

[15] [Online]. Available: https://play.google.com/store/apps/details?id=com.bitdefender.clueful

[16] [Online]. Available: https://play.google.com/store/apps/details?id=com.antivirus

[17] [Online]. Available: https://play.google.com/store/apps/details?id=com.avast.android.mobilesecurity

[18] [Online]. Available: https://play.google.com/store/apps/details?id=com.wsandroid.suite

[19] [Online]. Available: https://play.google.com/store/apps/details?id=com.palindrome.ph

[20] [Online]. Available: https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapturepro

[21] [Online]. Available: https://play.google.com/store/apps/details?id=com.googlecode.droidwall.free

[22] [Online]. Available: https://www.wireshark.org/

[23] [Online]. Available: http://developer.android.com/tools/help/adb.html

[24] "OECD guidelines on the protection of privacy and transborder flows of personal data." [Online]. Available: http://www.oecd.org/internet/ieconomy/oecdguidelinesontheprotectionofprivacyandtransborderflowsofpersonaldata.htm

[25] [Online]. Available: https://github.com/egirault/googleplay-api

[26] [Online]. Available: https://code.google.com/p/droidbox/

[27] [Online]. Available: http://tracedroid.few.vu.nl/

[28] A. Fattori, K. Tam, S. J. Khan, A. Reina, and L. Cavallaro, "CopperDroid: On the reconstruction of android malware behaviors," Royal Holloway University of London, Tech. Rep. MA-2014-01, Feb. 2014.

[29] CERT secure coding standards for Android. [Online]. Available: https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=111509535

[30] OWASP mobile security project - security testing guide. [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Mobile_Security_Testing

[31] [Online]. Available: http://portswigger.net/burp/

[32] S. Avancha, A. Baxi, and D. Kotz, "Privacy in mobile technology for personal healthcare," *ACM Computing Surveys*, vol. 45, pp. 1–54, Nov. 2012.

[33] B. Martinez-Perez, I. de la Torre-Diez, and M. Lopez-Coronado, "Privacy and security in mobile health apps: A review and recommendations," *Journal of Medical Systems*, vol. 39, no. 1, 2014.

[34] D. F. Schulke, "Regulatory arms race: Mobile-health applications and agency posturing, the," *BUL Rev.*, vol. 93, p. 1699, 2013. [Online]. Available: http://heinonlinebackup.com/hol-cgi-bin/get_pdf.cgi?handle=hein.journals/bulr93&section=67

[35] FDA. Content of premarket submissions for management of cybersecurity in medical devices - guidance for industry and food and drug administration staff. [Online]. Available: http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM356190.pdf

[36] A. Misra and A. Dubey, *Android Security: Attacks and Defenses*. Auerbach Publications, 2013.

[37] A. Gupta, *Learning Pentesting for Android Devices*. PACKT Publishing, 2014.

[38] K. Shah, "Penetration testing Android applications," 2010, white paper from Foundstone Professsional Services, McAfee. [Online]. Available: http://www.mcafee.com/it/resources/white-papers/foundstone/wp-pen-testing-android-apps.pdf

[39] Craig Michael Lie Njie, "Technical analysis of the data practices and privacy risks of 43 popular mobile health and fitness applications," PrivacyRights Clearinghouse, Tech. Rep., Aug. 2013. [Online]. Available: http://www.privacyrights.org/mobile-medical-apps-privacy-technologist-research-report.pdf

[40] D. He, M. Naveed, C. A. Gunter, and K. Nahrstedt, "Security concerns in Android mHealth apps," in *Proceedings of the AMIA 2014*, 2014. [Online]. Available: https://web.engr.illinois.edu/~naveed2/pub/AMIA2014.pdf

[41] D. He, "Security threats to Android apps," Master's thesis, University of Illinois at Urbana-Champaign, 2014.

[42] R. Adhikari, D. Richards, and K. Scott, "Security and privacy issues related to the use of mobile health apps," in *Proc. of the 25th Australasian Conference on Information Systems*. ACIS, 2014. [Online]. Available: http://aut.researchgateway.ac.nz/handle/10292/8117

[43] A. Sunyaev, T. Dehling, P. L. Taylor, and K. D. Mandl, "Availability and quality of mobile health app privacy policies," *Journal of the American Medical Informatics Association*, 2014.

[44] H. Nissenbaum, "A Contextual Approach to Privacy Online," *Daedalus*, vol. 140, no. 4, 2011.

[45] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638.

[46] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 73–84.