Original Paper

Server-Focused Security Assessment of Mobile Health Apps for Popular Mobile Platforms

Jannis Müthing[1], BSc; Raphael Brüngel[1], BSc; Christoph M Friedrich[1,2], PhD

[1]University of Applied Sciences and Arts Dortmund, Department of Computer Science, Dortmund, Germany

[2]Institute for Medical Informatics, Biometry and Epidemiology, University Hospital Essen, Essen, Germany

Corresponding Author:
Christoph M Friedrich, PhD
University of Applied Sciences and Arts Dortmund
Department of Computer Science
Emil-Figge-Straße 42
Dortmund, 44227
Germany
Phone: 49 231 9112 ext 6796
Email: christoph.friedrich@fh-dortmund.de

Abstract

Background: The importance of mobile health (mHealth) apps is growing. Independent of the technologies used, mHealth apps bring more functionality into the hands of users. In the health context, mHealth apps play an important role in providing information and services to patients, offering health care professionals ways to monitor vital parameters or consult patients remotely. The importance of confidentiality in health care and the opaqueness of transport security in apps make the latter an important research subject.

Objective: This study aimed to (1) identify relevant security concerns on the server side of mHealth apps, (2) test a subset of mHealth apps regarding their vulnerability to those concerns, and (3) compare the servers used by mHealth apps with servers used in all domains.

Methods: Server security characteristics relevant to the security of mHealth apps were assessed, presented, and discussed. To evaluate servers, appropriate tools were selected. Apps from the Android and iOS app stores were selected and tested, and the results for functional and other backend servers were evaluated.

Results: The 60 apps tested communicate with 823 servers. Of these, 291 were categorized as functional backend servers, and 44 (44/291, 15.1%) of these received a rating below the A range (A+, A, and A−) by Qualys SSL Labs. A chi-square test was conducted against the number of servers receiving such ratings from SSL Pulse by Qualys SSL Labs. It was found that the tested servers from mHealth apps received significantly fewer ratings below the A range (P<.001). The internationally available apps from the test set performed significantly better than those only available in the German stores (alpha=.05; P=.03). Of the 60 apps, 28 (28/60, 47%) were found using at least one functional backend server that received a rating below the A range from Qualys SSL Labs, endangering confidentiality, authenticity, and integrity of the data displayed. The number of apps that used at least one entirely unsecured connection was 20 (20/60, 33%) when communicating with functional backend servers. It was also found that a majority of apps used advertising, tracking, or external content provider servers. When looking at all nonfunctional backend servers, 48 (48/60, 80%) apps used at least one server that received a rating below the A range.

Conclusions: The results show that although servers in the mHealth domain perform significantly better regarding their security, there are still problems with the configuration of some. The most severe problems observed can expose patient communication with health care professionals, be exploited to display false or harmful information, or used to send data to an app facilitating further damage on the device. Following the recommendations for mHealth app developers, the most regularly observed security issues can be avoided or mitigated.

(J Med Internet Res 2019;21(1):e9818) doi: 10.2196/jmir.9818

KEYWORDS

mobile health; mobile apps; data security; computer security; confidentiality; health information technology; servers; data protection

Introduction

Mobile Health Apps

The ubiquitous availability of the internet and mobile devices facilitates new and powerful applications. Mobile health (mHealth) is describing health care–related usage of mobile devices [1]. Although some apps offer user-tailored health information, others facilitate easier communication between patients and health care professionals or offer shop systems for medication sale. mHealth apps have an inherently higher need for security. Besides the importance to protect health data of patients that are collected using apps, by ensuring authenticity of the communication partner and confidentiality of data in transit, information coming from apps must also be protected from unauthorized changes to the content displayed (thus data integrity must be maintained). In contrast to browser-based websites or Web apps, the security of connections used by (native) mobile apps is not transparent to the user. Web browsers display both the use of a secure connection and, importantly, display issues with connection security prominently [2].

In earlier research, client-focused transport security concerns regarding mobile apps were studied [3]; this study focuses on the security of server infrastructure of mHealth apps.

Server Security

The functionality of most apps relies on communication with a remote server over the internet. HTTP is the standard when it comes to client-server communication in the context of mobile apps [4] and offers no security features [5]. The communication through public infrastructure can potentially be observed, modified, or redirected. This endangers the integrity of data displayed by an app and confidentiality of the data sent to and received from a server and also could enable a malicious party to impersonate a server. Furthermore, a publicly reachable server must also guarantee availability [6].

The infrastructure between client and server represents an untrusted medium. Any third party in a privileged position between both communication partners can read and modify all data exchanged. A privileged position in this context is any hop on the path between them (bridges, routers, and gateways) [7]. A common attack vector is a technique called address resolution protocol (ARP) spoofing [7,8]. This enables the attacker to receive all requests normally intended for the router on the local network. Attacks where the third party is located between client and server and can read and modify messages are called man-in-the-middle (MitM) attacks. Another recent example enabling MitM attacks against clients of Wi-Fi Protected Access (WPA) 2 wireless networks is the Key Reinstallation Attack (KRACK) [9]. Older or unpatched protocols such as WPA and wired equivalent privacy (WEP) are also vulnerable to attacks, enabling traffic decryption or MitM setups [10,11]. For attacks on Wi-Fi infrastructure, physical proximity to the target is required. An attacker at least needs to be in the range of the victim's access point. For the KRACK attack, the attacker needs to be close to the victim's device.

To ensure confidentiality and integrity of data sent through an untrusted medium, the Transport Layer Security (TLS) protocol is used [12]. It is an important part of today's internet security infrastructure. TLS was designed for authenticity, integrity, and confidentiality protection of the underlying communication channel by offering secure authentication, data integrity protection, and confidentiality through asymmetric and symmetric cryptography. TLS is situated in the application layer of the transmission control protocol (TCP)/ internet protocol (IP) stack and can wrap and secure HTTP connections. TLS-secured HTTP connections are called HTTP Secure (HTTPS) connections [13].

Given certain server and client configurations, TLS can be set up to offer forward secrecy [14]. This means if secret keys are compromised in the future, past communication stays secure and cannot be decrypted with the compromised credentials alone.

Without this secure wrapper authenticity of a server, confidentiality of information exchanged with a server and the integrity of data sent to an app cannot be guaranteed, and the app might display arbitrary text, pictures, or video data. Because an attacker can potentially feed an app arbitrary input, missing integrity can also be exploited to make the app decode an image, video, or other data that could exploit vulnerabilities in decoders. The Stagefright exploit on Android devices, for example, relied on Android operating system (OS) processing modified media files [15].

TLS and its predecessor the Secure Socket Layer (SSL) protocol are not without security flaws. Since its introduction, multiple vulnerabilities in different layers of the protocols or implementations of the protocols were found and exploited to undermine their security [16]. To keep a server (and therefore patients) secure, special scrutiny and vigilance regarding new threats are required from server operators [17]. It is crucial to react quickly to the publication of new vulnerabilities. A malicious third party only has to test the exploitability of all known vulnerabilities to find a way to attack the server-client communication. Some well-known examples are the Padding Oracle On Downgraded Legacy Encryption (POODLE), Heartbleed, and the recent Return Of Bleichenbacher's Oracle Threat (ROBOT) that was made public in December 2017 [18-20]. To address newly found security issues in SSL and its successor TLS, new versions of the protocol are released regularly. Use of a newer version protects from known security flaws of older versions.

TLS relies on digital certificates to authenticate a server to clients [12,21]. These certificates must be issued (and signed) by certificate authorities (CAs) and have multiple characteristics that must be checked for a certificate to be valid for a given domain. Some characteristics are that a certificate must (1) be issued for the domain requested, (2) have a *valid from* date and must be before the current date, (3) have a *valid until* date in the future, and (4) must not be revoked.

A certificate's revocation status can be checked against a CA's certificate revocation list. As CAs are the roots of the chains of trust, they must operate responsibly. There are some CAs that offer certificate services for free (eg, the Linux foundation's *Let's Encrypt*), whereas others charge for certificates issued by them [22]. The management of the server certificates and

safekeeping of corresponding private keys are also crucial to the security a server can offer.

A general overview of security threats to internet-connected systems can be found at the Open Web Application Security Project (OWASP) [23]. The project collects information, tools, and best practices to avoid common security issues. The OWASP Top 10 and the OWASP mobile Top 10 are of great relevance to the research presented here [24,25]. Although not all vulnerabilities are relevant, the lists were a valuable starting point for the design of the tests.

Prior Work

In prior research, the transport security of mHealth apps from a client's perspective was investigated [3]. The research inspected the data exchanged between iOS and Android client apps and a server and evaluated it under security considerations. This also included the use of TLS and the TLS version. Furthermore, it considered the validation of server certificates by the clients. The study found severe problems with 40% of all 53 tested apps.

Existing literature also evaluated metadata of mHealth apps from Google's Play Store and Apple's App Store [26]. The study did not perform tests or technical analysis. A study of popular mobile apps from China found 97% of apps surveyed provided no information security [27]. The authors limited their investigation to the evaluation of available documentation and availability of auditing reports.

