TU WIEN Informatics

Developing a Vulnerability Assessment Concept for eHealth iOS Applications

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Vanessa Hohenegger, BSc
Matrikelnummer 01425703

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Thomas Grechenig

Wien, 11. Oktober 2021

Vanessa Hohenegger	Thomas Grechenig

TU WIEN Informatics

Developing a Vulnerability Assessment Concept for eHealth iOS Applications

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Medical Informatics

by

Vanessa Hohenegger, BSc
Registration Number 01425703

to the Faculty of Informatics

at the TU Wien

Advisor: Thomas Grechenig

Vienna, 11th October, 2021

Vanessa Hohenegger	Thomas Grechenig

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Vanessa Hohenegger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Oktober 2021

Vanessa Hohenegger

Acknowledgements

At this point, I wish to express my sincere gratitude to Thomas Grechenig and Florian Fankhauser, who made this possible.

I would like to thank Raphael Kiefmann and Thomas Stipsits for their generous support.

I would also like to thank my parents, friends and colleagues for their patience and motivation.

Kurzfassung

Heutzutage sind Smartphones ein fester Bestandteil unseres Lebens und sowohl für die Kommunikation als auch zum Sammeln von Informationen nicht mehr wegzudenken. Mobile Geräte werden immer mehr in alltägliche Abläufe integriert. Dieser Trend macht auch vor Gesundheits-Anwendungen nicht Halt. Solche Applikationen, kurz Apps, speichern viele sensible Daten, die geschützt werden müssen. Electronic health-Apps (eHealth Apps) sind vielen Bedrohungen ausgesetzt. Oft führen fehlerhafte Konfigurationen und Software-Implementierungen zu Sicherheitsproblemen. Aufgrund der hohen Relevanz von Gesundheitsdaten können Sicherheitsprobleme besonders schwerwiegende Folgen haben.

Mit Hilfe von Sicherheitstests, wie beispielsweise einer Schwachstellenanalyse, können Sicherheitsprobleme frühzeitig gefunden werden. Es existieren Standards und Leitfäden zur Durchführung solcher Tests, die jedoch nicht spezifische eHealth-Anforderungen berücksichtigen, z. B. die Minimierung der verarbeiteten Benutzerdaten, minimale Zugriffsberechtigungen und sichere Standardeinstellungen. Zudem legen viele Leitfäden ihren Fokus auf Android und kaum auf iOS Betriebssysteme oder beschreiben allgemeine Testprozesse. Das Ziel dieser Diplomarbeit ist es ein Schwachstellenanalyse-Konzept für iOS Applikationen auszuarbeiten, welches die speziellen Anforderungen von Gesundheits Apps berücksichtigt. Das Konzept soll den Zugang zu Sicherheitstests erleichtern und in verschiedenen Bereichen, beispielsweise bei Pentests, als Unterstützung dienen. Es kann aber auch für die Formulierung von Anforderungen oder von Entwicklern verwendet werden, um sichere Apps zu erschaffen.

Um das Konzept dieser Arbeit zu testen, wurden vier Apps aus dem Apple App Store ausgewählt und anhand dessen auf Sicherheitsprobleme untersucht. Mit Hilfe des Konzepts wurde in jeder App mindestens eine Schwachstelle gefunden.

Keywords eHealth, Schwachstellenanalyse, iOS Sicherheitstests, mobile Anwendungen security, iOS applikationen

Abstract

Nowadays, smartphones are an integral part of our lives and are indispensable for communication as well as for gathering information. Mobile devices are becoming more integrated into everyday processes and are even used in health-related matters. Electronic health applications, abbreviated as eHealth apps, are constantly increasing in popularity. Mobile health apps process a lot of sensitive data that needs to be protected. Misconfigurations and faultily designed software often lead to app vulnerabilities. Due to the high relevance of health data, vulnerabilities can have severe consequences.

Security testing, such as vulnerability assessments, can help detect vulnerabilities in apps early on. There are standards and guidelines for conducting security tests. However, these do not consider specific eHealth requirements, for example minimizing the processed user data, minimum permissions and secure-by-default settings. Furthermore, guidelines focus primarily on Android and hardly on iOS operating systems or only describe general reviewing processes. This thesis aims to design a vulnerability assessment concept for iOS apps that considers specific eHealth requirements. The concept is intended to facilitate access to security tests and can be utilized in various areas, such as pentesting. It can also be used by requirement engineers or by developers to create safe apps.

Some selected electronic health (eHealth) applications from the Apple App Store were assessed using the elaborated concept in order to validate its effectiveness. At least one vulnerability was found in each app using the concept.

Keywords eHealth, vulnerability assessment, iOS security testing, mobile application security, iOS applications

Contents

CHAPTER 1

Introduction

This chapter serves as an introduction to the thesis by introducing the problem description, expected results, methodical approach, related work, and structure of this thesis.

1.1 Problem Description

Mobile devices enable communication and other activities regardless of location and time. Applications (Apps) can enhance existing functionalities on mobile devices and are defined as „a computer program or piece of software designed for a particular purpose that you can download onto a mobile phone or other mobile device“ by the Cambridge Dictionary [102]. They can be downloaded from online platforms for mobile apps such as the Apple „App Store“.[1] Health apps are constantly increasing in popularity. For example, the App Store offers different health apps, including apps that support tracking a menstrual cycle or diabetes therapy. Moreover, as of January 1, 2021, statutory insured persons in Germany can use the German electronic health records (ePA) optionally for free, which was specified by the Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH (gematik) [63]. The ePA, in German „elektronische Patientenakte“, an electronic health record (EHR) is provided by the health insurance companies in form of a mobile app. It aims to simplify processes in the German health care system by making medical information easier to access. For example, doctors and therapists can upload treatment or laboratory reports and diagnoses to this EHR and share it between medical specialists.

Mobile health apps such as the ePA process a lot of sensitive data. In a literature review Fernández-Alemán et al. [60] found out that EHRs are more prone to privacy issues than paper-based health records. Furthermore, eHealth apps face additional security threats, as reports show:

[1] App Store, `https://www.apple.com/at/ios/app-store`, last accessed 2021/10/11

For example, Gieselmann, Tremmel, and Eikenberg [66] tested the free health app „Ada“[2] and found severe privacy deficiencies in 2019. Users can enter their symptoms and receive a list of possible diseases. The data provided by the user was partly shared with third parties, posing a significant privacy threat.

Another example of a German eHealth app with notable security flaws is „Vivy“[3]. This app serves as a mobile health assistant to organize a user's data regarding health by providing a digital vaccination certificate, medication plan, and personal health check. Tschirsich and Schröder [122] documented several problems such as erroneous implementation of cryptographic operations that would allow the leakage of confidential, private data.

The German Federal Office for Information Security (BSI) [72] warns that a compromised smartphone can reveal a lot of information about the user. Hence, user data, in particular sensitive user data, and the systems storing the information require special protection. The increasing number of eHealth app users and reports of security deficiencies in eHealth apps emphasize the importance of security testing in order to detect vulnerabilities early. Early detection can protect the systems to avoid denial of service and user data leakage, to name a few. Security testing such as vulnerability assessments detects security deficiencies in apps. Lists such as the Open Web Application Security Project (OWASP) [98] top 10 mobile risks support the security testing of mobile applications by providing an overview of the most common mobile application vulnerabilities. There are several guidelines and standards for general vulnerability assessment approaches that can be utilized for conducting a vulnerability assessment. Neither of these considers the high protection needs of the processed data and specific eHealth application as well as General Data Protection Regulation (GDPR) requirements, as summarized by the BSI [72]. Requirements include, for example, minimizing the processed user data, minimum permissions, secure-by-default settings, and handling insecure smartphones.

1.2 Expected Results

This thesis aims to design a vulnerability assessment concept for iOS eHealth apps to increase the security of sensitive user data. In order to achieve this goal, an analysis form is developed. This form includes checklists that aim to guide the assessment process.

The concept is based on security, eHealth, the iOS platform, and application fundamentals, as well as existing standards and guidelines. Moreover, the most prevalent mobile vulnerability types are considered to understand where security deficiencies can occur in iOS applications.

In addition, some selected eHealth applications from the Apple App Store are assessed using the elaborated concept in order to validate its effectiveness.

This thesis aims to answer the following research questions:

[2] Ada, `https://ada.com`, last accessed 2021/10/11
[3] Vivy, `https://www.vivy.com`, last accessed 2021/10/11

1. What are common vulnerabilities of iOS eHealth applications?
2. How can iOS eHealth application vulnerability testing be conducted?
3. Are vulnerability assessments an effective measure to detect vulnerabilities in iOS eHealth apps?

1.3 Methodological Approach

First, a literature research is carried out to obtain knowledge of security, mobile devices, applications, eHealth and iOS fundamentals. Knowledge of these fundamentals is needed in order to understand where and why vulnerabilities in eHealth applications occur. Moreover, existing standards and documents are analyzed to find common vulnerabilities in iOS applications.

Furthermore, existing security testing standards, guidelines, strategies, and approaches are analyzed to find out how to conduct iOS eHealth application vulnerability testing. Other components like application access, testing methods, and types are part of this analysis.

The information gathered by the literature research will be used to develop a vulnerability assessment guideline for iOS eHealth applications. The existing security testing standards will be used to elaborate the process of the vulnerability assessment concept. The application access, testing methods, and types are used to determine the focus of the vulnerability assessment.

Finally, some eHealth and ePA applications from the Apple App Store are selected according to specific criteria, such as media mention or popularity in the App Store. These apps are assessed following the elaborated guideline in order to determine the effectiveness of the proposed concept and to find out if vulnerability assessments are an effective measure to detect vulnerabilities in iOS eHealth apps.

1.4 Related Work

In the course of initial research, several approaches, guidelines, and testing processes regarding mobile security testing were found, but neither consider specific eHealth application requirements, for example, as summarized by the BSI [72]. The following literature will be taken as a basis for developing the vulnerability assessment concept within the scope of this thesis.

OWASP [97] published a Mobile-Security-Testing-Guide for Android and iOS, including a security assessments checklist mainly focuses on Android applications. This guide takes the most prevalent mobile vulnerabilities from the OWASP Mobile Top 10 [98] into account and describes how to they can be tested.

Velu [125] provides a step-by-step iOS app penetration testing guide. Chell et al. [53] describe iOS mobile application security and attack techniques from a hacker's view.

Relan [108] examines iOS vulnerabilities, security features, and tools and performs an iOS app black-box test as an example using the knowledge described in the book.

Khera, Kumar, and Nidhi Garg [77] discuss the importance of vulnerability assessments and penetration testing and describe fundamentals and processes of these techniques.

Garg and Baliyan [62] present a comparative analysis between Android and iOS on a wide range of security aspects, including common software vulnerabilities and vulnerability trends. This work highlights recent vulnerabilities in iOS that are necessary for the concept of this thesis.

Sahi, Lai, and Li [112] summarize state of the art approaches in security and privacy in the eHealth cloud. Moreover, the authors take papers into account that concentrate on eHealth data security requirements and current research trends in the areas of privacy and security.

There are standards and guidelines for security assessments and testing such as the OSSTMM [68], the NIST 800-115 [119], PCI-DSS [55], and BSI [73] that propose a general approach for performing an assessment. Due to their high level of abstraction, these methodologies cannot be directly utilized for performing a vulnerability assessment and, therefore, need to be adapted to meet the specific iOS and eHealth applications requirements.

None of the outlined works can be directly utilized for vulnerability assessments of iOS eHealth applications. Therefore, based on this literature, an iOS vulnerability assessment guideline for iOS eHealth applications will be developed.

The concept of this thesis considers GDPR and security requirements for eHealth applications provided by the BSI [72] and eHealth Network [86].

1.5 Thesis Structure

This thesis is divided into two parts: acquiring the necessary theoretical fundamentals for developing a vulnerability assessment concept and applying the concept.

The first part focuses on laying the foundations required to understand outlined concepts in the following chapters. Moreover, the proposed iOS eHealth vulnerability assessment concept builds on these fundamentals. Therefore, Chapter 2 introduces the security term, eHealth and iOS fundamentals. In addition, the chapters 3 and 4 go into further detail regarding iOS fundamentals. Chapter 5 defines testing, testing methods, security testing techniques and discusses existing standards and guidelines. On these fundamentals, a concept of an iOS vulnerability assessment for eHealth applications is proposed in Chapter 6.

In the second part, the elaborated concept is used for practical vulnerability assessments of chosen applications. The results are presented in Chapter 7. Finally, the conclusion and outlook are outlined in Chapter 8.

CHAPTER 2

Fundamentals

For this thesis, fundamental concepts are relevant and will be introduced, including what vulnerabilities are and how they can be classified, which is defined in Section 2.1. The fundamentals in Section 2.2 provide definitions of mobile devices and apps including their threats and attacks that are needed to understand where security deficiencies can occur. The eHealth fundamentals in Section 2.3 define what eHealth is including their threats and introduce the EHR.

2.1 Security Fundamentals

Security is an essential part of eHealth applications in order to protect sensitive data. This section defines what attacks are, the term vulnerability and how they can be classified.

2.1.1 CIA Triad

Confidentiality, integrity, and availability represent three important security goals. Together they form the confidentiality, integrity and availability (CIA) triad as shown in Figure 2.1. The data, which is placed in the center of the CIA triad, is considered secure when all three requirements are fulfilled, according to Singh, Rishiwal, and Kumar [118]. If persons retrieved information that was not meant to be public or a system is not usable anymore after an attack, it means that at least one of these goals has been violated.

Confidentiality guarantees, according to Fernández-Alemán et al. [60], that only permitted persons can access information. Additionally, Dehling and Sunyaev [56] point out that information needs to be protected during storage and transmission. Therefore, users should only have access to information they need, and unnecessary access rights should be avoided and revoked.

Integrity ensures, according to Umrao, Kaur, and Gupta [123], that data is not modified or deleted either by accident or by deliberately harmful intentions. According to Farooq

et al., [59], this goal includes the protection of information during storage, modification, and transmission.

Availability [56] means that data is accessible and systems are operational when needed. This goal includes scalability and proper backup mechanisms. Scalability means that a system's performance can handle changing performance needs.

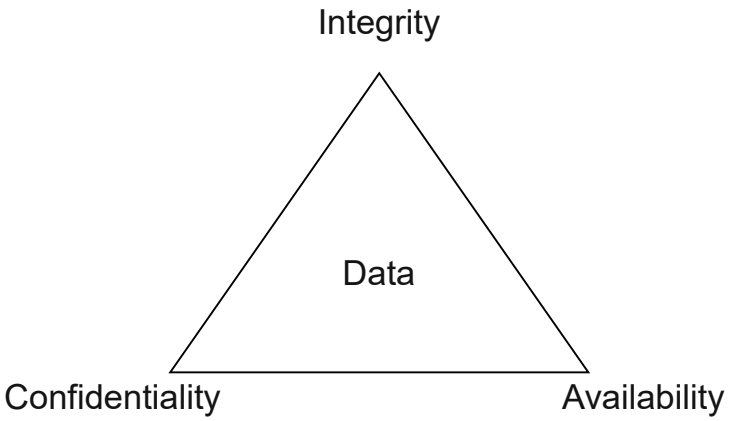

Figure 2.1: The CIA-Triad. Rebuilt after [59]

2.1.2 Defining the Vulnerability, Threat and Attack Terms

Formal definitions of the vulnerability, threat, and attack terms are needed for the following chapters and are introduced in this section.

Vulnerability

The term vulnerability is defined by Black et al. [50] as „one or more weaknesses that can be accidentally triggered or intentionally exploited and result in a violation of desired system properties." A weakness is defined by Black et al. [50] as „an undesired characteristic of a system's requirements, design, or implementation." Exploited means „to use something in a way that helps you" according to the Cambridge Dictionary [104].

A vulnerability can be a hardware, software bug or misconfiguration, according to Shah and Mehtre [114]. It can, warn Khera, Kumar, and Nidhi Garg [77], enable an attacker to exploit a system, for example, to get system privileges that are required to execute specific operations.

Vulnerabilities [77] arise because of faulty software and system misconfigurations. Dehling and Sunyaev [56] point out that there are differences in knowledge about secure programming and privacy protection among people who implement and distribute applications. Thus, bugs or misconfigurations can emerge and cause a vulnerability in the system.

Threat

The term threat is defined by the Cambridge Dictionary [106] as „the possibility that something unwanted will happen, or a person or thing that is likely to cause something unwanted to happen“. Another definition is provided by the OWASP [90]: „A threat is a potential or actual undesirable event that may be malicious (such as DoS attack) or incidental (failure of a Storage Device).“

Attack

The term „attack“ is defined by the Cambridge Dictionary [103] as „a violent act intended to hurt or damage someone or something“. Another definition is provided by the OWASP [89]: „Attacks are the techniques that attackers use to exploit the vulnerabilities in applications.“

2.1.3 Vulnerability Classifiers and Quantifiers

The Common Weakness Enumeration (CWE), Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (CVSS) are introduced by Ogata et al. [88] as frequently used vulnerability classifiers and quantifiers. They are used to describe, identify, and measure the severity of vulnerabilities. These classifiers aim to make exploits comparable to other exploits and common vulnerabilities.