Other research focused on 22 mHealth apps and found that 18 of those apps send data unencrypted over the internet [28].

As mobile apps are not limited to stock HTTP(S) implementations, the study of HTTPS implementations in Android apps is relevant [29]. The publication discusses common flaws in TLS deployments and server configurations.

In the realm of the Internet of Things, existing research analyzed internet-connected toys for children in regard to security and privacy concerns [30]. The authors also bring attention to severe transport security issues of toys when communicating with their backend servers.

Servers have been in use for many years to serve websites and interfaces for internet-based services. The servers' purpose to supply mHealth apps with data and functionality is only one of their most recent use cases. Due to their inherent exposure to public infrastructure (the internet), they always offered an attractive attack surface. As a consequence, knowledge about problems and secure configuration of servers is widely available [5,6,31,32]. An overview of the general landscape of SSL/TLS security on the internet can be observed on Qualys SSL Lab's SSL Pulse website [33].

mHealth apps are getting attention by the media and are covered in regard to treatment of patient data [34]. There is an initiative to build a central place to rate apps in regard to privacy matters (among other criteria) [35]. PrivacyScore has similar objectives and offers a configurable interface to test for a number of security and privacy issues of websites [36].

Existing research mostly considers nontechnical characteristics of mHealth apps, client-side implementations of apps, or solely the use of any encryption at all. In contrast, this research will focus on the configuration of servers used by mHealth apps.

The Methods section will describe how the tested apps were selected. Furthermore, it presents and explains transport security issues for servers and lists the tools used to test for these issues. In the Results section, the tested apps are presented. The test methodology is explained before the aggregated results are listed. These results will be discussed, and common issues will be pointed out.

Methods

App Selection

In prior research, free apps from 3 different European app stores were selected. As differences in behavior between apps from different European countries were not found without loss of generality for this study, only free apps from the German app stores' top lists were chosen [3]. Many apps from the German top-downloaded lists are popular across other nations' app stores. The difference between internationally available and popular apps and apps only in the top lists of the German stores will be discussed in the Results section. To mitigate any platform-dependent bias, apps for Android and iOS are tested.

Relevant Server Security Considerations

HTTP by itself transmits information as clear text without any encryption. It is an application layer protocol and can be secured by being used on top of a secure TLS connection [4,12,13]. TLS and its predecessor SSL are designed to ensure the authenticity of communication partners, confidentiality between parties, and integrity of transmitted data. To achieve this, TLS uses asymmetric cryptography and public key infrastructure (PKI) for authentication and exchange of key material. Symmetric encryption is used for payload data encryption [37,38].

SSL and TLS use version numbers. As earlier versions of the protocol had serious security issues, this paper will take the version of SSL or TLS into consideration [16,18]. While SSL 2.0 is considered insecure because of structural vulnerabilities [39], the POODLE exploit enables third parties to recover plaintext from SSL 3.0-protected traffic [40,41]. Apart from SSL 2.0 and SSL 3.0, newer TLS versions do not have known security vulnerabilities if the server (and client) is properly configured. Lacking a proper configuration, older versions such as TLS 1.0 are vulnerable to an improved POODLE attack and other vulnerabilities [18]. Another TLS 1.0 vulnerability can only be efficiently mitigated by the clients: the Browser Exploit Against SSL/TLS (BEAST) [42]. Although most modern browsers do mitigate the issue, the security of the protocol is still not controllable on the server side. TLS 1.1 and later protocols are not vulnerable to such attacks and can be configured on the server side to use secure ciphers [43]. More recent versions include improvements that are considered more secure. The use of SSL/TLS and the lowest supported version number will be part of the evaluation. We also evaluated support for the recently approved TLS 1.3 (August 2018) and mentioned it in the Results section [44].

To ensure the use of HTTPS and prevent protocol downgrade attacks, HTTP Strict Transport Security (HSTS) can be used [45]. A downgrade attack is designed to get a client to connect to the server using an unsecured HTTP connection. This enables a malicious third party to perform a MitM attack and consequently read and modify sensitive information. For HSTS to be used, the server sends a special header in response to a request. This tells the client to only connect through secure HTTPS connections. For HSTS to work, both client and server have to support it. Although HSTS is most important for browser-based Web apps, many apps include Web components that use platform browser components. In addition, Web-based versions of apps often exist. The presence of HSTS headers in server responses will be listed in the Results section.

To provide a better understanding of the following considerations, a basic understanding of the TLS handshake is required [12]. This description will focus on the most common case of server authentication only.

The client initiates the connection by sending a *client hello* message. This includes its highest supported TLS version, a random value, suggested compression methods, and its supported cipher suites. The server, in turn, answers with the chosen protocol version, cipher suite, compression method, and a random value. In a *certificate message*, the server also sends its certificate. The client now creates a premaster secret, encrypts it with the server's public key, and sends it to the server. Both parties generate the master secret and session keys based on the premaster secret. The client sends a *change cipher spec* message to the server to inform the server that it will use the session key for hashing and message encryption. This is followed by a *client finished* message. The server receives this message, switches to symmetric encryption for further messages, and sends a *server finished* message.

During the TLS handshake, the client uses the servers' public key from its certificate to encrypt a premaster secret. The encryption algorithm is dependent on the negotiated cipher suite. The certificate sent by the server fits the negotiated cipher suite: if, for example, a cipher suite is chosen that includes the *Elliptic Curve Diffie Hellman Ephemeral* (ECDHE) algorithm for key exchange, the certificate must include an elliptic curve (EC) public key [46,47]. As explained in the TLS handshake, the security of the key exchange is essential for the security of the connection. If an adversary is able to decrypt the *premaster* key by brute force, the security of the TLS connection would be compromised. To make this harder, the algorithm used as well as the key length of the public key are important. A commonly used algorithm is the Rivest-Shamir-Adleman (RSA) algorithm [48]. Key sizes vary from 1024 to 4096 bit. It has been shown that a 1024 bit key does not offer sufficient security [49]. Moreover, 2048 bit is the commonly recommended lower limit for RSA keys. The added complexity and negative performance impact during the TLS handshake are disadvantages of the use of longer keys. Newer algorithms such as the EC algorithm do require smaller keys, less computational requirements for clients, and servers while offering equivalent security [47,49]. The key algorithm and length are part of this evaluation.

Another aspect related to the handshake is the selection of the cipher suite. A server has a number of supported cipher suites [12,50]. When the client sends its list of possible cipher suites, the server selects one it supports. The most secure cipher suite should be negotiated between client and server. A server can be configured to have an order of preference for cipher suites [17]. If present, the server will choose the suite highest in priority, which is supported by the client. Whether a server has a preferred order is part of the results of the study because of the importance of the chosen cipher suite for the encryption of user traffic.

For the same reason, the list of supported cipher suites will be considered. Although uncommon, in the worst case, cipher suites may define no encryption at all for the TLS traffic. Other cipher suites define algorithms that can be cryptographically attacked and should not be used anymore. This section does not list all ciphers and their vulnerabilities, but the use of insecure cipher suites is part of the evaluation.

The authentication of the server is based on the server's certificate [12,21,51]. A certificate must fit certain criteria to be considered valid by the client. It must be issued for the requested domain, must not be expired or revoked, and must be trusted. In a PKI, a client trusts root certificates issued by CAs. These CAs use the corresponding private key to sign certificates of servers (or other sub-CAs). When the client verifies the validity of a server certificate, it follows this chain of trust from the server certificate until one of the certificates in the client's trusted root certificates is referenced. Certificate (chain) issues are also part of the research performed for this study.

Older SSL and TLS versions are vulnerable to certain exploits undermining their security. In addition to vulnerabilities of older versions, implementation-dependent issues such as Heartbleed and others are relevant. Heartbleed is an issue in the popular OpenSSL cryptographic software library. It enables an attacker to read memory contents from the server if the library is not patched. Another recently (December 2017) discovered attack uses an issue in RSA implementations and makes the key exchange observable by attackers (ROBOT) [19]. A server that supports RSA for its key exchange, and that is using a vulnerable implementation, is at risk to be attacked [52]. A relative of the BEAST vulnerability discussed earlier also enables attacks against TLS 1.2. It is named the Compression Ratio Info-leak Made Easy (CRIME) and works in conjunction with the use of cookies by protocols that use data compression (such as HTTPS) [53]. It can be used to observe and use a client's authentication cookie to enable further attacks. The vulnerability can be counteracted on the client as well as on the server side. The Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) attack is a variant of CRIME and can be exploited to a similar effect [53]. A comprehensive overview of known attacks against TLS can be found as a Request For Comments (RFC) by the Internet Engineering Task Force (IETF) [53]. Vulnerability to known attacks is considered during the tests.

Because the physical location of a server has consequences for applicable law, the server location is considered in this study.

Specifically, this study lists whether servers are inside the European Union (EU), as it is considerably harder for service providers dealing with data belonging to European citizens to host these data outside the EU and still comply with the General Data Protection Regulation (GDPR) [54,55]. Explicit exceptions are GDPR-compliant servers outside the EU covered under the EU-US privacy shield [54,56]. As mentioned in the Limitations section, this study does not differentiate GDPR-compliant servers outside the EU, and servers are simply listed to be located outside EU territory.

Selection of Appropriate Test Tools

The first step of the tests is to find out which of the apps communicate with which backend. Similar to prior research, the BProxy tool was used to facilitate the first test phase [3,57]. This software acts as a proxy between a device running an app and the internet. It is open source and can be found on GitHub [57]. All traffic from the device is analyzed by the proxy, and a report is displayed. As this study is concerned with server-side transport security aspects, only the domain names of servers an app communicates with and the use of unsecured connections are of interest. As the proxy records all traffic to and from the device, filtering is necessary to find out which domains are part of the app's backend. Traffic was inspected to distinguish between app-related servers and other servers. To facilitate correct categorizations, the disconnect service was used and the results were inspected manually, similar to the methodology of a prior publication [3,58].

To test the servers behind the domains, the testssl script and the Qualys SSL Labs test suite were chosen [59,60]. The testssl script uses OpenSSL to perform tests against a targeted server from the local machine. It generates a comma-separated value (CSV) file containing all findings for the domain. The script checks for some relevant characteristics, including all described relevant concerns. The script is actively developed and maintained to contain tests for recently discovered vulnerabilities. At the time of testing, the ROBOT attack was relatively new and already included in the development version of the testssl script [19,59]. The SSL Labs test suite is a Web-based tool for SSL/TLS-related tests. Qualys SSL Labs also offers a command line reference implementation for test automation [61]. The results include similar attributes as the testssl script but importantly assign a graded score A to F. This score is the result of an automated evaluation of the characteristics and vulnerabilities observed. A guide on how this score is calculated and, consequently, how SSL Labs rates the severity of security characteristics can be found on GitHub [62]. This rating guide is updated by Qualys on a regular basis, and a changelog can be found as part of the guide mentioned previously. The rating consists of multiple considerations and includes all but the first item in Table 1: (1) validity and trust of the certificate used by the server, (2) supported protocols (SSL 1.0 up to TLS 1.3), (3) key exchange algorithms supported (older algorithms score lower because of security issues), and (4) cipher suites supported (if no secure, up-to-date cipher suites are supported, the grade will be lower).

Each category scores between 0 and 100. The scores are combined, which results in a single overall score for the server. A 0 in any category will result in an overall score of 0. Although a low score results in an overall lower result, a 0 in a category is indicating a fatal security issue. An example for the score calculation in the supported protocols category according to the Qualys SSL Labs' server rating guide looks like this:

1. Start with the score of the best protocol.
2. Add the score of the worst protocol.
3. Divide the total by 2 [62].

Table 2 lists the scores that Qualys SSL Labs assigns each supported protocol version at the time of writing. A 0 in this category, for example, can only occur if only SSL 2.0 was supported. As this is a long outdated and insecure version, a score of 0 is justified. The other categories are evaluated in a similar manner.

Server configurations that cannot be captured by a score are accounted for by special rules to correct the grade calculated [62]. An example of a server rated B and the scores in each category can be observed in Figure 1.

Table 1. Main considerations evaluated in this study.

Security considerations[a]	Description
Use of secured connections (SSL[b]/TLS[c])	The use of any unsecured connections
SSL/TLS version	Evaluating the supported versions of SSL/TLS
Key exchange support	The cryptographic algorithm used to exchange the keys during the handshake for the following symmetric encryption
Cipher support	The cipher negotiated between client and server dictates what symmetric encryption is applied after the handshake and key exchange
Certificates	The security characteristics TLS offers rely on the server's certificate. Any trust issues here are critical
Vulnerabilities	Certain attacks are based on specific implementations or the absence of a patch on the server
HSTS[d]	Support HSTS can prevent downgrades to HTTP

[a]All but the first one (use of unsecure connections) are tested for by the tools presented in later sections.

[b]SSL: Secure Socket Layer.

[c]TSL: Transport Layer Security.

[d]HSTS: Hypertext Transfer Protocol Strict Transport Security.

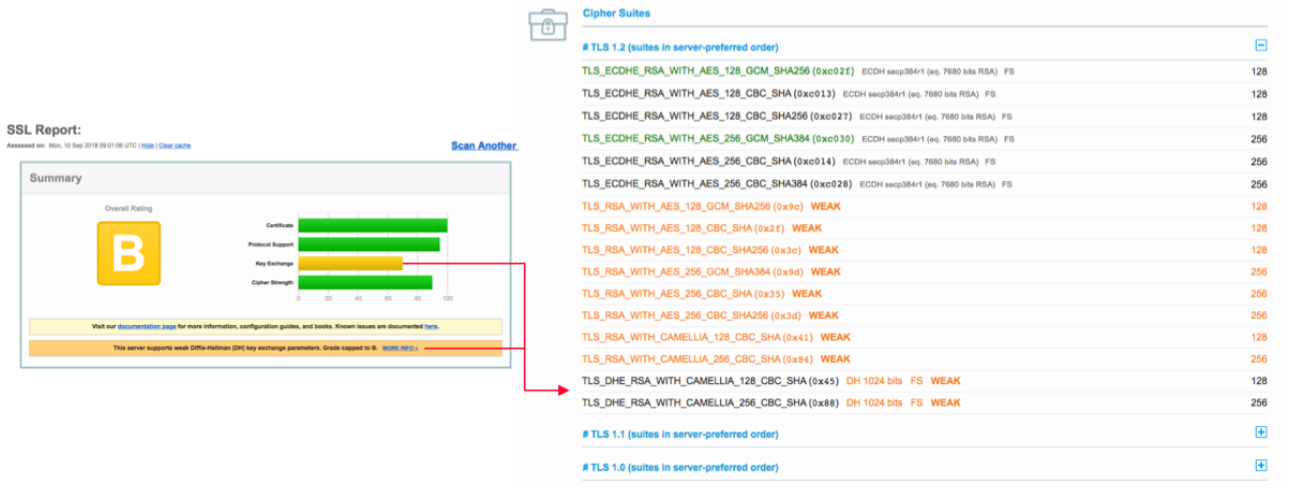
Table 2. Qualys SSL Labs scoring for protocol support.

Protocol	Score (%)
SSL[a] 2.0	0
SSL 3.0	80
TLS[b] 1.0	90
TLS 1.1	95
TLS 1.2	100

[a]SSL: Secure Socket Layer.

[b]TLS: Transport Layer Security.

Figure 1. Exemplary rating and scores of a domain in the test pool. The server was downgraded mainly for offering weak Diffie-Hellmann key exchange. The scores in the distinct categories can be observed to the left. On the right, the offered cipher suites, which include the key exchange algorithms, are listed and marked as weak points.



The overall score, based on a constantly updated set of rules, and the added safety of 2 sources for server security assessment are the reasons why both testssl (Version: 2.9.5dev) and the Qualys SSL Labs suite (Version: 1.32.6) are used. To determine the physical location of a server, a Web-based service was used [63,64]. To aggregate the results, a consistent logic was chosen. For negative observations (such as a rating below the A range containing ratings A+, A, and A− from Qualys SSL Labs), the worst observation in a given category was recorded. For positive observations, the inverse logic was applied: a category of an app was counted as supporting the positive characteristic (such

as HSTS and TLS 1.3) when at least one server supported the feature.

Limitations

The tests performed as part of this research focus on transport security and weaknesses in TLS configurations on the server side. The research did not include a full penetration test for every server that was found. Although availability is an important characteristic for a server, this attribute was not part of our tests as testing for denial-of-service (DOS) hardening would require staging test DOS attacks [6]. This would not be legally possible without cooperation of every app developer [65].

As this study focuses on the server side, correct certificate validation in the apps is not tested, neither is the traffic analyzed for leaked information of any kind.

Apps that did not function normally made a test of functions impossible and were removed during app selection. This includes apps that prevented traffic inspection by our test tool. It is possible that these apps used certificate pinning to ensure the servers identity.

The app tests were performed manually. Complete functional coverage cannot be guaranteed. Especially, functionality behind paywalls (in-app purchases or similar facilities) remained untested.

Categorization of servers as functional backends or others was aided by a Web-based service that keeps a list of known advertising and tracking servers but, in part, had to be performed manually [58]. The categorization in uncertain cases was vigilantly checked but might still contain errors.

The location of a server is hard to pin down. It is possible for a server to be inside the EU, although our test using a Web-based service did state a different location. Servers are often used to answer requests from different locales. A request from within the EU might be answered from a server in Ireland, whereas a request from the United States might be handled by a server located in the United States.

A server based outside the EU might fall under the EU-US privacy shield [56]. This allows organizations to store data outside the EU and still comply with applicable EU law [54]. As there is no straightforward way known to the authors to check if a server is part of privacy shield or complies with the GDPR in other ways, this differentiation is not made in this study.