CWE

The CWE [88] is a classification system for software weaknesses published by MITRE [1]. This system provides a common language for software weakness classes. Programming languages can produce language-specific interpretations of the same error. The CWE provides terminology to describe the same error across different languages and mitigation strategies.

CVE

The CVE [88] is a dictionary from MITRE that offers a naming scheme for vulnerabilities in software. A found vulnerability can be sent to a CVE Numbering Authority that returns an exclusive identifier for it. CVEs are assigned to the National Vulnerability Database (NVD) [2] for description and scoring. The NVD is a repository in the United States that manages standards-based vulnerability information.

CVSS

The CVSS [88] is a scoring model for vulnerabilities published by the Forum of Incident Response and Security Teams (FIRST) [3]. This system aims to maintain reusable and

[1] MITRE, `https://www.mitre.org`, last accessed 2021/10/11
[2] NVD, `https://nvd.nist.gov`, last accessed 2021/10/11
[3] FIRST, `https://www.first.org`, last accessed 2021/10/11

detailed measurements and make underlying characteristics of vulnerabilities transparent to users. The characteristics of vulnerabilities are classified using numerical scores. The CVSS is used when consistent and proper vulnerability exploits and impact scores are needed, for example, in industries, organizations and governments. The CVSS is mostly used for vulnerability severity calculation and to prioritize vulnerability recovery processes. The NVD utilizes CVSS for vulnerability scores.

2.2 Mobile Devices and Application Fundamentals

This section introduces mobile devices and application fundamentals including their threats.

2.2.1 Defining Mobile Devices and Applications

Mobile devices are defined as „any piece of electronic equipment such as a cell phone or small computer that you can use in different places“ by the Cambridge Dictionary [105]. Alwahedi et al. [1] point out that mobile devices consist of the hardware, the software, and the operating system. They enable communication regardless of the location and time leading to new security challenges and threats, warn Chell et al. [53].

Apps can enhance existing functionalities on mobile devices and are defined as „a computer program or piece of software designed for a particular purpose that you can download onto a mobile phone or other mobile device“ by the Cambridge Dictionary [102]. They can be downloaded from online platforms for mobile apps such as the Apple „App Store“ [4].

2.2.2 Threats of Mobile Devices

Mobile applications [53] often require an internet connection. Since mobile devices can be easily carried along, the device might connect to an insecure network, for example, in hotels, airports, or other public places, which can expose sensitive information. Mobile devices store a lot of sensitive data, as warned by Dehling and Sunyaev [56]. Devices like smartphones are commonly carried all day long by the owner and can collect a lot of data in this time. Mobile applications can utilize different input sources like Near Field Communication (NFC), Bluetooth, cameras, and microphones. The data saved on mobile devices can be used, for example, for identity theft. Kalyango and Maiga [75] define identity theft as a crime where unauthorized individuals retrieve and use personal data from somebody else. Medical identity theft is a crime where individuals use somebody else's data to receive unjustified medical services or products. This crime can entail a financial loss for the victim and difficulties in getting hired.

[4] App Store, `https://www.apple.com/at/ios/app-store`, last accessed 2021/10/11

2.2.3 Attacks on Mobile Devices

Mobile security issues [1] can be categorized based on to the CIA triad introduced in Section 2.1.1. Attacks can be classified by more than one of these categories.

Attacks on confidentiality utilize eavesdropping, Man-in-the-Middle (MitM) attacks and malicious software like spyware. Eavesdropping, also called disclosure attack, tries to retrieve confidential information by observing the network. In MitM attacks, the adversary intercepts the communication between sender and receiver. These attacks can be performed by establishing a malicious access point. If a user connects to this access point, the data sent through the network can be read by the attacker.

Attacks on integrity can target the encryption system and hence, the confidentiality as well as the integrity of the data. These attacks can be conducted by placing malware inside third-party applications.

Attacks on availability often utilize malware, botnets, and Denial of Service (DoS) attacks. Botnets are malicious networks consisting of machines that are under the control of the attacker. The attacker is called botmaster and the controlled systems bot or zombie. Mobile devices can be unknowingly turned into bots, leading to slow system performances and sending and receiving data unwittingly.

2.2.4 OWASP Mobile Top 10 – Listing the Most Prevalent Vulnerability Types

The nonprofit foundation OWASP [5] aims to improve the security of software. This foundation offers a platform to open-source projects to improve software security. One of the OWASP projects is a top 10 mobile risks list that introduces common mobile vulnerabilities and describes the following points for each vulnerability:

- Threat Agents
- Attack Vectors
- Security Weakness
- Technical Impacts
- Business Impacts
- Prevention

Some vulnerabilities show example attack scenarios.

[5] OWASP, `https://www.owasp.org`, last accessed 2021/10/11

OWASPs [99] top 10 mobile risks in 2016 were:

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

The exploitation of the vulnerabilities can lead to, among others, privacy violations and information theft.

This section provides an extract of the OWASP top 10 mobile risks 2016 and serve as an example. Therefore, a short overview of the first three risks in the top 10 list, namely improper platform usage, insecure data storage, and insecure authentication, is provided.

Improper Platform Usage

OWASP [99] describes that improper platform usage indicates a misused platform feature or a failing platform security control. This category includes, among others, platform permission issues and misuse of the iOS keychain that is introduced in Section 4.1.4. An exposed Application Programming Interface (API) is considered as an attack vector example. Exposed web services or API calls in the mobile application can lead to a vulnerability exploit. All platforms, such as iOS and Android, have guidelines regarding application development. It can lead to vulnerabilities if a developer does not follow those guidelines or interprets them wrong.

Insecure Data Storage

As pointed out in Section 2.2.2, mobile devices store a lot of sensitive data that can be used for identity theft by attackers. The insecure data storage category [99] implies that an attacker has physical access to the mobile device or that the attacker's malware is executed on the device. Free software allows browsing all third-party application directories that may contain stored Personally Identifiable Information (PII) or other sensitive data. Wrong assumptions about the access to a device file system and poor encryption libraries of a device can lead to this vulnerability. Exploitation can enable the extraction of sensitive data.

Insecure Communication

An architecture of a mobile application usually consists of a client and server application that communicate with each other. This communication requires an internet connection. As pointed out in Section 2.2.2, attackers can intercept the sent data. Compromised Wi-Fi, carrier or network devices [99] such as routers and malware on the device can lead to this vulnerability. For example, the usage of free public Wi-Fi access points increases the risk of this vulnerability. Mobile application often use Secure Sockets Layer (SSL)/Transport Layer Security (TLS) only for authentication and payment processes. An application might have implemented transport security incorrectly because best practices or platform guidelines were not considered.

2.3 eHealth Fundamentals

eHealth is defined by Chauhan and Jaiswal [52] as a collection of software applications that provides tools, processes, and communication systems to simplify health practices. Thus, eHealth applications manage health-related information and improve processes between patient and health professionals. According to Sabnis and Charles [110], eHealth offers electronic transmission of health resources that utilizes communication technologies for information sharing. Examples for eHealth services are EHR, telemedicine, and mobile health (mHealth). Sampat and Prabhakar [113] highlight that technological developments and the widespread use of mobile technologies created mHealth, a subset of eHealth. Based on mobile devices, mHealth enables the use of healthcare applications regardless of location and time. Thus, mHealth applications are software applications for devices like smartphones.

This section introduces fundamentals associated with eHealth concepts.

2.3.1 The GDPR – Personal and Health Data

The GDPR 2016/679 provides guidelines for collecting and processing data of natural persons living in the European Union (EU), as outlined by Lopes and Oliveira [83]. The protection of personal data is a fundamental right. The GDPR explains that any information that belongs to a person and can be used for their identification is considered as PII. Identifiers can be names, online identities like social media accounts, identification numbers, and location details. Data associated with social, cultural, physical, physiological, or economic factors are also considered identifiable. The GDPR prohibits, according to Eickmeier [57], personal data processing where racial and ethnic origin, political opinions, religious or ideological convictions, or trade union membership can be derived. The data processing regarding genetic, biometric, health information, and sexual orientation is also prohibited. There are exceptions where the guideline does not apply, for example, if the individual has explicitly consented to process the personal data.

The GDPR considers health data as sensitive personal data, according to Sahama, Simpson, and Lane [111]. Hence, there are specific conditions concerning access and processing of the data by third parties. The GDPR expects the usage of adequate organizational and technical measures to maintain a sufficient security level. The measures include the encryption and pseudonymization of data and guaranteed availability, confidentiality, and integrity of data and services.

Information on Data Collection

An eHealth application needs to provide at least the following information to its users, according to the European Commission [54]:

- information about the app publisher including contact details
- the purpose and legal justification of the personal user data collection and processing
- how long the personal user data will be kept
- who else has access to the data, such as third-parties
- if the personal user data will be transmitted to a destination outside the EU
- information about the users' right to
  - get a copy of their data and further fundamental rights regarding the data protection
  - file a complaint with a Data Protection Authority
  - withdraw consent anytime
- if automated decision-making is utilized, information about its processing logic and consequences

2.3.2 Threats of eHealth Application Concepts

Besides the mobile device threats outlined in Section 2.2.2, some attacks specifically target eHealth systems.

Security, especially the CIA, is crucial for eHealth systems because they process a lot of sensitive data, as pointed out by Bai, Dai, and Li [47]. There are confidentiality attacks such as activity and location tracking as well as eavesdropping. For example, eHealth websites may use the Hypertext Transfer Protocol (HTTP) instead of the Hypertext Transfer Protocol Secure (HTTPS) to communicate with a server. In this case, data can be intercepted by attackers because HTTP transmits data as plain-text. Attacks focusing on confidentiality can result in patient privacy breaches and data modification, leading to a loss of data integrity and subsequently to a potential wrong diagnosis or treatment.

Integrity attacks can influence emergency alarms, such as creating false alarm notifications that might delay processing real emergencies. Furthermore, the inaccessibility of eHealth resources is a threat to the availability of a medical system.

Fernández-Alemán et al. [60] emphasize that availability of health data is crucial for effective healthcare practice. eHealth systems have to stay usable in case of natural disasters, system failures, and security attacks. DoS [47] is a threat to availability. This threat can make eHealth services such as emergency services inaccessible.

2.3.3 Electronic Health Records

An EHR stores patients health data electronically, according to Bai, Dai, and Li [47]. Patient health data includes among others demographics, medical history, medication, and radiology images.

This section describes the advantages of EHRs and introduce the ePA, a German EHR.

Advantages

Advantages of EHRs include easy access and the elimination of poor handwriting as outlined by Menachemi and Collum [85]. The usage of paper-based health records, remark Fernández-Alemán et al. [60], leads to an extensive paper trail. EHRs avoid these paper trails and make health records less likely to be misplaced, enable mobility, as well as allow for easier interoperability. Other advantages are the improved quality of care, cost reduction and the encouragement of evidence-based medicine. Therefore, EHRs need to satisfy particular requirements like data completeness, high availability, and proper failure handling.

German Electronic Health Records

The gematik [64] points out that the German legislature decided that the EHR is an essential part of the telematics infrastructure. The gematik had to determine the specifications, approval procedures, and field test concepts for the ePA, in German „elektronische Patientenakte “, according to Section 291a SGB V (Fifth Book of the Social Security Code) [124].

The gematik [64] defines the technical requirements including: the EHR backend, a frontend for policyholders, and primary systems for healthcare service providers for the ePA. Companies have to implement products according to the requirements and apply for approval from the gematik. The approval requires that applicants prove that their component or service for the EHR meets the requirements for functionality, interoperability, and security as specified by the gematik.

The ePA aims to simplify processes in the German health care system by making medical information easier to access for patients and their healthcare providers. For example, doctors and therapists can upload treatment or laboratory reports and diagnoses to the

EHR and this information can be digitally shared between specialists. The ePA also offers the functionalities of an electronic medication plan and emergency data set. The ePA should be primarily used for documents, such as doctor's letters submitted by doctors and records submitted by patients such as pain diaries or self-measured blood pressure values as outlined by the gematik [65]. Patients decide which medical documents and how long these are stored in the EHR, as well as which health professionals can access their documents for what duration. People insured by the statutory health system have the right to optionally use the ePA for free from 1st January 2021 onwards, according to the gematik [63]. German statutory health insurances provide the EHR in the form of a mobile application (Insurant Frontend (FdV)).

2.3.4 Concerns and Risks

Hoerbst et al. [69] describe in their study that fears of the patients and health workers in Germany and Austria and the lack of information about the preferences have an impact on the acceptance of eHealth concepts. Concerns like the utilization of the EHR by criminals or systematic searches of health data of politicians and famous people to create a scandal were expressed in interviews. The concerns mainly addressed data privacy issues. Patients are aware of the risks of EHR like theft of sensitive medical data. Furthermore, the authors assume that reports of data breaches in the media increased awareness.

The acceptance of an EHR [69] by patients and health workers is essential for its success. The physicians' documentation time increased with implementation of an EHR, which was one of the concerns of health professionals, according to Baumann, Baker, and Elshaug [49]. EHRs involve more privacy issues than paper-based health records, as outlined by Fernández-Alemán et al. [60]. Moreover, EHRs face new security threats. An EHR is interoperable and can be accessed at different locations, such as hospitals and pharmacies. Due to these different access points, the probability that security problems will occur is higher. EHRs [60] are challenged with upholding data privacy because administrative staff could retrieve patient data without consent.

2.3.5 Requirements of eHealth Applications

eHealth applications have to consider at least the three protection goals confidentiality, integrity, and availability as introduced in Section 2.1.1. These applications are exposed to threats as listed in Section 2.3.2. Therefore, specific requirements for secure eHealth applications were introduced, for example by the eHealth Network [86] and the BSI [72]. This section summarizes requirements from the BSI and the eHealth Network.

eHealth applications should only request permissions and process user data that they need. If data or permissions are no longer required, they should be deleted or revoked.

As outlined in Section 2.2.4, platforms provide coding guidelines for proper usage that should be considered by eHealth applications. Apps should use built-in security func-

tionalities, including dedicated storage for app user data provided by the operating system.

Network communication should be TLS encrypted. Configuration of the TLS has to consider current best practice recommendations such as from the BSI [71]. The app should use security functionalities of the used operating system, secure cryptographic libraries or frameworks for communication with the backend.

Users of eHealth apps might not be tech-savvy. Therefore, app settings affecting user data should be disabled by default. For example, push notifications can expose sensitive information. In addition, the app should educate users about security best practice recommendations that they can follow, such as using a passcode on the device. eHealth apps should provide an authentication method for users to access their sensitive data.

Apps might be installed on older operating systems with known vulnerabilities, missing built-in security features, or jailbroken devices. Insecure systems should be detected and users warned in order to protect their data.

CHAPTER 3

iOS Architecture and Platform Security

This chapter introduces the architecture and platform security of the iOS operating system that are particularly important for the following chapters.

3.1 iOS Architecture

Apple [1] developed the iOS mobile operating system, initially called iPhone OS, for their mobile devices. It is similar to the Mac OS X operating system, which is the operating system for Apples laptops and desktop computers, according to Garg and Baliyan [62]. Moreover, iOS and the devices that run this operating system are both manufactured and maintained by the same company.

Figure 3.1 shows the layers in which iOS architecture can be divided. The first layer [62] is the hardware that includes the physical chips. The second layer, the core OS, communicates directly with the hardware. This layer [87] includes a kernel, the file, and network management, a safety system as well as various device drivers. The following core services layer contains the object-oriented APIs and basic features like the protection of data, SQLite database, options, and contact management. In addition, this layer includes the interaction with hardware features like network, GPS, compass, accelerometer, and gyroscope. The following layer handles graphics, audio, and video capabilities. The offered frameworks of the media layer allow access to photos and videos that are stored on the device. The last layer is the cocoa touch layer which offers the frameworks required for building iOS applications. This layer is the user interface of iOS that enables basic functionalities like touch, multitasking, and notifications.

[1] Apple, `https://www.apple.com`, last accessed 2021/10/11

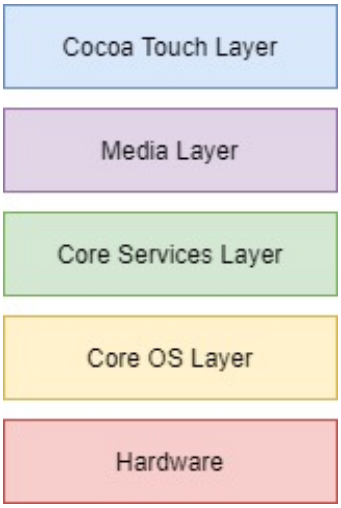

Figure 3.1: The iOS architecture layers. Rebuilt after [62]

3.2 iOS Platform Security

Apple [20] considers security for the designs of their device hardware, software, and services to protect the user's data. As a result, iOS has different security technologies implemented to maintain the integrity of the operating system, applications, and data. This section introduces the system, application, encryption and data protection, as well as the network security of the iOS platform security that has to be considered in iOS vulnerability assessments.