A correct certificate validation depends on a dependable PKI. Research has shown that domain name system (DNS)–based domain validation by CAs is not always dependable and can be abused to have a CA issue certificates for arbitrary domains [66,67]. An attacker in possession of a valid certificate for the domain name requested can impersonate an authentic server even when the client applies correct certificate validation. As this is a CA's responsibility and an app provider has no control in this regard, this issue will not be addressed further in this study.

Results

App Selection

To select a sample set of mHealth apps, the top-downloaded lists of free apps from the Android and iOS app stores are considered as a starting point. The *medical* category was selected as it is most likely to contain mHealth apps. In previous research, similar client security problems in apps from 3 different countries were found [3]. Differences in behavior between apps from stores of different countries were not found. In this study, 30 apps from the German Google Play store and 30 apps from the iOS AppStore are considered (60 apps in total). The top-downloaded lists were generated from the app store analytics site AppAnnie on August 31, 2018 [68]. Top lists for a specific day are available after registration on the website. The *medical* category includes apps that fulfill a broad spectrum of functions. For this reason, apps from both app stores were categorized. All subcategories found in the *medical* categories are listed and defined in Multimedia Appendix 1. The 5 subcategories selected in this study are (1) fertility, pregnancy, and parenthood; (2) drug information, shopping, and compliance; (3) reference and learning; (4) consultation, communication, and interaction; and (5) health, fitness, and monitoring.

To improve variety, the 6 highest positioned apps from these categories and both lists were selected. If a selected app was untestable with test devices, the next most popular app of the same category was selected. This was the case for 9 apps. Of all 60 apps, 26 (26/60, 43%) were in at least one other top 500 list in France, the United Kingdom, or the United States, covering a portion of internationally relevant mHealth apps. The top positions in the stores, developer information, and subcategorization of the apps are visible in Multimedia Appendices 2 and 3.

Performing the Tests

To help parallelize the app testing, iOS and Android apps were tested separately. All apps for both platforms were downloaded from their respective app stores and installed on the devices (iOS 11.4.1 on an iPhone 7 and Android 6.0.1 on a Nexus 7). Before any test, the apps were stopped and restarted. As described previously, the devices are set up to use an HTTP proxy for all HTTP/S traffic. The BProxy tool was used to compile a list of relevant domains.

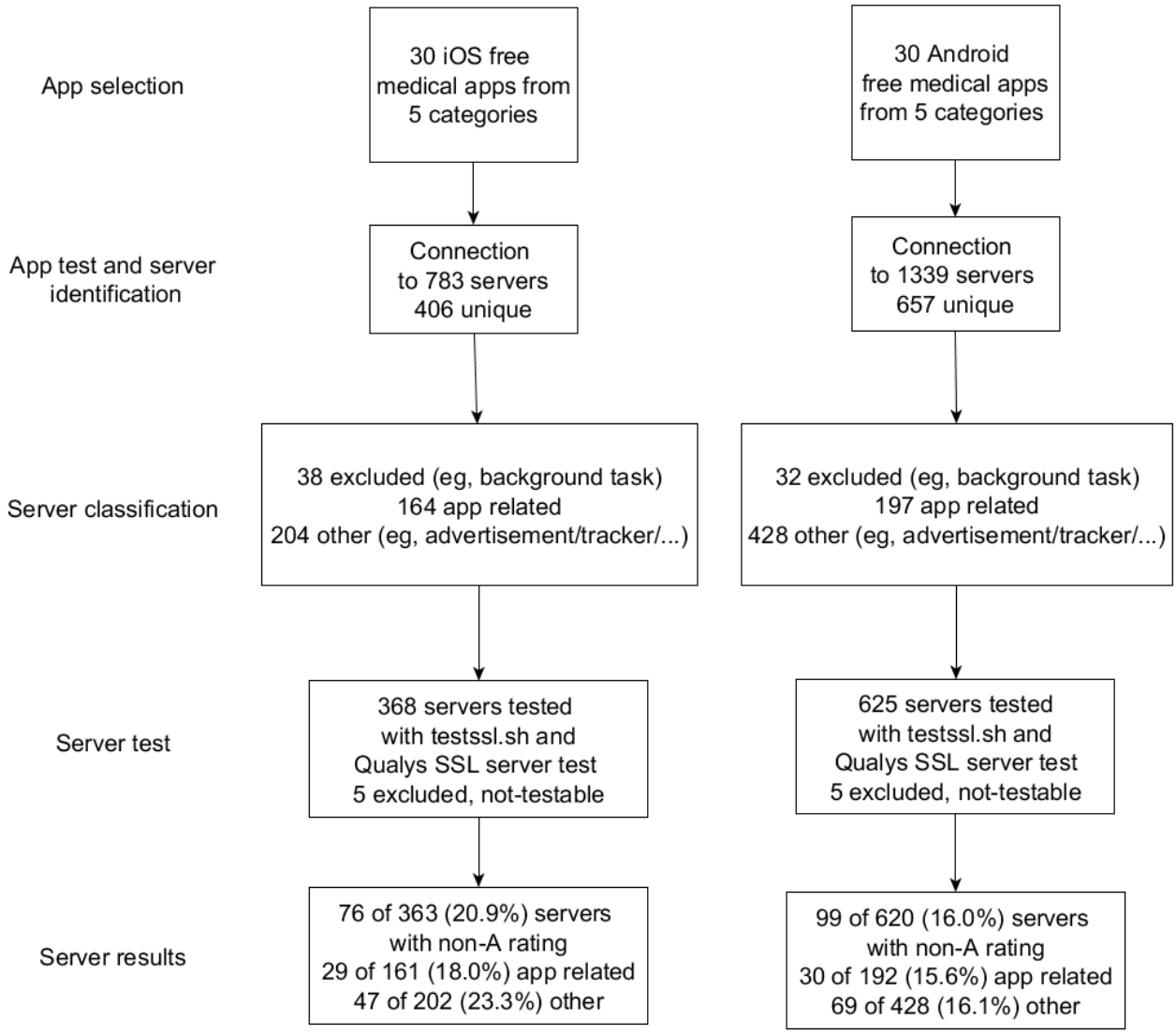
Many apps communicate with a plethora of endpoints. In addition, the Android or iOS OSs also communicate through HTTP/S connections for multiple purposes, including mail fetching, checking for updates, and sending analytics data. To filter the domains observed during the test of an app, multiple app runs are used to try to distinguish between app traffic and *background* traffic not related to the app. In addition, manual filtering was performed. Service calls from the OS (such as mail server communication) were disregarded. When necessary, an account for the app was created and activated for the tests.

A bash script was used to sequentially test each remaining domain using the testssl script [59]. The Qualys SSL Labs Server Test Web-based tool offers a bulk application programming interface (API) to test multiple domains. Both tools return

JavaScript object notation formatted files [59,69]. To reach the objective to generate informative and easy-to-understand results, it was decided to distinguish between backend, advertisement or tracker or analytics, external content, and other servers. The functional backend category includes servers that directly work to supply the app with functional content, perform operations with input from the app, and seem to be under the control of the app developer. All other servers were sorted into the second category of *other* servers, including all servers that serve advertisements, track user activity, and/or are part of an analytics service supplying the app provider with information about app usage. The categorization was performed by evaluating the domain name and analyzing HTTP/S traffic using the BProxy. Categorization as an *other* server was improved using the disconnect service where 311 of the 532 (58.5%) *other* servers were listed [58].

Figure 2 shows a graphical representation of the workflow described. To help evaluate and compile result summaries, the R programming language (version 3.4.2) was used [70]. Negative and positive observations were collected separately. During the compilation of the summaries, a negative or positive observation for a server during the tests was counted as the results for the entire *functional* or *other* backend category of the app. For some analyses, these sets have been combined to obtain results for medical apps in general.

Figure 2. Workflow for tests of mobile health apps. In the app selection phase, the 6 most popular apps from each of the 5 subcategories were selected. In the app test and server identification phase, the traffic between apps and servers was observed and unique servers recorded. The servers were categorized or disregarded as facilitating irrelevant background tasks (server classification phase). The relevant servers were used as the input for the testssl script and the Qualys SSL Labs suite (server test phase). Finally, the results tables were compiled (server results).



Summarized Results

The 60 apps tested communicated with 823 different servers. The distribution of the number of servers apps communicated with can be observed in Table 3. All apps communicated with servers beyond their functional backend. The median number of other servers the apps across both platforms communicated with is 18.5. The median number across Android apps is more than twice (24.5) the amount in comparison with iOS apps (11). In the most remarkable case, an Android app communicated with 82 *other* servers beyond its functional backend.

For easier evaluation of the overall results, the findings were divided into positive and negative results and summarized in Tables 4 and 5. These list the results for both functional as well as other backends. As the functional backends serve requests directly related to an app's functionality, these are the most interesting results.

Detailed results of the security tests can be found in Multimedia Appendices 2 and 3. They list the findings on a per app and per function basis. For further details, please refer to Multimedia Appendices 4 and 5. The tables in these appendices contain the number of connections per apps to servers exhibiting the security characteristics that are part of this study.