3.2.1 System Security

The iOS system security [41] focuses on protecting the system resources including the boot process and software updates.

One part of system security is the secure boot that aims to prevent attacks at boot time, according to Apple [5]. iOS protects integrity by providing a chain-of-trust as outlined by Kellner et al. [76]. This chain-of-trust is already used in the early stages of the boot process and is later used to verify iOS applications. Every step of this chain guarantees that the following step functions as intended. Figure 3.2 illustrates this chain-of-trust. The first component is the Boot Read-Only Memory (ROM). It is an immutable part of the processor and hence, it cannot be modified or updated. This component contains the Apple Root Certificate, including the public key, which is used to verify the signature of the iBoot bootloader. In case the signature verification fails, the boot process stops and no code execution is possible. The device switches into recovery mode and requires a reinstallation of the firmware. Applications that have a valid signature are allowed to be executed.

The secure boot ensures that only code signed by Apple can be installed.

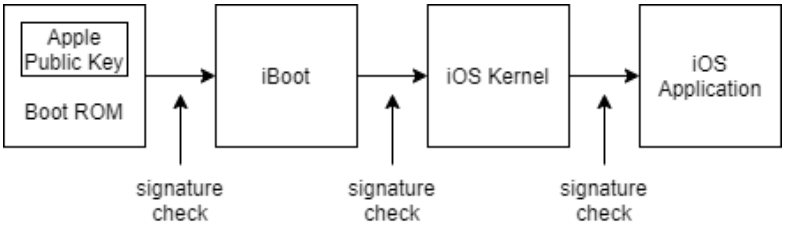

Figure 3.2: The chain-of-trust of iOS devices. Rebuilt after [76]

Another part of the system security are software updates [38] that are released regularly to remediate security issues. Apple provides secure processes to avoid downgrade attacks and discontinues signing older operating system versions with known vulnerabilities.

3.2.2 Application Security

Each third-party application runs in their own „sandbox“ [39], according to Kellner et al. [76]. The sandbox enables separation from other applications and the operating system. Each installed application gets its own directory named with a random char-sequence, in which it has all permissions.

Applications are not able to access files from other applications as pointed out by Wang et al. [126]. However, iOS enables Inter-Process Communication (IPC) with URL scheme handlers between applications. The URL scheme handlers allow applications to register a URL type and qualify other applications to launch and send messages to the registered application. To use this functionality, applications need to call a URL scheme such as „http“, „mailto“, „tel“ and „sms“. These URL schemes are linked by default to built-in applications in iOS. For example, if a „http“ URL is clicked, it launches the built-in web browser „Mobile Safari“ and opens the requested URL there. Apple [12] allows defining custom URL schemes for applications in order to refer to resources inside the application.

An application [76] cannot access files and directories outside its sandbox directory. However, specific directories like the photo directory can be accessed using special services when the user has been granted access. The user can revoke the access permissions at any time.

Permissions of applications [76] are limited on iOS. Most of the iOS devices run as the non-privileged user „mobile“. Privileges granted to the „mobile“ user cannot be escalated using the iOS API. Furthermore, the sandbox does not allow the execution of system calls, for example, „kill“, which terminates a process. In addition, the operating system partition does not allow write access because it is mounted as read-only. This security mechanism should protect system resources and files from third-party applications.

Address Space Layout Randomisation (ASLR) is a system protection against memory corruption attacks, according to Evtyushkin, Ponomarev, and Abu-Ghazaleh [58]. Built-in applications utilize ASLR to guarantee that the address space layout is randomized

when it is launched. Additionally, iOS Apple devices rely on the Execute Never (XN) functionality that does not allow the execution of code in the memory pages.

The executable code of third-party applications has to be signed with a certificate issued by Apple [26]. The signing guarantees that the source of the application is known and approved. Moreover, the signing enhances the chain-of-trust.

3.2.3 Encryption and Data Protection

Apple [14] provides encryption features to protect user data, for example, if the device is lost or stolen. Kernels of the operating system provide protection and security mechanisms like sandboxing and „Data Vault“ [11] which is a mechanism protecting the confidentiality of data.

„Data Protection“ is technology used by iOS to protect data saved in flash storage on the device as outlined by Apple [10] [13]. Built-in applications, for example, messages, mail, and health data values utilize it by default. „Data Protection“ can be utilized to protect files of the application. It is activated when the device owner uses a passcode on the device. iOS encrypts and decrypts the files automatically. There are the following levels [9] of how data can be protected, sorted from least to most restrictive:

- **NSFileProtectionNone:** No protection, the files are always accessible.
- **NSFileProtectionCompleteUntilFirstUserAuthentication:** The files are inaccessible until the first time the device owner unlocks the device. Afterward, the files stay accessible until the device is shut down or rebooted. This level is the default setting.
- **NSFileProtectionCompleteUnlessOpen:** The files can be accessed when the device is unlocked and can still be accessed when the device is locked again. New files can be created and accessed in both locked and unlocked device states.
- **NSFileProtectionComplete:** The files are only accessible if the device is unlocked.

NSFileProtectionNone and NSFileProtectionCompleteUntilFirstUserAuthentication allow access to the files even when the device is locked. When the device is connected to a computer, the user needs to allow access to the device. This process requires an unlocked device. When the user allows it, the computer is trusted. The default setting NSFileProtectionCompleteUntilFirstUserAuthentication [9] should prevent attacks that require a reboot.

3.2.4 Network Security

User data that is transmitted over the network need to be protected. Therefore, Apple [18] recommends using the TLS protocol in order to maintain the integrity and confidentiality of sent data.

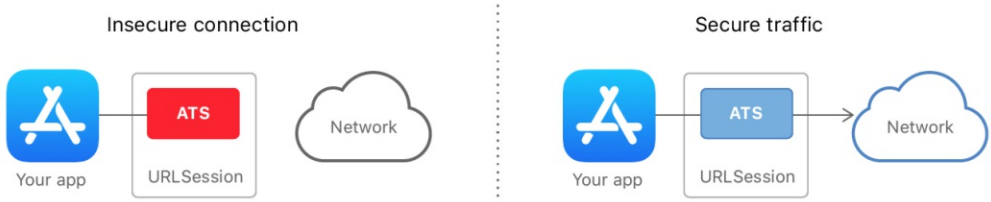

Figure 3.3: This illustration shows the application's usage of the ATS with an insecure connection and a secure connection. Figure taken from [32]

iOS [42] supports multiple TLS versions. Moreover, it prefers cipher suites with forward secrecy and Datagram Transport Layer Security (DTLS). Applications like Apple's Safari and Mail use this protocol per default to maintain an encrypted communication channel between network services and the device.

This section introduces some security mechanisms provided by Apple for securing an app's network communication.

App Transport Security

The networking security feature App Transport Security (ATS) [32] protects the integrity of privacy and data for every application. ATS ensures that applications use the TLS protocol utilizing strong ciphers and trusted certificates for network connections, for example, when using NSURLConnection or NSURLSession.

If a connection does not fulfill the minimum security requirements, it will be blocked by ATS, as shown on the right side of Figure 3.3. The unsecured connection is blocked by ATS and fails. Furthermore, a server's certificate has to be valid and signed using SHA-256 or stronger, requiring at least a 2048-bit RSA key or 256-bit elliptic curve key, according to Apple [42]. Moreover, forward secrecy and TLS 1.2 have to be supported. Since iOS 9, ATS is activated in applications by default.

Identity Pinning

If an app utilizes the TLS protocol, it checks the authenticity of the server and hence, increases its security. Identity pinning is something Apple [18] introduces to tighten the trust requirements of an app. Instead of accepting any certificate issued by a certificate authority, it limits the set of trusted certificates. Thus, an app that uses identity pinning will refuse the connection to a server whose public key is unknown.

3.3 Jailbreak

A jailbreak is defined by Kellner et al. [76] as an attack where privileges are escalated eliminating many of the restrictions set by Apple. Jailbreaking is a procedure that disables the codesigning mechanism of iOS, according to Thiel [121]. A jailbreak [81]

tries to break the system at a particular stage and utilizes iOS system vulnerabilities in order to achieve this goal. It cannot be applied on every iOS version as pointed out by Tamma et al. [120]. If a jailbreak utilizes a vulnerability fixed by an iOS system update, it will no longer be applicable.

There are tools for applying a jailbreak, for example, „checkra1n“ [2] and „unc0ver“ [3]. Most of the time „Cydia“ is automatically installed after jailbreaking. The Cydia app is a package manager for applications and tools for, among others, manipulating applications.

A jailbreak enables the installation of applications outside the sandbox. It is possible to execute code that is not approved by Apple and hence, applications from alternative sources can be installed. In addition, access to the file system can be obtained. Relan [108] indicates that jailbreaking enables a connection to the device via SSH. Hence, the contents of the directories can be read and files can be transferred.

[2] checkra1n, `https://checkra.in`, last accessed 2021/10/11
[3] unc0ver, `https://unc0ver.dev`, last accessed 2021/10/11

CHAPTER 4

Properties of iOS Applications

This chapter introduces the properties of iOS applications, including the architecture, developing, and deploying process.

4.1 Local Data Storage

Radwan and Zein [107] point out that almost every mobile application needs to store their data locally, for example, for caching backend data. There are different formats to store local data on iOS devices like Core Data, XML and plist, UserDefaults, Keychain, Log files, and SQLite files as outlined by Bojjagani and Sastry [51]. This section elaborates on SQLite databases, Core Data, UserDefaults, and Keychain.

4.1.1 SQLite Database

The SQLite database [1] is considered by Radwan and Zein [107] as one of the most popular ways to persist data on iOS. The authors point out that studies recommend using this database because of its easy handling, reliability and efficiency. SQLite can be used in Swift by using a wrapper, as described in Section 4.2. SQLite is considered a light version of complex relational database management systems like MySQL. Hence, it is less powerful compared to other systems. The engine of SQLite does not need a server.

4.1.2 Core Data

Radwan and Zein [107] point out that the iOS Core Data Framework [8] is a common approach to store data locally. Core Data can store the permanent data of applications for offline use, temporary cache data and provide undo-functionality in the application on one device. The framework can represent data as an entity-attribute model, which

[1] SQLite, `https://sqlite.org`, last accessed 2021/10/11

can be serialized, for example, into SQLite and XML. Core Data manages the instances of the object at runtime. The high-level abstraction representation of Core Data [107] allows direct communication with SQLite databases illustrated in Figure 4.1. Core Data is considered a fast and easy approach to store even large data without managing a database directly.

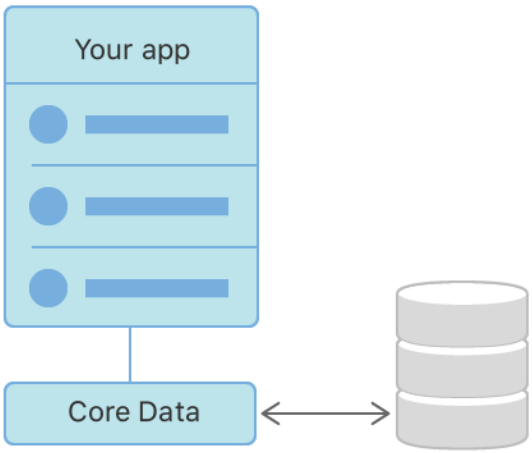

Figure 4.1: Core Data communicates directly with the database and the application. Figure taken from [8]

4.1.3 NSUserDefaults

The usage of user defaults [30], called NSUserDefaults, is considered by Radwan and Zein [107] as one of the most straightforward approaches to store user data in iOS applications. Data is stored as key-value pairs. A default object has to be a property list type. NSUserDefaults [107] should not be used to save large data because read and write operations have an impact on the performance of the application. Marin et al. [84] point out that user defaults are saved in the binary property list file, which is located in the preferences folder, as further described in Section 4.4. Data stored with user defaults is not encrypted and is not protected by Data Protection. Therefore, it is not recommended to use NSUserDefaults for saving sensitive data and credentials. However, NSUserDefaults can be used, for example, for saving settings data. Data in the UserDefaults can be retrieved by using specific tools and software even without jailbreaking the device.

4.1.4 Keychain

The keychain services API is a mechanism that enables the storage of short bits of data in the keychain, an encrypted database as outlined by Apple [24]. This mechanism should prevent, for example, that users choose weak passwords. Thiel [121] recommends to use the iOS keychain [24] to save sensitive data, including passwords, credentials, encryption

keys and certificates. Users can also store other sensitive data like credit card information or short notes.

Sensitive information has to be packaged as a keychain item in order to be able to store it in the database as stated by Apple [24]. Additionally, attributes need to be passed along with the data. Unlike data, attributes will not be encrypted. They are used for access control and search. Figure 4.2 illustrates that the data and attributes are handled by the keychain services for encryption and saved in the encrypted database. Processes that are allowed to use keychain services can find stored data and decrypt it. Keychain items [23] utilize two different AES-256-GCM keys for encryption. The first key is the table key used to de- and encrypt keychain metadata to enable fast search performances. The second key is a per-row key that is used for sensitive data.

The keychain is located on the file system and is implemented as an SQLite database. The database can only be accessed through the API, according to OWASP [93]. There is only one keychain available to all applications on iOS. Developers can sign multiple of their applications with the same key. Apps signed with the same key have access to data that is shared between applications from the same developer. The keychain data of an application remains on the device even if the application is uninstalled. On the other hand, data stored in the sandbox of the application is wiped. When selling the iOS device without carrying out a factory reset, the new owner can restore access to the previously used accounts by reinstalling the applications. A force wipe for data during the uninstallation process of an app does not exist. iOS application developers can implement that all keychain data of the application is wiped away when it is first launched after installation or that data is wiped when a user, for example, logs out. These implementations prevent access to accounts by unauthorized second users of the device.

Keychain items can have different attributes as described by Apple [22]. The *kSecAttrAccessible* attribute key controls the access to the item. These values sorted from least to most restrictive are possible:

- **kSecAttrAccessibleAlways:** The data is always accessible.
- **kSecAttrAccessibleAlwaysThisDeviceOnly:** The data is always accessible.
- **kSecAttrAccessibleAfterFirstUnlock:** The data is inaccessible until the first time the device owner unlocks the device.
- **kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly:** The data is inaccessible until the first time the device owner unlocks the device.
- **kSecAttrAccessibleWhenUnlocked:** The data is only accessible if the device is unlocked.
- **kSecAttrAccessibleWhenUnlockedThisDeviceOnly:** The data is only accessible if the device is unlocked.

- **kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly:** The data is only accessible if the device is unlocked and if a passcode is used on the device.

„ThisDeviceOnly“ prevents disclosure of the keychain items in backups on the computer and migration to a new device.

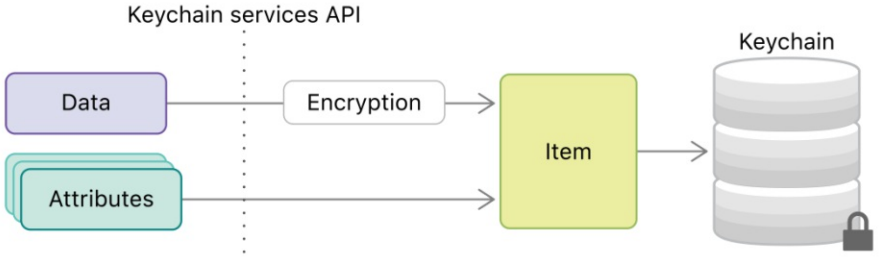

Figure 4.2: This illustration shows how data and attributes are stored in the keychain. Figure taken from [24]

4.2 Development of an iOS Application

Apple offers an iOS Software Development Kit (SDK) [45] that includes resources, technologies and tools for developing iOS applications. iOS applications can only be developed on Mac OS. Apple provides Xcode [46], a project and interface builder, free via its App Store.

Since 2015, Apple recommends using Swift [40] for iOS application development. Before, Objective-C was used for application development. Languedoc [80] highlights that Swift is interoperable with Objective-C, C and C++. Therefore, it is, for example, possible to create wrappers for a C library.

In addition to native applications, there are less common web view applications. A web view application has a browser embedded to display mobile-optimized websites as described by Relan [108]. iOS offers the WebKit [44] framework to include web content in the application.

4.3 Distribution of an iOS Application

One way to distribute an iOS application is to use the Apple's „App Store“ [2]. The App Store is a digital platform for distributing iOS applications. It allows users to download applications that are developed with the iOS SDK.

In order to develop and install developed applications on iOS devices, developers need to register in the Apple Developer Program [4] according to Apple [17]. Developers have to prove their real-world identity to obtain a signing certificate and to be allowed to publish their applications to the App Store. The App Store has requirements for

applications including security and privacy and inspects the application before it is published. Moreover, the application is analyzed for common bugs and if it works as described. This process should avoid the distribution of malicious or faulty applications. Garg and Baliyan [62] describes that the application is digitally signed and published on the App Store if the authenticity of the application can be verified. Kellner et al. [76] remark that if the analysis by Apple fails because of obvious bugs or violation of the App Store guidelines, it will be rejected and not listed in the App Store until a later analysis succeeds.

Apple offers another way to distribute apps to businesses only. First, the business has to register in the Apple Developer Enterprise Program [3] and fulfill their requirements, such as having at least 100 employees to create internal business applications. Then Apple checks the identity and eligibility of the business and provides a provisioning profile that allows the execution of in-house applications on the employees' devices. Finally, the employees need to install the provisioning profile to ensure that only eligible users can load and use the application on their devices.

4.4 iOS Application Properties

According to Thiel [121], iOS devices utilize Property List (plist) files for storing application configuration data. A plist file can either have a binary or XML format. It is a dictionary that stores hierarchical key-value pairs. Thiel [121] advises not to store sensitive information like credentials in plist files since they can be retrieved in plain-text.

Apple [25] [19] requires executable bundles to have an information property list file. This file is always called `Info.plist` and is stored in the root directory of the application. Most of the information stored in `Info.plist` are human-readable strings. This file stores information about the application like the bundle identifier (bundle ID) or supported iOS versions. A bundle ID [7] is a unique identifier of an application that can be registered, modified and deleted. For example, the bundle ID of the App Store is „com.apple.App Store“. Bundle IDs are needed in order to assign capabilities [6] or to create a provisioning profile [33].

Permissions

An application has to request needed permissions by using „entitlements“ [43] when it is installed. Entitlements [15] are key-value pairs that enable an application to utilize a service or technology and prevent increasing privileges by compromised components. They are signed to be protected against modifications and embedded in the code signature of the binary executable. Once the applications' requested permissions are granted and signed, they cannot be changed without having a differing signature.

The following list an extract of the entitlements defined by Apple [15]:

- Authentication
- Contacts
- Health
- Networking
- Push Notifications
- Security
- Sensors
- System
- Wallet

Applications on iOS [35] are not allowed to access user data without permission. The permissions are assigned based on the principle of least privilege. Thus, applications need to request further permissions from the user if the given ones are not sufficient. However, Apple allows third-party applications access to only a small set of permissions.

When an application tries to access a protected resource for the first time, a prompt appears where the user is asked for permission. Device settings enable users to check which applications they granted permissions and to revoke those permissions.

Devices [37] hold a lot of sensitive user data. For example, navigation apps require access to the user's current location, allowing them to process the location data. There is only limited access to protected resources on iOS, such as the user location.

A purpose string [36], also called usage description string, needs to be provided in the `Info.plist` file when an application request access to sensitive resources like location or photos. The purpose string should explain why the application needs access to the resource and is used in the prompt when asked for permission. If no string is available, the attempt to access the resource fails and the application might crash.

Figure 4.3 illustrates an example of a prompt and purpose string where an application called „MyRoute“ asks the user for permission to access the location. If the permission is granted by clicking „Allow While Using App“, the system saves the choice and does not ask again. If „Allow Once“ is chosen, the user will be asked again when the app is reopened. Otherwise, when „Don't Allow“ is selected, the application is not allowed to access the location and, therefore, cannot perform the action that needed this resource.

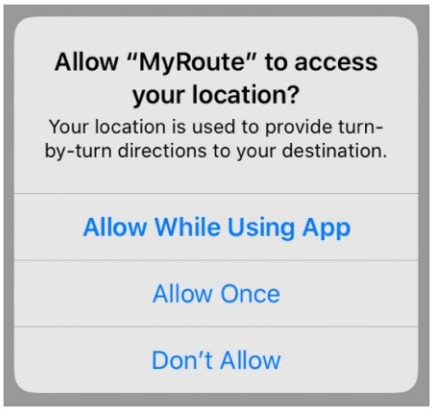

Figure 4.3: An example of a prompt on an iOS device where an application asks for permission to access the user's location. Figure taken from [37]

There are over 20 protected resources [34] that include:

- Camera and Microphone
- Files and Folders
- Health
- Location
- Networking
- NFC
- Security
- Wi-Fi

Structure

Resources, compiled application code, and metadata are bundled and then signed with the developers signing certificate as explained by Velu [125]. Afterward, the app can be published as an iOS App Store Package (IPA) in the App Store. An IPA is a zip-archive and therefore can be opened with file archive software like 7-Zip [2].

An application package, as seen in Figure 4.4, contains a „Payload" folder that includes the application data. Next to this folder, there is the `iTunesMetadata.plist` file that contains metadata of the application like the bundle identifier (ID). The „Payload" folder contains an „Application.app" file that includes the app binary, bundle resources, `Embedded.mobileprovision` and `_CodeSignature`. `Embedded.mobileprovision` enables developers to re-sign the iOS application without Xcode. `_CodeSignature`

[2] 7-Zip, `https://www.7-zip.org`, last accessed: 2021/10/11

allows the verification of the applications integrity. The executable in the bundle is in the Mach-O object file format.

Liu et al. [81] explain that a Mach-O file can be divided into three parts, as shown in Figure 4.5. The first part is the header that identifies the file format and includes details and information, for example, about the architecture. The next part contains the load commands that include segments where each describes a group of sections and specifies the file's layout and linkage details. The last part represents the data that includes the data that is linked to the load segments. A segment can include zero or more sections. A section can contain data of a specific code or type.

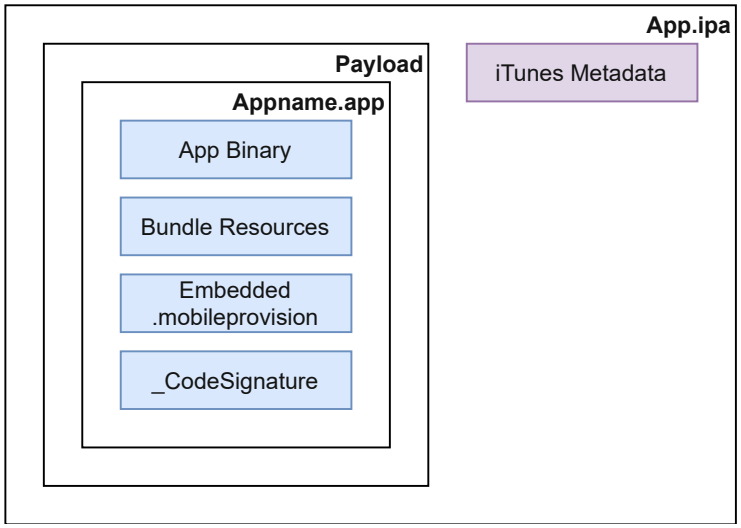

Figure 4.4: The structure of an IPA. Rebuilt after [125]

Directories

Applications store their data as outlined by Thiel [121] in the „Bundle“, „Data“ and „Shared“ directories that are introduced in this section.

A device backup includes all directories belonging and associated with an application, except for the „Library/Caches/“ directory. Hence, it is not recommended to store sensitive data cleartext in the application directories.

Bundle Directory. The „Bundle“ directory [121] includes a directory for every application installed on the device and is located in `var/containers/Bundle/Application/<app bundle id>`. The bundle ID of the application is used for its directory name. Each application directory contains an `<app name>.app` folder where the resources, `Info.plist` and binary is located.

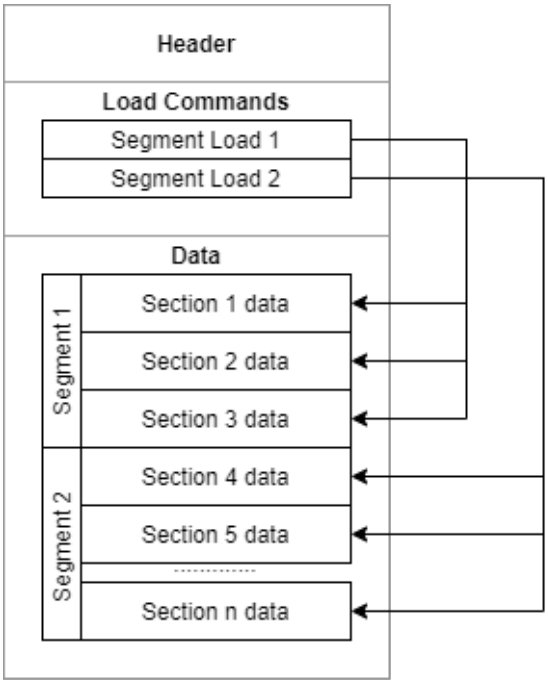

Figure 4.5: The Mach-O architecture. Rebuilt after [125]

Data Directory. The runtime data of an application [121] is stored in the „Data“ directory which is located in `/private/var/mobile/Containers/Data/Application/<app bundle id>`. The „Data“ directory contains a folder with its bundle ID as the name for each installed application. In those application folders, data like preferences, caches, and cookies are saved. Figure 4.6 illustrates how an application folder in the „Data“ directory is structured.

The „Data“ directory [121] has the following sub-directories:

- **Documents:** This directory saves non-transient data like content created by the user or local information that enables the application to run offline. It contains a folder called *Inbox* that includes files that other applications requested and need for specific actions. The files stored in this folder can only be read or deleted. A system process provides other application files in the inbox folder with more privileges since applications cannot write in foreign application directories.

- **Library:** This directory stores most of the application files and contains the following directories:

  - **Application support:** This directory stores additional files that the application utilizes, like purchased downloadable contents.
  - **Cookies:** This directory includes cookies placed by the URL loading system. For example when NSURLRequest [29] is called. Cookies are not shared between applications. Ryu et al. [109] point out that there can be a `Cookies.binarycookies` file in this directory that may include data like username and session ID that is used for automatic login.
  - **Preferences:** This directory saves the preferences of applications and can contain plist files. The applications are not allowed to write in this folder. For example, the NSUserDefaults handles the files there. The preference files are saved in plain-text.
  - **Saved application state:** This directory enables consistency by saving, for example, data entered into text fields by the user in the UI when the user switches to another application. In addition, developers can utilize the State Preservation API [31] to mark UI parts that should be included in the State Preservation.
  - **Caches:** This directory stores data for performance reasons. The stored data is transient and can change or disappear when the app is closed. For example, the URL load request-response caching provided by the NSURLCache [28] is stored there. In addition, the „Caches“ directory might expose sensitive user data cached by the app. App developers can prevent caching in this directory.
  - **SplashBoard:** The directory contains a „Snapshots“ folder that stores screenshots of the application that are auto-generated by iOS. They are used for UI animations if the application is moved into the background and might display sensitive data.

- **tmp:** This directory stores transient files similar to the „Caches“ directory. The application creates the files in the „tmp“ directory.

The Shared Directory. The „Shared“ directory [121] is intended to be used by applications that have a specific application group, for example, applications that utilize the keyboard. The „Shared“ directory includes applications for sharing data.

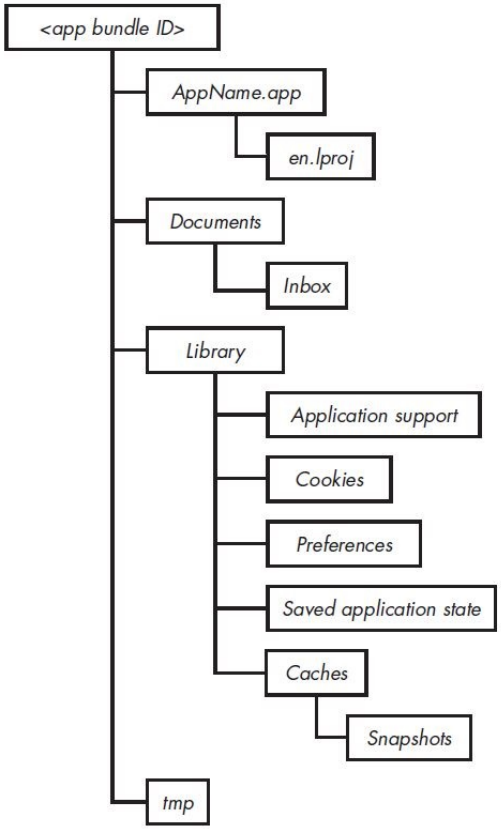

Figure 4.6: The layout of the application's „Data" directory. Figure taken from [121]

CHAPTER 5

General Aspects of Security Testing

Security testing is defined by Souppaya and Scarfone [119] as an information security assessment method where the actual and expected behaviors of one or more objects are compared under specified conditions. An information security assessment is a process where it is determined how well an entity, for example, a system or network, complies with particular security specifications.

This chapter presents different security testing and vulnerability detection techniques.

5.1 Testing Methods

Various testing methods can be used to examine mobile applications. The applied testing method mainly depends on the goal, the access to the source code and the availability of an executable based on the source code.

5.1.1 Application Access

An important factor is an access to the source code and binaries of an application. In total, two variations will be discussed in the following paragraphs. These variations can also be combined.

Source Only

The provision of the source code only means that it comes without a build environment or binary of the application. A tester can try to build a functional binary, but it could miss some required components. Therefore, only static analysis is usually utilized for the review if only the source code is available.

Ogata et al. [88] emphasize that the access to the source code of an application is an important aspect of application testing. A review of the source code aims to detect vulnerabilities, which can be very time-consuming. Therefore, tools can be used that offer the advantage of consistency between reviews and quick analysis of many code lines. Ogata et al. [88] recommend utilizing static analysis tools for automated as well as manual reviews. Based on the reviews, the found vulnerabilities can be classified using classifiers like CWE.

Binary Only

Liu, Tan, and Chen [82] describe that the binary code of an application can be examined if there is no access to the source code. Without access to the source code, a tester may rely on reverse engineering and dynamic analysis to analyze an app. The latter is introduced in Section 5.3.1.

Reverse engineering is defined as „the process of studying another company's product to see how it is made, sometimes in order to be able to copy it“ by the Cambridge Dictionary [102]. According to the OWASP [94], attackers usually retrieve the application from an app store. Then, they analyze the binary code using different tools in order to understand, for example, the control flow path of the app, which may reveal information such as of the backend servers or used cryptographic algorithms.

5.1.2 Black-Box Testing

Black-box testing observes, according to Khera, Kumar, and Nidhi Garg [77], the functionality of the system. The testers neither know much about the target architecture nor have access to the source code for this test. Therefore, testers have to rely on their knowledge and experience, as pointed out by Goel and Mehtre [67] or use standards and guidelines such as the OWASP.

5.1.3 White-Box Testing

In white-box testing [77], also called glass-box testing, testers have complete insight into the system architecture and configurations. They have access to, among other things, the source code, which allows the testers to deduce potential security flaws by analyzing them.

5.1.4 Grey-Box Testing

In grey-box testing [67] [77], testers are partly familiar with the system architecture. For example, testers have some information about the configuration of the system. On the other hand, the source code is commonly not available to the testers. Therefore, grey-box testing is considered as a combination of black-box and white-box testing.

5.2 Security Testing Techniques

Security testing techniques [119] can be categorized into analysis and validation techniques. Both can be conducted manually or by using automatic tools. Analysis techniques, such as scans and vulnerability assessments, aim to find useful information and identify potential vulnerabilities. Validation techniques, for example, penetration testing, confirm the existence of vulnerabilities. This section describes scans, vulnerability assessments, and penetration testing in more detail.

5.2.1 Vulnerability Scan

Vulnerability scans [119] are automatic tools utilized to detect vulnerabilities and find information about the host, the used operating system, and open ports in a system. Furthermore, the scans can be used to recognize deprecated software versions and misconfigurations. Moreover, they can validate conformity with a security policy. The found information can be used for further tests.

5.2.2 Vulnerability Assessment

A vulnerability assessment is defined by Shah and Mehtre [115] as a process where vulnerabilities are detected. In contrast to vulnerability scans, vulnerability assessments are primarily performed manually using different techniques that will be introduced in Section 5.3. Baloch [48] points out that the found potential vulnerabilities are documented in the vulnerability assessment summary, also called „findings summary“. Yaqoob et al. [127] add that the vulnerabilities are commonly ranked according to their risks and impact on the system in case of a real attack in the documentation.

5.2.3 Penetration Testing

Penetration testing [115] aims to prove that a vulnerability exists and analyze what information and accesses attackers can retrieve. Therefore, testers try to find out how intruders could circumvent the security features of a system by simulating attacks.

5.2.4 Applicability of the Techniques

The Figure 5.1 illustrates how the vulnerability scan, the vulnerability assessment, and the penetration testing techniques relate to each other.