Table 3. Minimum, maximum, and median numbers of functional and other backends for iOS and Android apps.

Statistics	Android (functional)	iOS (functional)	Android (others)	iOS (others)	Overall (functional)	Overall (others)
Minimum number of servers	0	1	2	1	0	1
Maximum number of servers	33	21	82	39	33	82
Median number of servers	5	4	24.5	11	4.5	18.5

Table 4. A summarized table of negative results regarding backends of Android and iOS apps. Negative observations are counted for the functional or other category on a per-app basis when it was present in at least one of the apps' servers.

Security issues	Android (functional), n=30	iOS (functional), n=30	Android (others), n=30	iOS (others), n=30	Total (functional), n=60, n (%)	Total (others), n=60, n (%)
Qualys SSL Labs non-A rating	14	14	24	24	28 (47)	48 (80)
Server only offers TLS[a] version <1.2	5	3	0	1	8 (13)	1 (2)
Server without set cipher order	7	5	4	1	12 (20)	5 (8)
Certificate (chain) validation issues present	9	5	14	6	14 (23)	20 (33)
Downgrading vulnerabilities	5	4	8	7	9 (15)	15 (25)
Servers outside the EU[b]	24	21	30	30	45 (75)	60 (100)
Missing forward secrecy support	2	2	1	1	4 (7)	2 (3)
Unsecure connection/s observed	10	10	10	8	20 (33)	18 (30)

[a]TLS: Transport Layer Security.

[b]EU: European Union.

Table 5. A summarized table of positive results regarding backends of Android and iOS apps. One observation of a positive characteristic makes the functional or other category count for the app.

Positive findings	Android (functional), n=30	iOS (functional), n=30	Android (others), n=30	iOS (others), n=30	Total (functional), n=60, n (%)	Total (others), n=60, n (%)
TLS[a] 1.3 support observed	4	5	21	17	9 (15)	38 (63)
HSTS[b] support observed	12	15	28	25	27 (45)	53 (88)

[a]TLS: Transport Layer Security.

[b]HSTS: Hypertext Transfer Protocol Strict Transport Security.

Of the apps tested, 28 (28/60, 47%) used servers where at least one functional backend received a non-A rating from Qualys SSL Labs. In contrast, 48 (48/60, 80%) apps used advertisement, analytics, or external content providers (others) that received a rating below the A range.

Regarding the support of TLS 1.2, only 8 (8/60, 13%) apps used functional backend servers that did not offer TLS 1.2 (3 iOS apps and 5 Android apps). All but one app used only other servers that offered TLS 1.2.

Functional backend servers without a set cipher order were observed when using 12 (12/60, 20%) apps. Other servers without a cipher order were used by 5 (5/60, 8%) apps.

It was found that 14 apps (14/60, 23%) used functional backends that did not offer a valid certificate for the domain requested. This also includes certificates that fail proper validation for any reason (certificate chain issues and domain name mismatch). This issue was also found with other servers for 20 (20/60, 33%) apps.

It was discovered that 9 (9/60, 15%) apps worked with functional backend servers that were downgraded by Qualys SSL Labs because of their vulnerability to an exploit. More apps communicated with other backends that were downgraded (15/60, 25%) apps. Vulnerabilities that lead to a downgrade are related to the support of certain cipher suite or unpatched implementations on the server side [62]. Vulnerability to the POODLE attack was the most observed issue that led to a downgrade. The tables in the Multimedia Appendices 2 and 3 can be consulted for further information.

During the tests, it was found that 45 (45/60, 75%) apps appeared to use some functional backend servers outside the EU. The same is true for all 60 (60/60, 100%) apps regarding other backends.

Forward secrecy was supported by all but 4 (4/60, 7%) apps in at least one functional backend server. The support was observed in at least one other backend server for all but 2 (2/60, 3%) apps.

During the tests, we also recorded entirely unsecured connections by the apps to their servers. This was observed in 20 (20/60, 33%) apps during their communication with a functional backend server and in 18 (18/60, 30%) apps with *other* backend server.

During the evaluation of the test data, some apps showed especially severe issues and are responsible for many of the concerns listed in Table 4. Moreover, 1 app provider, for example, offered 4 apps in total (2 iOS and 2 Android apps). All these apps communicated with backends that did not support an up-to-date TLS version. In addition, it was found that the apps used a mix of unprotected HTTP and protected HTTPS connections to communicate with their backends, further undermining the security of the communication.

It was found that of the 291 functional backend servers, 15.1% received ratings below the A range by Qualys SSL Labs. Of the 532 nonbackend servers tested, 18.8% were rated below the A range. Qualys SSL Labs' SSL Pulse website lists popular security characteristics of servers. They summarize these and also show the percentage of servers receiving a rating below the A range. In the statistics from September 2018, of the 139,849 servers, 37.75% received non-A ratings. A conducted chi-square test (alpha=.05) shows the tested servers of mHealth apps to be significantly better rated than servers observed by SSL Pulse in general (P<.001 for both functional backends as well as others) [33].

Chi-square tests were also conducted between each of the subcategories regarding the number of apps that communicated with servers receiving a non-A rating. For these statistical tests, iOS and Android apps as well as functional and other servers were not differentiated. It was found that *reference and learning* apps received a significantly worse rating when tested against both *fertility, pregnancy, and parenthood* (alpha=.05; P=.02) and *drug information, shopping, and compliance* (alpha=.05; P=.01) apps. The other categories showed no significant differences between each other. Looking at differences between internationally available apps and apps only present in the German top lists, the international apps were found to perform significantly better (alpha=.05; P=.03). For this test, any app that was also listed in a top 500 list in the United States, the United Kingdom, or France was considered international.

Looking at regularly contacted domains that can be observed in Table 6, the services behind these top 10 most contacted servers receive data from many of the apps tested and will be able to reconstruct a comprehensive user profile.

Many advertising and analytics companies operate multiple second-level domains. Using the disconnect.me lists revealed, a high number of domains requested belonging to Google's and Facebook's services [58]. Google is very much present in the overview. Almost all (55/60, 92%) apps communicated with servers under a Google API domain, and 8 of the 10 most frequently observed second-level domains are attributable to the company.

Of the 60 apps tested, 17 offered a user-controllable opt-out option for certain advertisement or tracking services. In earlier research, these options were mentioned as desirable [3].

As for positive observations, TLS 1.3 support was observed in 9 (9/60, 15%) apps when communicating with their functional backend servers and 38 (38/60, 63%) with other backend servers. HSTS support was observed in at least one functional backend server for 27 (27/60, 45%) apps and at least once for other backend server in 45 (45/60, 75%) apps. The high percentages in the *others* category is partly a result of the way the results were counted. The number of other servers was regularly higher than the number of functional backend servers, and the positive observations were combined using the Boolean or-conjunction. To gain further insight into how many servers of an app's backend were exhibiting which security characteristic, please refer to Multimedia Appendices 4 and 5.

Table 6. Number of apps that communicated with a subdomain of the second-level domains listed.

Domain	Apps, n=60, n (%)
*.googleapis.com	55 (92)
*.google-analytics.com	46 (77)
*.google.com	38 (63)
*.googleapis.com	37 (62)
*.doubleclick.net	36 (60)
*.gstatic.com	33 (55)
*.crashlytics.com	29 (48)
*.google.de	23 (38)
*.googleadservices.com	23 (38)
*.fbcdn.net	11 (18)

Discussion

Principal Findings

Some backends display problematic characteristics. The most problematic cases exhibit invalid server certificates. In these cases, a client cannot distinguish between a real server certificate and a certificate from a third party attempting a MitM attack. The use of outdated TLS protocol versions can lead to integrity, authenticity, and confidentiality issues with data displayed or sent to the server. The implications vary from app to app. Affected apps did facilitate medical and drug information and interaction checks and patient to health care professional communication. The latter case can expose patient information and enable fabrication of answers from the health care professional to the patient.

In contrast to browsers, apps on mobile platforms have a disadvantage: they can (and do) hide transport security issues from their users. Although private data might be sent to a server without properly securing the connection, a developer can choose to ignore the issues. A browser, on the other hand, displays security warnings when a website can only be connected to using unsecure connections and will make interactions harder for the user [71]. Apple's App Transport Security effort is a step in the right direction but will still allow exceptions for app-specified URLs [72]. It could be the objective of further work to evaluate the possibility for a mobile OS to monitor all traffic from an app and warn the user about insecure connections.

Apps that are localized, available, and successful in more than the German app store suggest bigger organizations with more resources. These organizations can be expected to also devote more resources to the (security) maintenance of their servers. The observation that internationally popular apps performed significantly better validates this expectation.