A vulnerability scan analyzes the application for vulnerabilities using automatic tools. The scan is often part of the vulnerability assessment, which also includes analyzing the vulnerability scan output.

A vulnerability assessment aims to detect and analyze vulnerabilities. In addition, the risks and impacts on the application or systems are evaluated and documented in a findings summary.

Based on the results of the vulnerability assessment, a penetration test can be performed. In this phase [117], the testers try to prove the existence of the identified vulnerabilities from the previous steps by simulating attacks. In addition, this test helps to evaluate how difficult it is to perform the attack and the impact on the application or system.

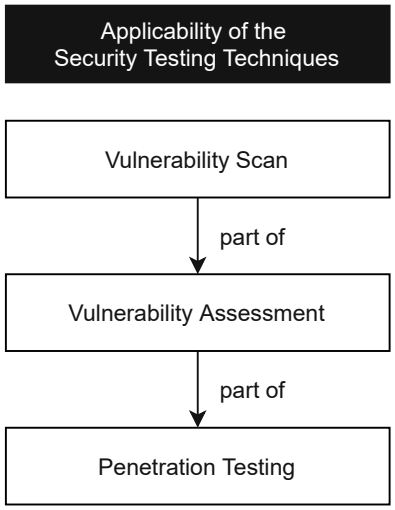

Figure 5.1: The applicability of the security testing techniques.

5.3 Vulnerability Detection Techniques

Static and dynamic analysis, as well as manual and automated testing, can be used to detect vulnerabilities. These techniques will be introduced in this section.

5.3.1 Static and Dynamic Analysis

Ogata et al. [88] describes that a static analysis observes the source code or binary code of an application. The analysis can provide results that precisely reflect the system's behavior and do not depend on test data input. The control flow of machine code is difficult to understand, especially when there are many functions. A common intermediate step when analyzing byte code is using a disassembler or a decompiler which allows an easier manual review and automatic analysis.

In contrast to static analysis, a dynamic analysis [88] concentrates mainly on the runtime behavior of the code. Therefore, the application is executed with different input cases, and the code is observed. A debugger can be used to highlight the currently executed code parts.

5.3.2 Manual Testing

Neither automated tools nor software are needed to conduct manual testing, as outlined by Goel and Mehtre [67] and Shah and Mehtre [115]. The responses of different test cases are manually monitored. The disadvantages of manual testing are that it is time-consuming, and testers require knowledge and experience. A distinction is made between systematic and exploratory manual testing.

Exploratory Testing. In exploratory testing [67] [115], the tester analyzes the code without a particular plan. Hence, the assessment depends on the experience of the tester.

Systematic Testing. In systematic testing [67] [115], a predefined test plan is followed. The plan is used for the full vulnerability assessment. All components of the system are taken into consideration to create the plan.

5.3.3 Automated Testing

Automated testing [67] [115] utilizes tools to detect vulnerabilities in the system. It connects the system to different test cases and compares the expected and actual results. If the results do not match, the system may contain misconfigurations and is considered vulnerable. Tools enable the repeated execution of all test cases. Hence, the testing time is reduced. A disadvantage of automated testing is that one tool cannot cover all vulnerability types.

5.3.4 Fuzz Testing

Fuzzing [115] [67], also called „Fuzz Testing“, is an automated or semi-automated testing technique. Invalid or inadequate random data is sent to the system to provoke error states like exceptions or crashes. Various types of vulnerabilities can be found with fuzzing. Moreover, since it can be executed at least semi-automated, it requires less human interaction. However, fuzzing requires manual preparation and configurations which might require experience.

CHAPTER 6

Conceptual Design of an iOS Vulnerability Assessment

This thesis aims to develop a concept for iOS eHealth application vulnerability assessments. The concept enables consistent assessments that cover the most common risks according to the OWASP Mobile Top 10 [98] and considers requirements for eHealth apps as outlined in Section 2.3.5. Therefore, checklists are provided to ensure that no common vulnerabilities are missed. The template for the iOS vulnerability assessment report is provided in the appendix.

The vulnerability assessment process of this concept, illustrated in Figure 6.1, consists of three phases, namely preassessment, information gathering and vulnerability assessment, and final analysis. The documentation is updated parallel to each phase. The process will be further described in this section.

The previous Chapter 5 introduced testing and vulnerability detection techniques that are utilized in the concept. The following Section 6.1 lists standards and guidelines that form the basis of the concept.

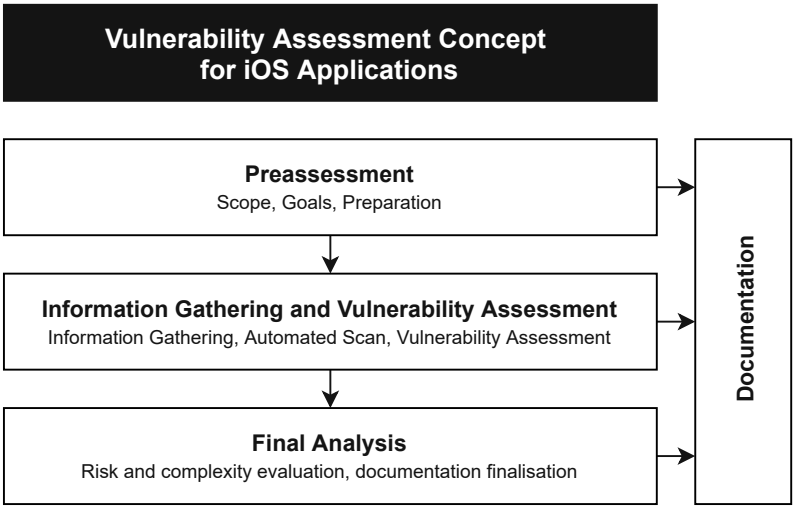

Figure 6.1: The vulnerability assessment's process of the concept for iOS eHealth applications.

6.1 Standards and Guidelines

Standards commonly aid to enhance the accuracy and efficiency of the vulnerability assessment. They help to maintain correctness and hence, lessen the failure risk. Yaqoob et al. [127] and Shah and Mehtre [114] name the Open Source Security Testing Methodology Manual (OSSTMM), Payment Card Industry Data Security Standards (PCI-DSS), OWASP and Information Systems Security Assessment Framework (ISSAF) as known vulnerability assessment and penetration testing standards and guidelines. These are only a few of many guidelines and standards.

This section introduces the OSSTMM, PCI-DSS, National Institute of Standards and Technology (NIST), BSI, and OWASP standards, and guidelines that are accessible for free.

6.1.1 OSSTMM

The OSSTMM is a manual for security testing. It is published by the Institute for Security and Open Methodologies (ISECOM) [1], which is a non-profit organization.

The tester has to decide what the test target is and how it is tested, as outlined by Herzog [68]. The OSSTMM comes with a Security Test Audit Report (STAR) that supports the testing process and should reduce the overall time for testing and reporting. The manual can be used for different purposes like penetration tests and vulnerability assessments.

The STAR report includes the following items:

1. Test time and date
2. Test duration
3. Names of responsible analysts
4. Test type
5. Scope of test
6. Index (method of target enumeration)
7. Channel tested
8. Test vector
9. Attack surface metric
10. Which tests have been completed, not completed, or partially completed and to what extent
11. Issues concerning the test and the validity of the results
12. Processes which have an impact on the security limitations
13. Unknowns or anomalies

[1] ISECOM, `https://www.isecom.org`, last accessed 2021/10/11

6.1.2 NIST

The NIST 800-115 is described by Souppaya and Scarfone [119] as a standard that serves as a guide for carrying out security assessments. This standard includes testing and analysis processes that can be used for assessments. It should support the establishment of security assessment policies as well as plans and should support the conducting of assessments. Moreover, the document offers guidance on what components should be assessed.

The four testing phases, namely planning, discovery, attack, and reporting, are illustrated in Figure 6.2. The planning phase creates rules and defines objectives used later during the active testing. The discovery phase consists of two parts, information gathering and active testing. Information gathering focuses on collecting information about the target, for example, server addresses or used software. Active testing includes the vulnerability analysis that utilizes vulnerability databases like NVD. In the attack phase, which is the core part of penetration testing, previously found vulnerabilities are exploited. The reporting phase runs parallel to the other three phases. The resulting report contains the detected vulnerabilities, risk estimation, and mitigation strategies.

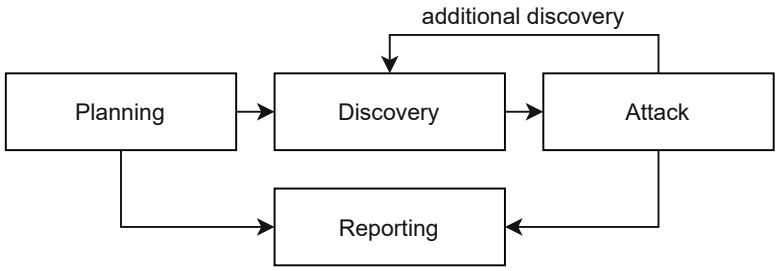

Figure 6.2: The four penetration testing phases by the NIST. Rebuilt after [119]

6.1.3 OWASP

The Mobile Application Security Verification Standard (MASVS) [96] is published by the OWASP and provides basic mobile app security requirements. These basic requirements can be used as reference in vulnerability assessments and penetration tests to maintain consistency and ensure completeness.

The Mobile Security Testing Guide (MSTG) [97] is a manual published by the OWASP for mobile Android and iOS application security testing. There are static and dynamic testing descriptions for finding vulnerabilities in iOS and Android applications. Moreover, the MSTG provides a checklist that can be used for security assessments. The testing guidelines can be linked with MASVS requirements.

Figure 6.3 shows how the MASVS is related to the MSTG: The MASVS provides mobile application security requirements and the MSTG utilizes these requirements for the testing guide.

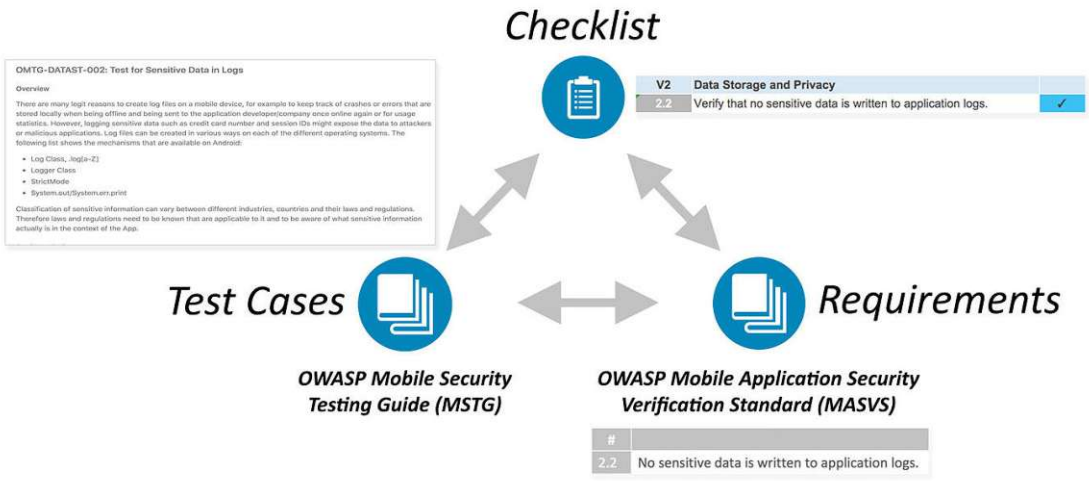

Figure 6.3: The relationship of the OWASP MASVS and MSTG. The checklist is part of the MSTG. From [97]

6.1.4 PCI-DSS

The PCI-DSS is described by Shah and Mehtre [114] as a proprietary standard for the security of cardholder information like from credit and debit cards. The PCI-DSS standard is published by the PCI-DSS Security Standards Council.

The PCI Security Standards Council [55] created the PCI-DSS as guidance for penetration testing. The PCI-DSS introduces penetration testing components and explains the distinction between penetration testing and vulnerability scans. Moreover, the PCI-DSS provides penetration testing reporting guidelines, including a checklist that can be used for testing.

The three testing phases pre-engagement, engagement, and post-engagement, are illustrated in Figure 6.4. The pre-engagement phase includes organizational and planning steps. In these steps, specifications like success criteria and used methodology are defined. The PCI Security Standards Council [55] recommends to use best practices like the OSSTMM, the NIST 800-115 standard, and OWASP security testing guides. It is emphasized that scoping is important, especially for complex environments. For example, a scope could include all systems that directly process, save or transmit data and exclude specific servers that do not process user data. The next phase is the engagement phase that focuses on the information gathering, vulnerability detection, and exploitation of found vulnerabilities. In the end, found vulnerabilities are classified as *high*, *medium* or *low* according to their risk. For example, *high* rated vulnerabilities include code execution, *medium* rated vulnerabilities include weak SSL ciphers, and *low* rated vulnerabilities include slow HTTP denial-of-service attacks. The last phase, post-engagement, analyzes the identified vulnerabilities and provides recommendations and remediation for these. Finally, all vulnerabilities and their remediation are summarized in a report.

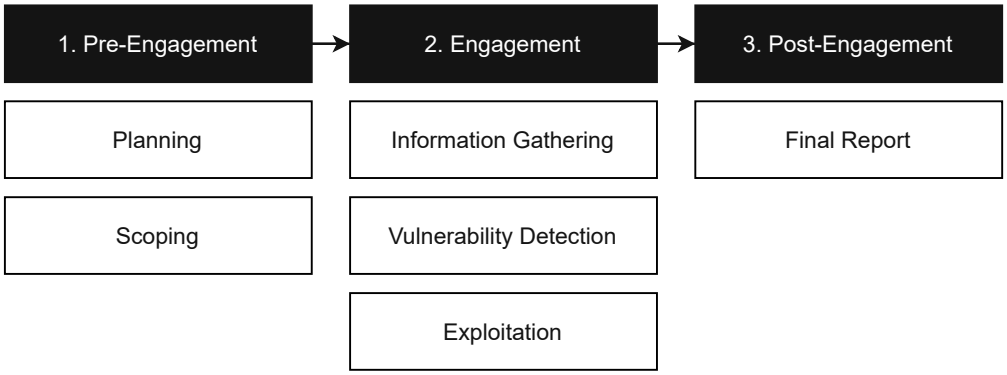

Figure 6.4: The three penetration testing phases by the PCI Security Standards Council. Rebuilt after [55]

6.1.5 BSI

The BSI [73] proposes a concept that should enhance the efficiency of penetration testing. It provides fundamental terminology regarding IT security and penetration testing. Moreover, the concept includes a penetration test classification and objectives and outlines a penetration testing process.

In addition, the BSI [72] published security requirements for digital health applications. This technical guideline has a similar structure to the OWASP MASVS and can be utilized by app developers and provides best practices for processing and storing sensitive data.

This section introduces the penetration testing method and classification by the BSI.

Penetration Testing Method

The penetration testing method by the BSI, as illustrated in Figure 6.5, consists of the following five phases:

1. **Preparation:** The first phase concentrates on the preparation of the testing and defines the scope and goals.

2. **Information Gathering:** The second phase focuses on information gathering. This phase is considered passive penetration testing. It aims to obtain a detailed and complete overview of all available systems, including potential attack vectors.

3. **Information and Risk Evaluation:** In the third phase, the gathered information and the risks are evaluated. This phase is needed to ensure efficient active penetration testing. The evaluation includes potential threats to the systems and complexity to exploit found vulnerabilities.

4. **Active Penetration Testing:** The BSI considers this phase as the most difficult one. Potential consequences that can occur by exploiting vulnerabilities have to be taken into account. Active penetration testing confirms if found vulnerabilities in previous phases represent actual risks.

5. **Final Analysis:** In the last phase, a final analysis is conducted. It includes the evaluation of found vulnerabilities.

Parallel to each phase, documentation should be created and updated. The documentation in each phase ensures that all analysis steps and results are listed.

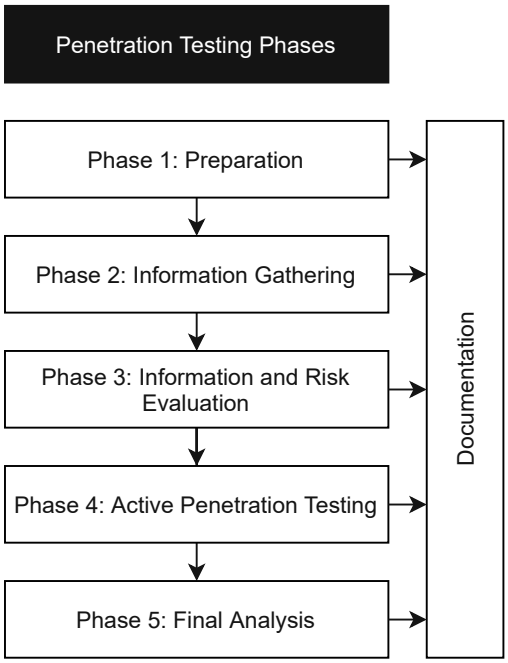

Figure 6.5: The penetration testing phases by the BSI. Rebuilt after [73]

Classification

The BSI [73] introduces a classification for penetration tests, illustrated in Figure 6.6, that presents how penetration tests can be categorized in order to support the planning of the test process. This classification can also be used for vulnerability assessments. The left side shows six criteria, and the right side shows possible values. The basis of this information is the state of knowledge of the tested system. It distinguishes between black-box and white-box testing. The second criterion is aggressiveness that includes four different levels. On the highest level, the goal is to exploit all potential vulnerabilities. On the other hand, only passive scanning that analyzes components but does not exploit them is used in the lowest level. For vulnerability assessments, only the lowest level is used since no active attacks are performed. The third criterion determines the scope

of the test that can be full, limited, or focused. The BSI recommends conducting a full analysis when the system is reviewed for the first time. The next criterion is the selection of the approach that can be open or covert to prevent detection. The following criterion deals with the technique used for the test that includes network access, other communication, physical access, and social engineering. The BSI points out that the network is usually to perform attacks during penetration testing. Social engineering takes advantage of the lack of security awareness to gain information. The last part of the classification describes if the test starts from inside or outside.

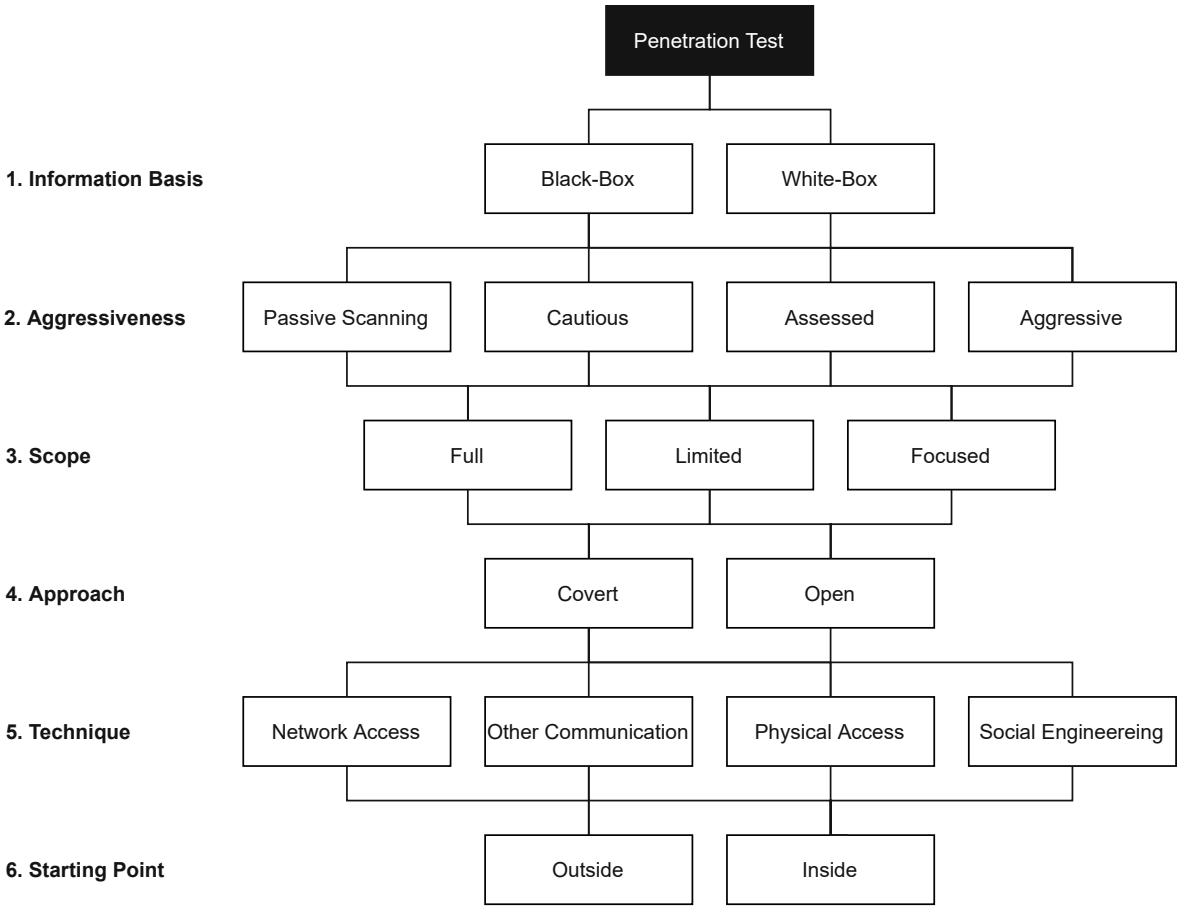

Figure 6.6: The penetration testing classification by the BSI. Rebuilt after [73]

6.2 Preassessment

The preassessment phase covers the organizational and planning steps of the assessment. It is divided into scope and goals, as well as a preparation phase.

6.2.1 Scope and Goal

The scope defines what areas of applications are supposed to be analyzed. The goal describes what the vulnerability assessment aims to find out.

The scope of the vulnerability assessment only includes the frontend in form of an iOS application and will not include a backend analysis. Moreover, the system of the target will not be actively attacked. Hence, the vulnerability assessment will not affect the target system's performance. The assessment will mainly cover the first three risks of the OWASP Mobile Top 10 list enumerated in Section 2.2.4, namely, the improper platform usage, insecure data storage, and insecure communication.

The vulnerability assessment will be classified according to the BSI classification for penetration tests introduced in Section 6.1.5 using the template shown in Table 6.1.

Test Classification	
Criteria	**Value**
Information Basis:	Black-Box
Aggressiveness:	Passive Scanning
Scope:	Full/Limited/Focused
Approach:	Covert/Open
Technique:	Network Communication/Other Communication/Physical Access/Social Engineering
Starting Point:	Outside/Inside

Table 6.1: The test classification template of the concept, inspired by the BSI [73].

The assessment approach and test method used in the vulnerability assessment and their respective possible values will be documented in a template as shown in Table 6.2.

Depending on the application access, a proper testing method approach, introduced in Section 5.1.1, needs to be determined. Since there will probably be no design documentation of the apps, a hybrid assessment approach is used and high-level characteristics are gathered in the first step of the information gathering process.

The vulnerability assessment aims to determine if the sensitive health data stored and transmitted by the application is protected against unauthorized access.

Date:	
Assessment Approach:	Hybrid
Test Method:	Binary Code Testing

Table 6.2: The template of the concept for the used approach and testing methods in the vulnerability assessment.

6.2.2 Testing Setup Preparation

Setting up the test setup is part of the preassessment phase and includes preparing the used hardware, tools, and test data.

First, the utilized hardware needs to be prepared. This concept is designed for binary code and black-box testing. Therefore a jailbroken iOS device is required. Commonly, a computer is necessary to jailbreak the device and for further analysis. It is recommended to run and test the application as well as insert test user data without a jailbreak first and jailbreak the iOS device afterward. Some applications have a jailbreak detection implemented that prevents launching and analyzing the application if the device is jailbroken, requiring additional preparation.

The next step is to retrieve the application for the analysis. First of all, the application is downloaded from the Apple App Store. Then, the downloaded IPA is dumped from a jailbroken iOS device using tools such as frida-ios-dump [2] and prepared on the computer for further analysis.

6.3 Information Gathering and Vulnerability Assessment

The information gathering and vulnerability assessment phase focuses on collecting useful information about the application and finding vulnerabilities.

In order to avoid missing parts and paths when analyzing the application, the tester commonly follows a plan. For the analysis, checklists will be used that are inspired by the OSSTMM [68], the BSI [73] [72], and the OWASP [95] that were introduced in Section 6.1.

Checklists are divided into three parts. The first part focuses on the information gathering process. The second part is used to gather results from an automated IPA scan. Finally, the third part focuses on the actual vulnerability assessment process.

6.3.1 Information Gathering

The description of the app in the App Store and the functionalities by running and using the app are examined to find useful information. This phase utilizes automated or manual testing tools, or a combination of both to collect information, according to Shebli and Beheshti [116].

The information gathering checklist template is shown in Table 6.3. A checklist item consists of an ID and a description.

The first part of the checklist is used to gather publicly available data. Information about the version, category, iOS compatibility, and purpose of the application is gathered and retrieved from Apple's App Store. Moreover, the app's App Store page lists which data linked to the user's identity is collected by the application.

The second part of the checklist is used to gather information about the application publisher and developer. The application developer is usually a software development company hired by the publisher to develop the application.

[2] frida-ios-dump, `https://github.com/AloneMonkey/frida-ios-dump/tree/56e99b2138fc213fa759b3aeb9717a1fb4ec6a59`, last accessed: 2021/10/11

ID	Description	
I1 Evaluation of publicly available data		
I1.1	App Name:	
I1.2	App Store Link:	
I1.3	Analyzed Version:	
I1.4	Application category:	
I1.5	iOS compatibility:	
I1.6	Purpose of the application:	
I1.7	Collected data linked to the user's identity:	
I2 Information about the application publisher		
I2.1	Publisher name:	
I2.2	Publisher website:	
I2.3	Developer name:	
I2.4	Developer website:	

Table 6.3: The information gathering checklist of the vulnerability assessment.

6.3.2 Tool-Based Scan

The checklist shown in Table 6.4 is used as guideline to evaluate the results of the tool-based scan of the application. The Mobile Security Framework (MobSF) [3] is used for the automated static analysis of the IPA.

The tool-based scan checklist consists of three parts: information about the system, ATS, and IPA binary analysis. It has three columns: ID, description, and status. Each checklist item has a unique ID that can be used later as a reference in the report. The second column describes the checklist item. The last column shows the result that is either *OK*, *not ok (NOK)* or contains app information. If the result was *NOK*, a detailed description of the result is provided.

The first part of the checklist is utilized to list the used programming languages, the ID of the app, and the requested permissions according to the app's `Info.plist` file.

The second part is used to evaluate if ATS, introduced in Section 3.2.4, is enabled. ATS enforces security mechanisms for the communication between app and server, including the use of TLS. It can be disabled to allow insecure HTTP requests. For example, NSExceptionAllowsInsecureHTTPLoads [27] can be set to *YES* to enable insecure HTTP loads for a given domain.

The last part checks if the following toolchain security features [92], that are activated by default by Xcode, are activated:

[3] MobSF, `https://github.com/MobSF/Mobile-Security-Framework-MobSF/releases/tag/v3.4.3`, last accessed: 2021/10/11

- **Automatic Reference Counting (ARC):** This is a feature for memory management that offers a security mechanism against memory corruption vulnerabilities.
- **Stack Canary:** Stack canaries are used to prevent buffer overflows.
- **Position-Independent Executable (PIE):** This activates ASLR for the binary.

ID	Description	Result
T1 Information about the system		
T1.1	Programming languages:	
T1.2	App Identifier:	
T1.3	App Permissions:	
T2 App Transport Security		
T2.1	ATS is enabled. No insecure connections are configured.	
T3 IPA Binary Analysis		
T3.1	ARC is enabled.	
T3.2	The No eXecute (NX) bit is set in the binary.	
T3.3	The stack canary value is added to the stack of the binary.	
V3.4	PIE support is activated.	

Table 6.4: The tool-based scan checklist of the vulnerability assessment.

6.3.3 Vulnerability Assessment

In the vulnerability assessment, the target system is examined to find security issues. Gathering basic information about the target online, like in the App Store or website of the application publisher first, is recommended by Baloch [48]. The quality of a vulnerability assessment depends on the tester's understanding of the target system, according to Goel and Mehtre [67]. Moreover, systematic information gathering helps to build an accurate security status model of the target system.

The vulnerability assessment checklist has four parts: design and privacy, network communication, authentication management, and data storage. The last three parts are based on the first three risks of the OWASP Mobile Top 10 list enumerated in Section 2.2.4, namely, improper platform usage, insecure data storage, and insecure communication.

The checklists consist of the three columns ID, description, and status. Each checklist item has a unique ID that is later used as a reference in the final analysis. The second column includes a description of the checklist item. Finally, the last column depicts the status of the checklist item that has one of the following three values: *OK*, *NOK* or *not available (N/A)*. If the value equals *N/A*, it means that the checklist item could not be

checked. If the result is *NOK* or *N/A*, the issue that leads to this result will be described in detail.

V1 Design and Privacy

The first part of the checklist, shown in Table 6.5, is used as a guideline to explore the „Design and Privacy“ of the application. This part is conducted by running and testing the application on a non-jailbroken physical device. It mainly explores the functionalities of the app and aims to get familiar with it.

The assessment starts with launching the application to conduct further content analysis. The content analysis includes legal information and data collecting since eHealth apps must comply with the GDPR as described in Section 2.3.1.

Next, the analysis continues with inspecting functionalities like notification options or the possibility of withdrawing consent and reporting security problems.

Finally, the assessment focuses on analyzing if the user interface or error messages expose sensitive information such as passwords.

The following paragraphs describe the checklist items in detail.

V1.1. The Apple App Store provides a link to an app's privacy policy and describes what data linked to the user the app collects. However, when users launch the app for the first time, the app should inform them about its primary purpose. Additionally, the app needs to provide the information required by the GDPR, as listed in Section 2.3.1. The checklist item status will be *OK* if the app provides information about the usage of data before users enter personal data, such as an e-mail address. If the app shows the information after user data was collected and processed or if the information is missing, the status results in *NOK*.

V1.2. If the app collects and processes only data that serves its primary purpose, the status will result in *OK*. For example, if the app's primary purpose is to track the menstrual cycle, but it collects or processes the user's location, it will lead to *NOK*.

V1.3. The `Info.plist` might contain purpose strings that explain why the app needs access to specific resources. If the app needs the requested permissions for its primary purpose, it will lead to the *OK* status. For example, an eHealth app might require access to the Bluetooth interface in order to be able to connect to a measuring device.

V1.4. Users need to give their consent before any PII is collected or processed, for example, by ticking a checkbox („I agree to ...“). If the app does not obtain a declaration of consent, the status will be *NOK*.

V1.5. Applications that process sensitive data should recommend security mechanisms that users can apply to protect their data locally, such as using a device passcode and not jailbroken devices. If the app provides no safety notes, the status results in *NOK*.

V1.6. This checklist item verifies if the app detects a jailbroken device. The status results in *OK* if users are warned or prevented from using the app in case they use a jailbroken device.

V1.7. The app needs to explain how users can withdraw their consent and what consequences it has. For example, if a user withdraws consent, it leads to the deletion of the account and user data. This case will result in *OK*.

V1.8. If the app provides no push notification functionality, this checklist item is *N/A*. The status results in *OK* if push notifications are optional and legitimate. For example, a logbook app might require push notifications in order to remind the user to add a log entry.

V1.9. Error messages shown in the app should not contain any sensitive data. Moreover, error messages should not display application or exception messages, such as „500 internal server error“. For example, if the server throws an exception, the error should not be displayed directly in the app. If the app discloses any information that is not relevant to the user, the status will be *NOK*.

V1.10. The application should provide a contact method, such as an e-mail address, and information for reporting security problems to get an *OK* in the checklist. For example, if an application provides functionality for sending developers the app logs, users should be informed how to describe the encountered issues. Users might receive a support ID if they send a log to the developers. Then, they can use the ID in an e-mail describing the encountered issue.

V1.11. The app should mask input fields containing sensitive data such as passwords. If the input is displayed as plaintext by default, the status will be *NOK*.

V1 Design and Privacy		
ID	**Description**	**Status**
V1.1	The app discloses its primary purpose and the use of personal data before installation (e.g., in the description of the app store). The user is informed about this, at least when the app is first launched.	
V1.2	The app only collects and processes data that serves its primary purpose.	
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	
V1.5	The app educates about security best practices the user should follow in using the app (e.g., using a passcode)	
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	
V1.9	Error messages do not contain sensitive data.	
V1.10	The application provides a simple option to report security problems.	
V1.11	No sensitive data (e.g., passwords, pins) is exposed through the user interface.	

Table 6.5: The vulnerability assessment's „Design and Privacy“ checklist.

V2 Network Communication

Commonly, eHealth apps either send or retrieve data from a server. Thus, it is necessary to analyze the network communication of an app. The results of this analysis are tracked using a checklist illustrated in Table 6.6.

The network communication part of the vulnerability assessment checks the following properties that have been, to some extent, introduced in Section 3.2.4:

- The usage of the TLS protocol
- The presence of certificate pinning
- The hosts the app connects to

This part utilizes the Burp proxy [101] to analyze the communication between the application and remote endpoints. It is necessary to add the self-signed certificate of Burp to the iOS device's trust store in order to intercept the connections protected by TLS. Otherwise, the connection setup will fail.