As discussed in the study by Müthing et al [3], the use of advertisement and tracking services in medical apps can pose as a challenge. These services offer app developers insight into the usage of their app. But the data are also collected by these services for further monetization and data mining [73]. As medical (patient) data are protected under special jurisdiction and should be protected for ethical reasons, the use of any third-party services must be met with scrutiny. For many apps, the tests revealed the use of a great number of servers from various services. The high number of apps that use the same services for advertising or analytics can be problematic. These services can collect user information across multiple apps [73,74]. Another reason to be aware of the use of third-party services is the relatively large number of servers used for advertisements or tracking that received a non-A rating from Qualys SSL Labs (48% of all apps).

Looking at the results in regard to server locations, all (60/60, 100%) apps used *other* servers outside the EU. When looking at the source data, it can be observed that these servers are often related to tracking, analytics, and advertising.

The data also reveal severe issues not directly visible when only considering the server setup: apps were observed using entirely unsecured connections or a mixture of secured and unsecured connections in communication with the same server. Although the backend server might be set up to use state-of-the-art TLS and certificates, this will always undermine potential security and put user data at risk.

The wide support of forward secrecy (functional: 15% and others: 63%) can be seen positively, as it prevents unauthorized decryption of sensitive data in the future and entails a performance burden for attacks on the connection. The high number (functional: 45% and others: 88%) in servers supporting HSTS can help to mitigate against protocol downgrade and cookie hijacking attacks.

During an earlier phase of the preparation of this study—and before the tests discussed so far—40 mHealth apps were tested. The 20 most popular apps from the *medical* category from the German iOS and Android app stores were tested with a very similar methodology as was described in this study. A description of the 40 apps, their categories, and the summarized results can be found in Multimedia Appendix 6. Detailed results per app tested for Android and iOS can be found in Multimedia Appendices 7 and 8, respectively. Some apps were tested in both test runs. Although the earlier results showed a lower number of functional backends receiving a Qualys SSL Labs grade below the A range (28% in early 2018 vs 47% in

September 2018), there are also positive developments observable. Vulnerability to the ROBOT attack was observable in a nonfunctional backend in early 2018 but was fixed in the later tests. The relative rise of backends receiving a rating below the A range might be caused by a difference in classification but can also be attributed to the changes in the Qualys SSL Labs rating calculation algorithm [62]. This dynamic is characteristic of the fast-paced nature of the security ecosystem of TLS deployments. Although servers are maintained by some providers, the discovery of new vulnerabilities and the reinvention of old ones leading to new threats make constant vigilance a necessity for server administrators and (mHealth) app providers. The apparent security deterioration of servers by their grades might indicate a slower pace of app providers in keeping their TLS deployments secure.

Common Security Concerns and Recommendations

Common security concerns are listed and summarized below. Prevention, mitigation, or alleviation recommendations are given:

1. Apps were found using a server without valid certificates both for their functional backend as well as for other purposes. This is problematic as this implies missing or erroneous certificate validation in the apps. When a client does not expect valid credentials from the server, any attacker can present equally invalid certificates and impersonate the server. This enables MitM attacks. It is strongly recommended to use a trusted CA and have a valid certificate issued for the domains that are to be secured.
2. A number of apps communicate with servers through unsecured channels. To find and prevent apps and users from using unsecured connections, a server could be configured to use HSTS. In addition, client apps should be inspected and any unsecured URL schemes should be removed.
3. Insecure server configurations indicated by a low Qualys SSL Labs' rating were found in multiple apps' server backends. This includes a missing set cipher order and the support of vulnerable cipher suites. These vulnerabilities can be exploited to undermine the security supplied by the use of TLS. Insecure server configurations can change over time. Any time a new exploit is discovered and/or is widely exploited, a server operator should update his server's configuration. To keep a server configured as securely as possible, the basic security concerns should be understood [5], server-side security patches installed, and the domains should be tested using a service similar to Qualys SSL Labs' server test [60].
4. Most apps use multiple advertising or analytics servers. Not only does this add to the data and processor time used by apps, for medical apps, but can also be especially problematic as the analytics data can undermine a user's or patient's privacy. A patient looking for pregnancy-related content, for example, might be pregnant. In addition, most of these services appear to be located outside the EU, and most apps used at least one such server that received a rating below the A range by Qualys SSL Labs. Although the use of advertising and analytics services is common in mobile apps, mHealth app developers should thoroughly reconsider the usage of such third-party services and frameworks [73-75]. A possible trade-off could be to offer an opt-out function to the user [3].

Conclusions

Modern mHealth apps from popular subcategories were tested in depth and their behavior was analyzed. Although servers of mHealth apps performed significantly better than servers in general, most apps communicate with a considerable number of different servers by different operators. It was observed that these servers and connections to them are regularly not as secure as connections to the apps' functional backends. The services behind some of these servers (advertisement and app analytics) should also be seen critically in regard to user or patient data protection. Almost half of all apps communicate with functional backends that do not offer a secure TLS setup (non-A Qualys SSL Labs rating).

The most severe problems observed in a small number of apps can expose patient communication with health care professionals, be exploited to display false or harmful information to the user, or used to send data to an app facilitating further damage to the device. These problems include communication through entirely unsecured connections, a mix of secured and unsecured connections, invalid certificates used by servers, certificate chain validation issues, missing support for modern TLS versions, and unpatched vulnerabilities.

As made evident by the comparison of the results discussed in this study with results of previous studies, the security of servers is heavily dependent on the existence and propagation of vulnerabilities. A provider of mHealth apps dealing with (potentially) sensitive information should have an even higher interest in keeping their servers up-to-date and vulnerabilities patched.

The recommendations proposed in the previous section can be used by app developers to improve their transport security setup and prevent putting patients and/or users at risk.

Acknowledgments

The authors would like to thank Obioma Pelka for the help during the proofreading process.

Authors' Contributions

CMF, RB, and JM designed the setup of the tests and chose the tools for testing. CMF and RB performed the tests. RB evaluated the results and compiled the results tables. JM drafted the paper. All authors provided corrections and approved the final version.

Conflicts of Interest

None declared.

Multimedia Appendix 1

Definitions of the subcategories of apps found under the medical category.

[PDF File (Adobe PDF File), 63KB-Multimedia Appendix 1]

Multimedia Appendix 2

Table containing detailed results and information about the Android apps tested.

[XLSX File (Microsoft Excel File), 14KB-Multimedia Appendix 2]

Multimedia Appendix 3

Table containing detailed results and information about the iOS apps tested.

[XLSX File (Microsoft Excel File), 14KB-Multimedia Appendix 3]

Multimedia Appendix 4

Table containing the numbers of connections to servers per app and security characteristic for the Android apps tested.

[XLSX File (Microsoft Excel File), 23KB-Multimedia Appendix 4]

Multimedia Appendix 5

Table containing the numbers of connections to servers per app and security characteristic for the iOS apps tested.

[XLSX File (Microsoft Excel File), 23KB-Multimedia Appendix 5]

Multimedia Appendix 6

App descriptions, apps by categories and summarized results from server security results of popular mHealth apps form early 2018 for both iOS and Android.

[PDF File (Adobe PDF File), 480KB-Multimedia Appendix 6]

Multimedia Appendix 7

Detailed results from server security results of popular mobile health apps from early 2018 for Android.

[XLSX File (Microsoft Excel File), 8KB-Multimedia Appendix 7]

Multimedia Appendix 8

Detailed results from server security results of popular mobile health apps from early 2018 for iOS.

[XLSX File (Microsoft Excel File), 8KB-Multimedia Appendix 8]

References

1. Istepanian R, Laxminarayan S, Pattichis CS, editors. M-Health: Emerging Mobile Health Systems. New York: Springer; 2006.
2. Akhawe D, Felt AP. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In: Proceedings of the 22nd USENIX conference on Security. 2013 Aug 14 Presented at: SEC'13; August 14-16, 2013; Washington, DC, United States p. 257-272 URL: https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_akhawe.pdf
3. Müthing J, Jäschke T, Friedrich C. Client-focused security assessment of mHealth apps and recommended practices to prevent or mitigate transport security issues. JMIR Mhealth Uhealth 2017 Oct 18;5(10):e147 [FREE Full text] [doi: 10.2196/mhealth.7791] [Medline: 29046271]
4. Fielding R, Irvine UC, Gettys J, Frystyk H, Masinter L, Leach P, et al. IETF Tools.: IETF; 1999. Hypertext Transfer Protocol--HTTP/1.1 URL: https://tools.ietf.org/html/rfc2616 [accessed 2017-09-18] [WebCite Cache ID 6tZREH58V]