Based on the intercepted network traffic, it is possible to infer data sent to the server or the used authentication method. The latter will be more thoroughly discussed in Section 6.3.3.

The following paragraphs describe the „Network Communication“ items in detail.

V2.1. This item checks if the app uses the TLS protocol for every request to a remote endpoint. Therefore, different outgoing requests are analyzed using Burp. If one request does not use TLS, it will lead to *NOK*.

V2.2. If the application verifies the certificate of the remote endpoint and hence, the traffic is not visible without further configurations in Burp, the result will be *OK*.

V2.3. If the application supports certificate pinning, the result will be *OK*.

V2 Network Communication		
ID	**Description**	**Status**
V2.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	
V2.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	
V2.3	The app supports certificate pinning.	

Table 6.6: The vulnerability assessment's „Network Communication“ checklist.

V3 Authentication Management

eHealth apps commonly include features that require the user to authenticate either locally or remotely to a backend.

A premise to check for authentication is that a test user is registered beforehand. First, a test user with a simple password, such as „1234“ is created. Then, the network traffic is dumped using the network traffic interception setup described in 6.3.3.

This procedure allows checking if a password policy is in place or if communication with a backend is necessary for the authentication process. Information gathered during the assessment process is collected in a checklist as shown in Table 6.7.

The following paragraphs describe the „Authentication Management“ checklist items in detail.

V3.1. This checklist item analyzes if the app performs the authentication at a remote endpoint. Therefore it is analyzed with Burp if the app sends the authentication request to a server. If the app performs the authentication locally on the device, the status results in *NOK*.

V3.2. If the app has a password policy such as a minimum number of characters and enforces it, the status will be *OK*.

V3.3. If users can change their password in the app, the result will be *OK*.

V3.4. This item checks if the app prevents attacks such as brute-forcing on login, as described by OWASP [91]. For example, if it is possible to enter a wrong password ten times without consequences like delays in subsequent logins or temporary locks, the status results in *NOK*.

V3 Authentication Management		
ID	**Description**	**Status**
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	
V3.2	A password policy exists and is enforced.	
V3.3	Users can change their passwords.	
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	

Table 6.7: The vulnerability assessment's „Authentication Management“ checklist.

V4 Data Storage

The last part of the vulnerability assessment focuses on examining the data storage. It uses the checklist in Table 6.8 to analyze if user data, especially sensitive health data, can be retrieved by unauthorized persons.

A jailbroken device is used in order to find the data stored on the device. Otherwise, it is not possible to examine an app's directory, due to the file permissions enforced by the operating system as outlined in Chapter 3.

There are multiple locations where applications might store sensitive data. All of the following storage locations have been previously introduced in Chapter 4:

- Info.plist
- Keychain
- NSUserDefaults
- Core Data
- *Snapshot* directory

Different tools, such as Objection [4] are used to extract the information stored in each location.

The following paragraphs describe the „Data Storage" checklist items in detail.

V4.1. Sensitive data stored in the keychain should be protected, ideally by using `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`. The status results in *NOK* if the app's keychain is insecurely configured using `kSecAttrAccessibleAlways` or `kSecAttrAccessibleAlwaysThisDeviceOnly` or exposes sensitive data in cleartext.

V4.2. If insecure storages, such as the NSUserDefaults, Core Data, and `Info.plist` contain sensitive user data, the status will be *NOK*.

V4.3. The status results in *OK* if the app has a mechanism that prevents exposing sensitive data in auto-generated snapshots by the operating system. Therefore, the *Snapshots* directory is examined to verify if it contains images that reveal sensitive data.

V4.4. Data found during the assessment that is hashed using deprecated algorithms such as MD5 leads to an NOK status.

[4] Objection, `https://github.com/sensepost/objection/releases/tag/1.11.0`, last accessed: 2021/10/11

V4 Data Storage		
ID	**Description**	**Status**
V4.1	The keychain is used to store sensitive data. They are sufficiently protected.	
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	

Table 6.8: The vulnerability assessment's „Data Storage“ checklist.

6.4 Final Analysis

Lastly, the findings are summarized and categorized into vulnerabilities and miscellaneous issues, using the templates shown in the tables 6.9 and 6.10.

Each finding has a unique ID, starting with the app's name and a sequential number. Thus, if it was found while assessing a checklist item, then the ID of the checklist item is noted alongside a brief description and how it was discovered.

Miscellaneous issues include user interface issues or a lack of information for the users regarding their private information.

Additionally, vulnerabilities are evaluated according to the CVSS 3.1 standard [70]. The CVSS 3.1 standard takes, among other things, the attack complexity, the required privileges, and the required user interaction into account. Furthermore, references to similar problems found in the CVE or OWASP guidelines are documented.

ID:	
Checklist ID:	
CVSS Score:	
References:	
Title:	
Description:	

Table 6.9: The template for found vulnerabilities in the vulnerability assessment.

ID:	
Checklist ID:	
Title:	
Description:	

Table 6.10: The template for found miscellaneous issues in the vulnerability assessment.

CHAPTER 7

Execution of a Vulnerability Assessment Targeting eHealth iOS Applications

This section uses the concept from the previous Chapter 6 to conduct vulnerability assessments of chosen applications from the Apple App Store.

7.1 Application Target Selection

Mainly apps that were recently mentioned in the media in Germany and Austria were selected for a practical vulnerability assessment.

Section 2.3.3 describes that people insured by the statutory health system have the right to optionally use the ePA from 1st January 2021 onwards. Thus, the ePA app of the biggest health insurance company was chosen for the vulnerability assessment. The Techniker Krankenkasse (TK) is the largest statutory health insurer in Germany, according to Krankenkassen [78]. IBM [1] developed the TK ePA app.

Another app that was chosen for the analysis is the free health app Ada [2] that had massive privacy deficiencies in the past that were found by Gieselmann, Tremmel, and Eikenberg [66]. Ada is still one of the most popular health apps in the Apple App Store. This app was selected for a vulnerability assessment in order to check if there are still deficiencies.

MySugr [3] is an Austrian company that was introduced by Palmai [100] in the „Der Standard“ newspaper. They developed an application that aims to improve diabetes

[1] IBM, `https://www.ega.de`, last accessed: 2021/10/11
[2] Ada, `https://ada.com`, last accessed 2021/10/11
[3] mySugr, `https://www.mysugr.com`, last accessed 2021/10/11

therapies in the long term and is, as outlined on their website, a verified medical device following the International Organization for Standardization (ISO) 13485. Hence, it must respect the highest security standards in order to maintain the integrity and confidentiality of the user's health data. Therefore, the mySugr app was chosen to verify if it protects sensitive data sufficiently.

The last eHealth application for the practical assessment, Femometer [4], was randomly selected from the "Health & Fitness" category in the App Store. The App Store, as well as online shopping platforms such as Amazon, advertise this fertility tracker app. The Femometer app helps track fertility signs, such as the period, LH tests, and interprets the user's fertility by combining all key symptoms. Therefore, a user needs to provide different health data.

7.2 Preassessment

The preassessment is outlined in this section since it is the same for all four practical vulnerability assessments.

7.2.1 Scope and Goal

The used approach and testing method of the vulnerability assessments are summarized in Table 7.1.

Date:	20.05.2021
Assessment Approach:	Hybrid
Test Method:	Binary Code Testing

Table 7.1: The the used approach and testing methods in the practical vulnerability assessments.

The scope of the vulnerability assessments only includes the frontend iOS application and will not include a backend analysis. Moreover, no active attacks will be performed.

[4] Femometer, `https://www.femometer.com/app/femometer-app.html`, last accessed 2021/10/11

The vulnerability assessments will cover the following areas:

1. Design and Privacy
2. Network Communication
3. Authentication Management
4. Data Storage

The classification of the vulnerability assessments is described in Table 7.2.

Test Classification	
Criteria	**Value**
Information Basis:	Black-Box
Aggressiveness:	Passive Scanning
Scope:	Focused
Approach:	Covert
Technique:	Physical Access
Starting Point:	Outside

Table 7.2: The test classification of the practical assessments.

The goal of the vulnerability assessments is to identify vulnerabilities in the covered areas and find out if sensitive data entered and transmitted during the assessments is sufficiently protected against unauthorized access.

7.2.2 Testing Setup

This section describes the setup used for the practical vulnerability assessments.

Hardware

An „iPhone SE“ without jailbreak is used for testing the usual processes that reflect the expected behavior by app developers.

An „iPhone 8“ jailbreaked with „checkra1n“ [5] is utilized for retrieving the IPA file as well as the app data from local storages and for examining the exceptional processes.

The iOS version of both devices is 14.3 [21].

An Apple „MacBook Pro“ is used for preparations such as installing the jailbreak and for analyzing the IPA files as well as app data.

[5] checkra1n, `https://checkra.in`, last accessed 2021/10/11

Software and Tools

frida-ios-dump [6] is utilized for decrypting and extracting the IPA files.

The MobSF [7] is utilized for the tool-based scan.

Objection [8] is used for examining the local storages of the app and disabling security mechanisms such as certificate pinning in the apps.

The Burp suite [101] is utilized to analyze the network communication of the app. A proxy needs to be set up in Burp to test the security mechanisms. When the application launches, the traffic should be visible. Otherwise, errors will be listed in the „Alerts“ tab. If the traffic is visible, then no validation of the certificate was done. On the other hand, if the traffic is not visible and a SSL handshake failure occurred, it means that the certificate was checked. In this case, the Burp certificate needs to be installed to test if the certificate is validated against the device's trust store. If the handshake was successful, the traffic should be visible in Burp. Otherwise, it is an indication that the application uses certificate pinning.

Test Data

The following data will be used as test data if the applications require specific information. The utilization of this data simplifies the verification processes, for example, if this data is stored in insecure storage.

Additionally, the MD5 hashes of the passwords and passcodes are listed here and searched for in the device storages. The hashes were calculated with dCode [9].

- **E-Mail:** `vulnerabiliy.assessment@vancy.at`
- **Password:** `MyPassword1234` (MD5: `dbeb1c353a8a988d19460a0c30b5aa08`)
- **Birthday:** `01.02.1993`
- **Gender:** `Female`
- **Passcode:** `1234` (MD5: `81dc9bdb52d04dc20036dbd8313ed055`)
- **KVNr:** `A01234`

[6] frida-ios-dump, `https://github.com/AloneMonkey/frida-ios-dump/tree/56e99b2138fc213fa759b3aeb9717a1fb4ec6a59`, last accessed: 2021/10/11

[7] MobSF, `https://github.com/MobSF/Mobile-Security-Framework-MobSF/releases/tag/v3.4.3`, last accessed: 2021/10/11

[8] Objection, `https://github.com/sensepost/objection/releases/tag/1.11.0`, last accessed: 2021/10/11

[9] dCode MD5, `https://www.dcode.fr/md5-hash`, last accessed: 2021/10/11

Each German insured person receives a KVNr, in German „Krankenversicherungsnummer“ from its health insurance, according to KrankenkassenInfo [79]. The KVNr enables the electronic management of the health insurance account. It consists of 19 to 30 digits depending on the health insurance and is part of the eHealth card. Ten digits are visible on the card, as highlighted in Figure 7.1, and are used for the login to the ePA. The KVNr starts with an uppercase letter followed by nine digits. For the vulnerability assessment, an invalid KVNr will be used in order to avoid problems for the owners of the KVNr. The apps need to validate the KVNr. Otherwise, it will count as a vulnerability assessment finding.

Figure 7.1: An example of a German health card where the KVNr is highlighted. Figure taken from [79]

7.3 Practical Assessment of the TK eHealth Record Application

The TK app was downloaded from the Apple App Store and installed on the non-jailbroken as well as on the jailbroken iOS device.

7.3.1 Information Gathering

The gathered information is documented in the checklist shown in Table 7.3.

The TK published the TK ePA app. The developer of the app is the IBM company, based in Germany.

7.3.2 Tool-Based Scan

The results of the tool-based scan with MobSF are summarized in Table 7.4.

The following paragraph describes the reason for the NOK status in the tool-based scan checklist of the TK app.

ID	Description	
I1 Evaluation of publicly available data		
I1.1	App Name:	TK-App
I1.2	App Store Link:	https://apps.apple.com/de/app/tk-app/id1163694230
I1.3	Analyzed Version:	3.7.0
I1.4	Application category:	Health & Fitness
I1.5	iOS compatibility:	iOS 12.0 or later
I1.6	Purpose of the application:	German ePA app of the TK.
I1.7	Collected data linked to the user's identity:	Financial Info, Contact Info, User Content, Identifiers, Usage Data, Sensitive Info
I2 Information about the application publisher		
I2.1	Publisher name:	Techniker Krankenkasse (TK)
I2.2	Publisher website:	https://www.tk.de
I2.3	Developer name:	IBM
I2.4	Developer website:	https://www.ega.de

Table 7.3: The TK app's information gathering checklist.

T3.3. MobSF warned that the binary does not have a stack canary value added to the stack.

ID	Description	Result
T1 Information about the system		
T1.1	Programming languages:	Swift
T1.2	App Identifier:	de.tk.tk-app
T1.3	App Permissions:	NFC reader, Calendars, Camera, Face ID, read/write Health Data, Photo Library
T2 App Transport Security		
T2.1	ATS is enabled. No insecure connections are configured.	OK
T3 IPA Binary Analysis		
T3.1	ARC is enabled.	OK
T3.2	The NX bit is set in the binary.	OK
T3.3	The stack canary value is added to the stack of the binary.	NOK
T3.4	PIE support is activated.	OK

Table 7.4: The TK app's tool-based scan checklist.

7.3.3 Vulnerability Assessment

After analyzing the tool-based scan results, the vulnerability assessment was performed utilizing the vulnerability assessment checklists as a guideline.

V1 Design and Privacy

The app presented an intro that described the purpose and functions of the TK ePA app when it was launched for the first time. The last slide of the intro provided two navigation buttons, as shown in Figure 7.2. The second button redirected to the TK website. If the first button was clicked, the privacy policy was shown and users needed to confirm the policy by ticking the confirmation box. The privacy policy contained all information required by the GDPR. Then, users were redirected and asked if they allow the app to analyze the usage behavior. Finally, the login screen, shown in Figure 7.3, where the insurance number and password could be entered.

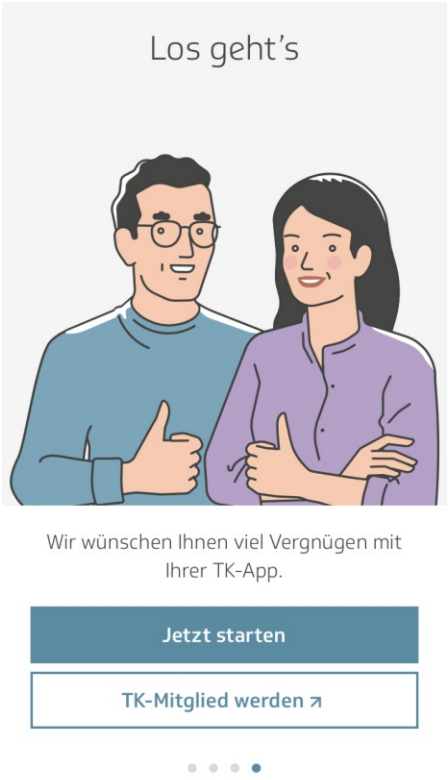

Figure 7.2: The screenshot shows the last slide of the TK app's intro.

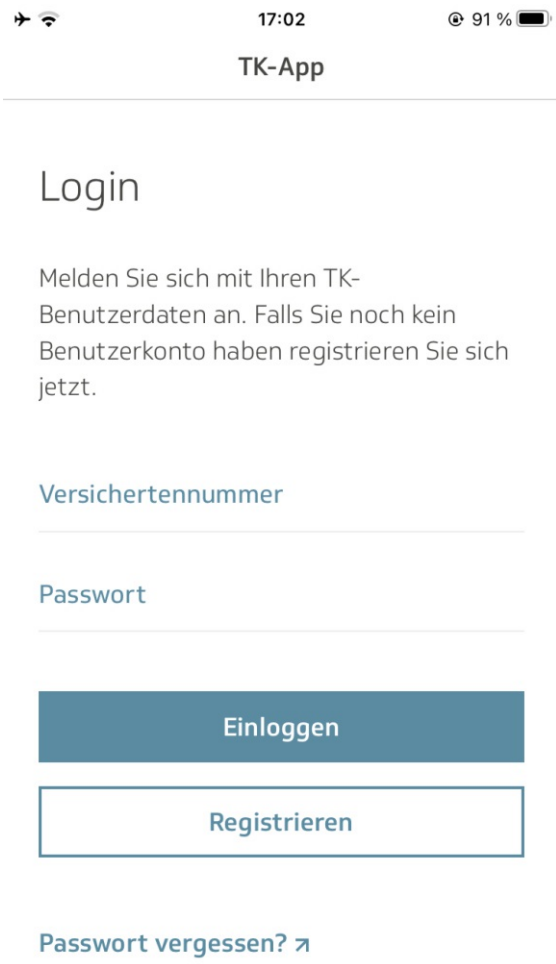

Figure 7.3: The screenshot shows the TK app's start screen.