5. Ristic I. Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications. London: Feisty Duck; 2018.
6. Perrin C. TechRepublic. 2008. The CIA Triad URL: https://www.techrepublic.com/blog/it-security/the-cia-triad/ [accessed 2018-05-11] [WebCite Cache ID 6zJlBN5yg]
7. Plummer D. IETF Tools. 1982 Nov. Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48bit Ethernet Address for Transmission on Ethernet Hardware URL: https://tools.ietf.org/html/rfc826 [accessed 2018-11-30] [WebCite Cache ID 74IRcZoIS]
8. Ramachandran V, Nandi S. Detecting ARP Spoofing: An Active Technique. In: Proceedings of the International Conference on Information Systems Security.: Springer; 2005 Presented at: ICISS'05; December 19-21, 2005; Kolkata, India p. 239-250. [doi: 10.1007/11593980_18]
9. Vanhoef M, Piessens F. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.: ACM Press; 2017 Presented at: CCS '17; October 30-November 03, 2017; Dallas, Texas, USA p. 1313-1328. [doi: 10.1145/3133956.3134027]
10. Sepehrdad P, Susil P, Vaudenay S, Vuagnoux M. Smashing WEP in a Passive Attack. Berlin, Germany: Springer; 2013 Presented at: International Workshop on Fast Software Encryption; March 11, 2013; Washington, DC, USA p. 155-178. [doi: 10.1007/978-3-662-43933-3_9]
11. Cassola A, Robertson W, Kirda E, Noubir G. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. 2013 Presented at: NDSS Symposium 2013; February 24, 2013; San Diego, CA, USA.
12. Dierks T, Allen C. The TLS Protocol Version 1.0. 1999. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc2246.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKmXnHw7]
13. Rescorla E. HTTP Over TLS. 2000. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc2818.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKaCoz7G]
14. Huang LS, Adhikarla S, Boneh D, Jackson C. An experimental study of TLS forward secrecy deployments. IEEE Internet Comput 2014 Nov;18(6):43-51. [doi: 10.1109/MIC.2014.86]
15. Drake JJ. Stagefright: An Android Exploitation Case Study. In: 10th USENIX Workshop on Offensive Technologies (WOOT). 2016 Presented at: WOOT '16; August 8-9, 2016; Austin, TX, USA.
16. Meyer C, Schwenk J. International Association for Cryptologic Research.: IACR Cryptology ePrint Archive; 2013. Lessons Learned From Previous SSL/TLS Attacks A Brief Chronology Of Attacks And Weaknesses URL: https://eprint.iacr.org/2013/049.pdf [accessed 2018-05-11] [WebCite Cache ID 6zL5btUjI]
17. Stanek M. Comenius University.: arXiv; 2017 Aug 24. Secure by default - the case of TLS URL: https://arxiv.org/pdf/1708.07569.pdf [accessed 2018-11-30] [WebCite Cache ID 74JyxErWt]
18. Möller B, Duong T, Kotowicz K. Google. 2014. This POODLE Bites: Exploiting The SSL 3.0 Fallback URL: https://www.openssl.org/~bodo/ssl-poodle.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKcsqBIb]
19. Böck H, Somorovsky J, Young C. Return Of Bleichenbacher's Oracle Threat (ROBOT). In: Proceedings of the 27th USENIX Conference on Security Symposium.: USENIX Association; 2018 Presented at: SEC'18; August 15-17, 2018; Baltimore, MD, USA.
20. Synopsys, Inc. 2014. The Heartbleed Bug URL: http://heartbleed.com [accessed 2018-01-22] [WebCite Cache ID 6wemwGqYP]
21. Housley R, Ford W, Polk W, Solo D. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. 1999. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc2459.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKcZKZNa]
22. Let's Encrypt. San Francisco, CA, USA: Internet Security Research Group (ISRG); 2018. URL: https://letsencrypt.org/ [accessed 2018-05-11] [WebCite Cache ID 6zKUHrmJn]
23. Open Web Application Security Project (OWASP). 2018. URL: https://www.owasp.org/index.php/Main_Page [accessed 2018-01-22] [WebCite Cache ID 6wen4anph]
24. Open Web Application Security Project. 2017. Top 10-2017 Top 10 URL: https://www.owasp.org/index.php/Top_10_2017-Top_10 [accessed 2018-05-11] [WebCite Cache ID 6zKcCrm8S]
25. Open Web Application Security Project. 2016. Mobile Top 10 2016-Top 10 URL: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10 [accessed 2017-02-09] [WebCite Cache ID 6o9JIoR8V]
26. Dehling T, Gao F, Schneider S, Sunyaev A. Exploring the far side of mobile health: information security and privacy of mobile health apps on iOS and android. JMIR Mhealth Uhealth 2015 Jan 19;3(1):e8 [FREE Full text] [doi: 10.2196/mhealth.3672] [Medline: 25599627]
27. Hsu J, Liu D, Yu YM, Zhao HT, Chen ZR, Li J, et al. The top Chinese mobile health apps: a systematic investigation. J Med Internet Res 2016 Dec 29;18(8):e222 [FREE Full text] [doi: 10.2196/jmir.5955] [Medline: 27573724]
28. He D, Naveed M, Gunter CA, Nahrstedt K. Security concerns in Android mHealth apps. AMIA Annu Symp Proc 2014;2014:645-654 [FREE Full text] [Medline: 25954370]
29. Wei X, Wolf M. A survey on HTTPS implementation by Android apps: issues and countermeasures. Prog Adv Comput Intell Eng 2017 Jul;13(2):101-117 [FREE Full text] [doi: 10.1016/j.aci.2016.10.001]
30. Chu G, Apthorpe N, Feamster N. IEEE Internet Things J. 2018. Security and privacy analyses of internet of things children's toys URL: https://arxiv.org/pdf/1805.02751.pdf [accessed 2018-12-06] [WebCite Cache ID 74Si2UYZc]

31. Oppliger R. SSL and TLS: Theory and Practice, Second Edition. Norwood, MA: Artech House; 2016.
32. Jiang S, Smith S, Minami K. Securing Web servers against insider attack. In: Proceedings of the 17th Annual Computer Security Applications Conference. 2001 Dec 10 Presented at: ACSAC '01; December 10-14, 2001; New Orleans, Louisiana, USA p. 265-276. [doi: 10.1109/ACSAC.2001.991542]
33. Qualys SSL Labs. 2018. SSL Pulse URL: https://www.ssllabs.com/ssl-pulse/ [accessed 2018-01-22] [WebCite Cache ID 6wepClCQS]
34. Zeit. 2018. This app does not help me fall asleep [in German] URL: https://www.zeit.de/wissen/gesundheit/2018-09/psychische-gesundheit-apps-moodpath-schlafstoerungen-erfahrung [accessed 2018-11-30] [WebCite Cache ID 74Jy0oH3g]
35. HealthOn. HealthOn App Code of Ethics for Health Apps [in German] URL: https://www.healthon.de/ehrenkodex [accessed 2018-10-29] [WebCite Cache ID 73XM1QpoP]
36. Maass M, Wichmann P, Pridöhl H, Herrmann D. PrivacyScore: Improving PrivacySecurity via Crowd-Sourced Benchmarks of Websites. In: Schweighofer E, Leitold H, Mitrakas A, Rannenberg K, editors. Privacy Technologies and Policy. Cham: Springer International Publishing; 2017:178-191.
37. Maurer U. Modelling a Public-Key Infrastructure. In: Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security. UK: Springer-Verlag; 1996 Presented at: ESORICS '96; September 25-27, 1996; Rome, Italy p. 325-350 URL: https://dl.acm.org/citation.cfm?id=699185 [doi: 10.1007/3-540-61770-1_45]
38. Housley R, Ford W, Polk W, Solo D. Internet X.509 Public Key Infrastructure Certificate and CRL Profile.: IETF; 1999 Jan. URL: https://www.rfc-editor.org/rfc/rfc2459.txt [accessed 2018-11-30] [WebCite Cache ID 74JyCVbS7]
39. Turner S, Polk T. Prohibiting Secure Sockets Layer (SSL) Version 2.0. 2011 Mar. URL: https://www.rfc-editor.org/pdfrfc/rfc6176.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKnAWzH3]
40. Barnes R, Thomson M, Pironti A, Langley A. Deprecating Secure Sockets Layer Version 3.0. 2015. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc7568.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKeVo47b]
41. Rizzo J, Duong T. Practical Padding Oracle Attacks. In: Proceedings of the 4th USENIX conference on Offensive technologies. 2010 Presented at: WOOT'10; August 9, 2010; Berkeley, CA, USA p. 1-8 URL: http://dl.acm.org/citation.cfm?id=1925004.1925008
42. Ristic I. Qualys SSL Labs. 2013. Is BEAST Still a Threat? URL: https://blog.qualys.com/ssllabs/2013/09/10/is-beast-still-a-threat [accessed 2018-01-22] [WebCite Cache ID 6wepLFf5q]
43. Sheffer Y, Holz R, Saint-Andre P. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). 2015. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc7457.txt.pdf [accessed 2018-05-11] [WebCite Cache ID 6zKac2MmJ]
44. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3.: IETF; 2018 Aug. URL: https://www.rfc-editor.org/rfc/rfc8446.txt [accessed 2018-11-30] [WebCite Cache ID 74JsSwFHr]
45. Hodges J, Jackson C, Barth A. HTTP Strict Transport Security (HSTS).: IETF; 2012 Nov. URL: https://www.rfc-editor.org/rfc/rfc6797.txt [accessed 2018-11-30] [WebCite Cache ID 74JscC3j5]
46. Barker E, Chen L, Roginsky A, Smid M. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. In: NIST special publication 800-56A Revision 2. Gaithersburg, MD, USA: NIST; May 2013.
47. Seroussi G. Elliptic curve cryptography. : IEEE; 1999 Jun 27 Presented at: Information Theory and Networking Workshop; 27 June-1 July 1999; Metsovo, Greece p. 41. [doi: 10.1109/ITNW.1999.814351]
48. Rivest RL, Shamir A, Adleman LM. Cryptographic communications system and method. Washington, DC, USA: U.S. Patent and Trademark Office; 1983.
49. Barker E, Roginsky A. NIST Special Publication 800-131A Revision 1. Gaithersburg, MD, USA: NIST; 2015. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths URL: https://www.nist.gov/publications/transitions-recommendation-transitioning-use-cryptographic-algorithms-and-key-lengths-0[WebCite Cache ID 75cQFIPp5]
50. Dierks T, Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2.: IETF; 2008 Aug. URL: https://www.rfc-editor.org/rfc/rfc5246.txt [accessed 2018-11-30] [WebCite Cache ID 74JtLXOrt]
51. Maurer U. Modelling a public-key infrastructure. In: Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security. Berlin, Heidelberg: Springer; 1996 Presented at: ESORICS 1996; September 25-27, 1996; Rome, Italy p. 325-350. [doi: 10.1007/3-540-61770-1_45]
52. Böck H, Somorovsky J, Young C. Robot Attack. 2017. The ROBOT Attack - Return Of Bleichenbacher's Oracle Threat URL: https://robotattack.org [accessed 2018-10-03] [WebCite Cache ID 72toEp1MS]
53. Sheffer Y, Holz R, Saint-Andre P. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS).: IETF; 2015 Feb. URL: https://www.rfc-editor.org/rfc/rfc7457.txt [accessed 2018-11-30] [WebCite Cache ID 74JuA4NmD]
54. Official Journal of the European Union. Brussels, Belgium: European Commission; 2016. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) URL: http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=DE, [accessed 2018-01-22] [WebCite Cache ID 6wepXOTzy]

55. European Commission.: European Commission; 2016. Privacy Code of Conduct on mobile health apps URL: https://ec.europa.eu/digital-single-market/en/privacy-code-conduct-mobile-health-apps [accessed 2018-01-22] [WebCite Cache ID 6wepa89cD]
56. Official Journal of the European Union. Brussels, Belgium: European Commission; 2016 Apr 27. COMMISSION IMPLEMENTING DECISION (EU) 2016/1250 pursuant to Directive 95/46/EC of the European Parliament and of the Council on the adequacy of the protection provided by the EU-U.S. Privacy Shield URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016D1250&from=EN [accessed 2018-11-30] [WebCite Cache ID 74JuwyjwB]
57. Müthing J. Github. BProxy URL: https://github.com/j4nnis/bproxy [accessed 2018-01-22] [WebCite Cache ID 6weprpw1R]
58. Disconnect. 2018. URL: https://disconnect.me [accessed 2018-09-28] [WebCite Cache ID 72lxKBy6i]
59. Wetter D. Test SSL. 2018. Testing TLS/SSL encryption URL: https://testssl.sh [accessed 2018-01-22] [WebCite Cache ID 6wepvH2Jd]
60. Qualys SSL Labs. 2018. SSL Server Test URL: https://www.ssllabs.com/ssltest/ [accessed 2018-01-22] [WebCite Cache ID 6wepy1keE]
61. Qualys SSL Labs. 2018. ssllabs-scan URL: https://github.com/ssllabs/ssllabs-scan [accessed 2018-01-22] [WebCite Cache ID 6weq0ynxY]
62. Qualys SSL Labs. 2017 May 08. Qualys SSL Server Rating Guide URL: https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide [accessed 2018-01-22] [WebCite Cache ID 6weq5XAMB]
63. IP-API. 2018. IP Geolocation API URL: http://ip-api.com [accessed 2018-11-30] [WebCite Cache ID 74JwHDsIj]
64. Shavitt Y, Zilberman N. A geolocation databases study. IEEE J Select Areas Commun 2011 Dec;29(10):2044-2056. [doi: 10.1109/JSAC.2011.111214]
65. Rasch M. Security Current.: security current; 2013 Nov 26. Legal Issues in Penetration Testing URL: https://securitycurrent.com/legal-issues-in-penetration-testing/ [accessed 2018-09-29] [WebCite Cache ID 72nS3BAlU]
66. Son S, Shmatikov V. The Hitchhiker's Guide to DNS Cache Poisoning. Berlin, Heidelberg: Springer; 2010 Presented at: International Conference on Security and Privacy in Communication Systems; September 7-9, 2010; Singapore, Singapore p. 466-483. [doi: 10.1007/978-3-642-16161-2_27]
67. Herzberg A, Shulman H. Security of Patched DNS. Heidelberg, Berlin: Springer; 2012 Presented at: European Symposium on Research in Computer Security (ESORICS); September 10-12, 2012; Pisa, Italy p. 271-288. [doi: 10.1007/978-3-642-33167-1_16]
68. App Annie. 2018. URL: https://www.appannie.com [accessed 2018-11-30] [WebCite Cache ID 74JxBE6BG]
69. Qualys SSL Labs. 2018. SSL Labs APIs URL: https://www.ssllabs.com/projects/ssllabs-apis/index.html [accessed 2018-10-29] [WebCite Cache ID 73XMSSddH]
70. R Core Team. R Foundation for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing; 2018. The R Project for Statistical Computing URL: https://www.r-project.org/ [accessed 2018-12-01] [WebCite Cache ID 74JxPQXSy]
71. Reis C, Barth A, Pizano C. Browser security: lessons from Google Chrome. Commun ACM 2009 Aug 01;52(8):45-49. [doi: 10.1145/1536616.1536634]
72. Apple. Cupertino, CA, USA: Apple; 2018. Cocoa Keys URL: https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33, [accessed 2018-09-10] [WebCite Cache ID 72nSFR3OB]
73. Razaghpanah A, Nithyanand R, Vallina-Rodriguez N, Sundaresan S, Allman M, Kreibich C, et al. Apps, Trackers, Privacy, and Regulators-A Global Study of the Mobile Tracking Ecosystem. 2018 Presented at: Network and Distributed Systems Security (NDSS) Symposium; February 18-21, 2018; San Diego, CA, USA. [doi: 10.14722/ndss.2018.23353]
74. Binns R, Lyngs U, van Kleek M, Zhao J, Libert T, Shadbolt N. Third Party Tracking in the Mobile Ecosystem. In: Proceedings of the 10th ACM Conference on Web Science. New York, NY, USA: ACM; 2018 Presented at: WebSci '18; May 27-30, 2018; Amsterdam, NL p. 23-31. [doi: 10.1145/3201064.3201089]
75. Thurm S, Kane YI. The Wall Street Journal.: The Wall Street Journal; 2010 Dec 17. Your Apps Are Watching You: A WSJ Investigation finds that iPhone and Android apps are breaching the privacy of smartphone users URL: https://www.wsj.com/articles/SB10001424052748704694004576020083703574602 [accessed 2018-09-29] [WebCite Cache ID 72nLyoV6U]

Abbreviations

API: application programming interface
ARP: address resolution protocol
BEAST: Browser Exploit Against Secure Socket Layer/Transport Layer Security
CA: certificate authority
CRIME: Compression Ratio Info-leak Made Easy
CSV: comma-separated value
DNS: domain name system

DOS: denial-of-service
EC: elliptic curve
ECDHE: Elliptic Curve Diffie Hellman Ephemeral
EU: European Union
GDPR: General Data Protection Regulation
HSTS: Hypertext Transfer Protocol Strict Transport Security
HTTPS: HTTP Secure
IETF: Internet Engineering Task Force
IP: internet protocol
KRACK: Key Reinstallation Attack
mHealth: mobile health
MitM: man-in-the-middle
OS: operating system
OWASP: Open Web Application Security Project
PKI: Public Key Infrastructure
POODLE: Padding Oracle On Downgraded Legacy Encryption
ROBOT: Return Of Bleichenbacher's Oracle Threat
RSA: Rivest-Shamir-Adleman
SSL: Secure Socket Layer
TCP: transmission control protocol
TLS: Transport Layer Security
WEP: wired equivalent privacy
WPA: Wi-Fi Protected Access

Edited by M Stanley, E Perakslis; submitted 22.01.18; peer-reviewed by T Dehling, M Abdelhamid, S Albakri, A Ferreira; comments to author 08.05.18; revised version received 03.10.18; accepted 01.11.18; published 23.01.19

Please cite as:
Müthing J, Brüngel R, Friedrich CM
Server-Focused Security Assessment of Mobile Health Apps for Popular Mobile Platforms
J Med Internet Res 2019;21(1):e9818
URL: https://www.jmir.org/2019/1/e9818/
doi: 10.2196/jmir.9818
PMID: 30672738