The following paragraphs describe the reasons for the N/A and NOK states in the TK app's „Design and Privacy“ checklist shown in Table 7.5.

V1.5. The app did not recommend any safety measures that users can follow in order to increase the device's security.

V1.6. The app neither showed a warning nor prevented its usage when it was launched on a jailbroken device.

V1.7. The app's privacy policy did not describe how users can withdraw their given consent. The status is N/A because the app may provide a withdrawal functionality or explanation after login.

V1.8. The push notifications settings could not be tested since this may require an account, and hence, the status is N/A. When the app was launched, no pop-up asking for permission to send notifications showed up.

V1.9. The app did not explain how users can report security problems on the start screen. Moreover, it did not provide contact information such as an e-mail or a link to the TK website.

V2 Network Communication

The Burp tool was utilized to analyze the network communication in the TK app.

The app presented an error message as shown in Figure 7.4 when it was launched on a device that was configured to use the Burp proxy. Hence, it was not possible to perform actions that send requests to the server. The installation of the Burp CA certificate did not solve this issue. Burp reported „Remote host terminated the handshake“ in the event log, indicating that the app used certificate pinning.

It was possible to bypass the certificate pinning by using the Objection tool on the jailbroken device. As a result, no further error messages occurred.

The TK app contacted the following hosts during the vulnerability assessment, according to Burp:

- app-api.tk.de

The TK app's „Network Communication“ checklist is shown in Table 7.6.

V1 Design and Privacy		
ID	**Description**	**Status**
V1.1	The app discloses its primary purpose and the use of personal data before installation (e.g., in the description of the app store). The user is informed about this, at least when the app is first launched.	OK
V1.2	The app only collects and processes data that serves its primary purpose.	OK
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	OK
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	OK
V1.5	The app educates about security best practices the user should follow in using the app (e.g., using a passcode)	NOK
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	NOK
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	N/A
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	N/A
V1.9	Error messages do not contain sensitive data.	OK
V1.10	The application provides a simple option to report security problems.	NOK
V1.11	No sensitive data (e.g., passwords, pins) is exposed through the user interface.	OK

Table 7.5: The TK app's vulnerability assessment „Design and Privacy“ checklist.

V2 Network Communication		
ID	**Description**	**Status**
V2.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	OK
V2.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	OK
V2.3	The app supports certificate pinning.	OK

Table 7.6: The TK app's vulnerability assessment „Network Communication“ checklist.

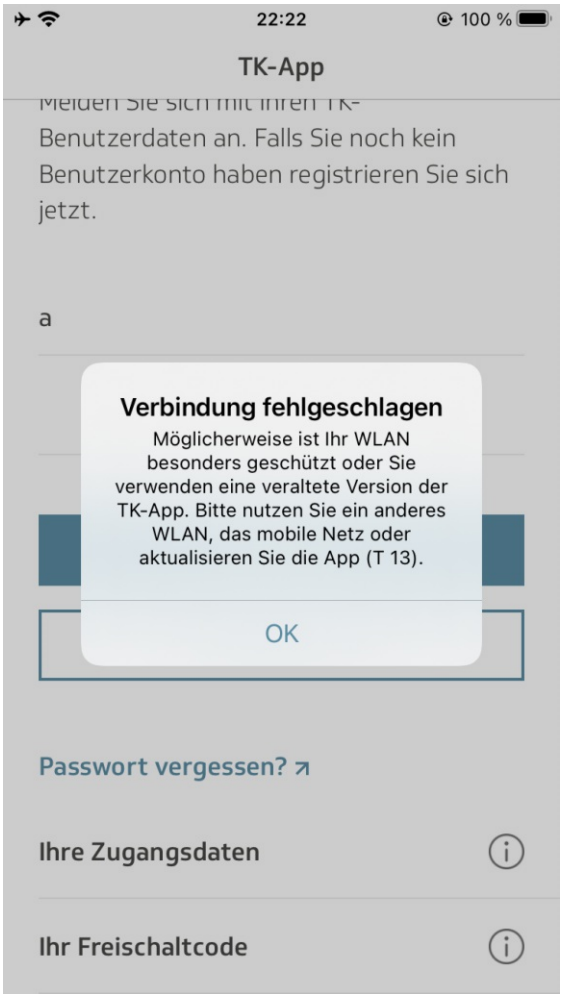

Figure 7.4: The TK app presented an error message when it was launched on a device that was configured to use the Burp proxy.

V3 Authentication Management

The steps from Section 7.3.3 were repeated to enable the analysis of the authentication management.

Then, the login was tested by entering the insurance number as well as password from the test data, and the intercepted login request was analyzed in Burp. The insurance number („fachschluessel“) and password stored as JSON in the body of the request were transmitted in cleartext. In this case, the server's response to a successful authentication could not be examined due to the lack of account.

The following paragraph describes the reason for the N/A states in the TK app's „Authentication Management“ checklist shown in Table 7.7.

V3.3, V3.4. These parts could not be tested because an account was required in order to test functionalities regarding account management.

V3 Authentication Management		
ID	**Description**	**Status**
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	OK
V3.2	A password policy exists and is enforced.	OK
V3.3	Users can change their passwords.	N/A
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	N/A

Table 7.7: The TK app's vulnerability assessment „Authentication Management“ checklist.

V4 Data Storage

The TK app prevented exposure of sensitive data by presenting its logo as shown in Figure 7.5 when it was moved to the background.

The following paragraphs describe the reasons for the N/A states in the TK app's „Data Storage“ checklist shown in Table 7.8.

V4.1, V4.2, V4.4. These parts could not be verified since they required an account. Without an account, it was impossible to add test data in the TK app that reflects sensitive user data.

7.3.4 Final Analysis

This section lists the four vulnerability assessment findings of the TK app. The findings are sorted chronologically.

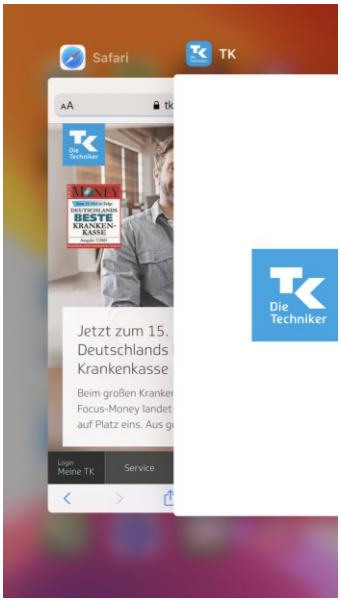

Figure 7.5: The TK app prevented exposure of sensitive data by presenting its logo when it was moved to the background.

V4 Data Storage		
ID	**Description**	**Status**
V4.1	The keychain is used to store sensitive data. They are sufficiently protected.	N/A
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	N/A
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	OK
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	N/A

Table 7.8: The TK app's vulnerability assessment „Data Storage“ checklist.

Vulnerabilities

The found vulnerability in the TK app is described in Table 7.9.

Miscellaneous

Table 7.10 summarizes the three miscellaneous vulnerability assessment findings of the TK app.

ID:	TK-001
Checklist ID:	T3.3
CVSS Score:	4.0 (Medium)
References:	CWE-121
Title:	Missing stack canary value
Description:	
The app might be vulnerable to stack smashing attacks because no stack canary value was added to the stack.	

Table 7.9: This vulnerability concerning the missing stack canary value was found in checklist item T3.3 in the TK app.

ID:	TK-002
Checklist ID:	V1.5
Title:	Missing safety notes for users
Description:	
The app does not recommend any safety measures that users can follow in order to increase the device's security.	
ID:	TK-003
Checklist ID:	V1.6
Title:	Missing jailbreak detection
Description:	
The app neither shows a warning nor prevents its usage when it is launched on a jailbroken device.	
ID:	TK-004
Checklist ID:	V1.10
Title:	Lacking information on how to report security problems
Description:	
The app does not explain how users can report security issues.	

Table 7.10: The miscellaneous „Design and Privacy“ findings of the TK app's vulnerability assessment.

7.4 Practical Assessment of the Ada eHealth Application

The Ada app was downloaded from the Apple App Store and installed on the non-jailbroken as well as on the jailbroken iOS device.

7.4.1 Information Gathering

The gathered information is documented in the checklist shown in Table 7.11.

Ada published and developed the app. The company is based in Germany, according to its website.

ID	Description	
I1 Evaluation of publicly available data		
I1.1	App Name:	Ada
I1.2	App Store Link:	https://apps.apple.com/de/app/ada-check-your-health/id1099986434
I1.3	Analyzed Version:	3.11.0
I1.4	Application category:	Medical
I1.5	iOS compatibility:	iOS 9.3 or later
I1.6	Purpose of the application:	Checking symptoms and discovering what might be causing them.
I1.7	Collected data linked to the user's identity:	Health & Fitness, Location, Contact Info, User Content, Usage Data, Diagnostics, Identifiers, Sensitive Info, Search History
I2 Information about the application publisher		
I2.1	Publisher name:	Ada Health GmbH
I2.2	Publisher website:	https://ada.com
I2.3	Developer name:	Ada Health GmbH
I2.4	Developer website:	https://ada.com

Table 7.11: The Ada app's information gathering checklist.

7.4.2 Tool-Based Scan

The results of the tool-based scan with MobSF are summarized in Table 7.12.

The following paragraph describes the reason for the NOK status in the Ada app's tool-based scan checklist.

T2.1. MobSF reported the following ATS issues:

- **NSAllowsArbitraryLoads is allowed:** HTTP connections are allowed and additional security checks such as minimum TLS version are not provided.

- **NSExceptionAllowsInsecureHTTPLoads is allowed:** Insecure communication with HTTP to localhost is allowed that might loosen the server trust evaluation requirements for HTTPS connections to the domain.
- **NSExceptionRequiresForwardSecrecy is allowed:** The boolean value is set to *false* for localhost.

ID	Description	Result
T1 Information about the system		
T1.1	Programming languages:	Swift
T1.2	App Identifier:	net.medx.Ada.production
T2 App Transport Security		
T2.1	ATS is enabled. No insecure connections are configured.	NOK
T3 IPA Binary Analysis		
T3.1	ARC is enabled.	OK
T3.2	The NX bit is set in the binary.	OK
T3.3	The stack canary value is added to the stack of the binary.	OK
T3.4	PIE support is activated.	OK

Table 7.12: The Ada app's tool-based scan checklist.

7.4.3 Vulnerability Assessment

After analyzing the tool-based scan results, the vulnerability assessment was performed utilizing the vulnerability assessment checklists as a guideline.

V1 Design and Privacy

The app presented a screen with a slide and two navigation buttons for login and account creation when it is launched for the first time, as shown in Figure 7.6. The slide explained the purpose of the Ada app. When users clicked the „Create your account“ button, they were redirected to a screen that required them to read the privacy policy and confirm the terms and conditions. The privacy policy contained all information required by the GDPR and described how users can withdraw their consent and how this will affect the behavior of the app.

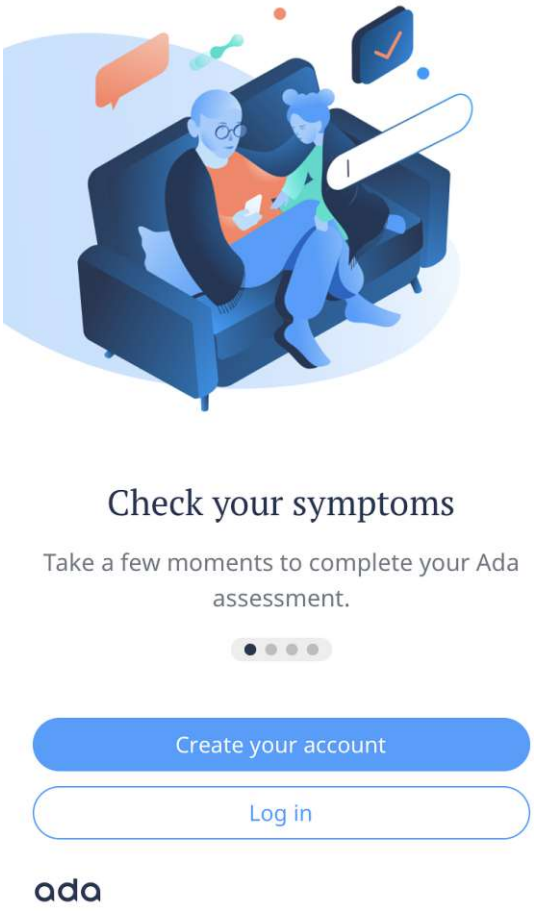

Figure 7.6: This screenshot shows the intro that is presented when the Ada app is launched for the first time.

Then, the user needed to enter an e-mail address and password. Afterward, the app verified the e-mail address by sending a confirmation mail. After the user confirmed the e-mail address, the Ada app asked for consent to use personal health data, app activity, and performance data to improve the app. Furthermore, Ada asked if the user wants to participate in health research projects. Then, the user was required to enter the following information to create the user's health background used for health assessments:

- Name
- Birthday
- Gender
- have you ever been diagnosed with high blood pressure? (optional)
- Do you have diabetes? (optional)
- Are you a smoker? (optional)
- Are you pregnant? (optional)

The primary purpose of the Ada app was to check symptoms and discover what might be causing them. Therefore, users could start a health assessment as shown in Figure 7.7. This assessment included several questions regarding symptoms. In the end, possible causes of these symptoms were listed.

Ada provided a navigation point to give feedback to the developers. It redirected to the mail program of the device.

The following paragraphs describe the reasons for the N/A and NOK states in the Ada app's „Design and Privacy“ checklist shown in Table 7.13.

V1.5. The app's terms and conditions recommended using a „strong“ user password but provided no further safety notes.

V1.6. The app neither showed a warning nor prevented its usage when it was launched on a jailbroken device.

V1.8. The push notification settings could not be tested since the app might not offer this functionality, and hence, the status is N/A. When the app was launched, no pop-up asking for permission to send notifications showed up.

V2 Network Communication

The Burp tool was utilized to analyze the network communication in the Ada app.

The app presented an error message as shown in Figure 7.8 when it was launched on a device that was configured to use the Burp proxy. Hence, it was not possible to perform actions that send requests to the server. However, it was possible to use the app and intercept the traffic after the Burp CA certificate was installed on the iOS device.

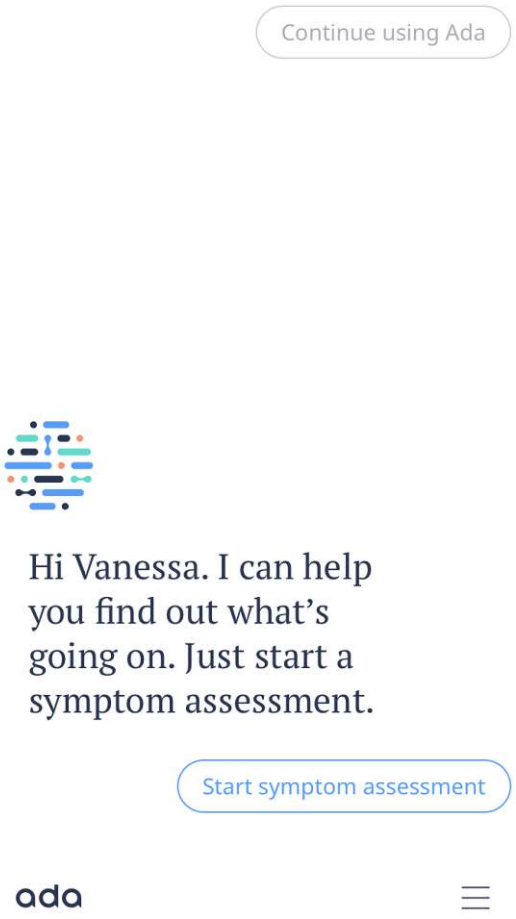

Figure 7.7: This screenshot shows the start screen of the Ada app, where users can start a symptom assessment.

V1 Design and Privacy		
ID	**Description**	**Status**
V1.1	The app discloses its primary purpose and the use of personal data before installation (e.g., in the description of the app store). The user is informed about this, at least when the app is first launched.	OK
V1.2	The app only collects and processes data that serves its primary purpose.	OK
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	OK
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	OK
V1.5	The app educates about security best practices the user should follow in using the app (e.g., using a passcode)	NOK
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	NOK
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	OK
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	N/A
V1.9	Error messages do not contain sensitive data.	OK
V1.10	The application provides a simple option to report security problems.	OK
V1.11	No sensitive data (e.g., passwords, pins) is exposed through the user interface.	OK

Table 7.13: The Ada app's vulnerability assessment „Design and Privacy“ checklist.

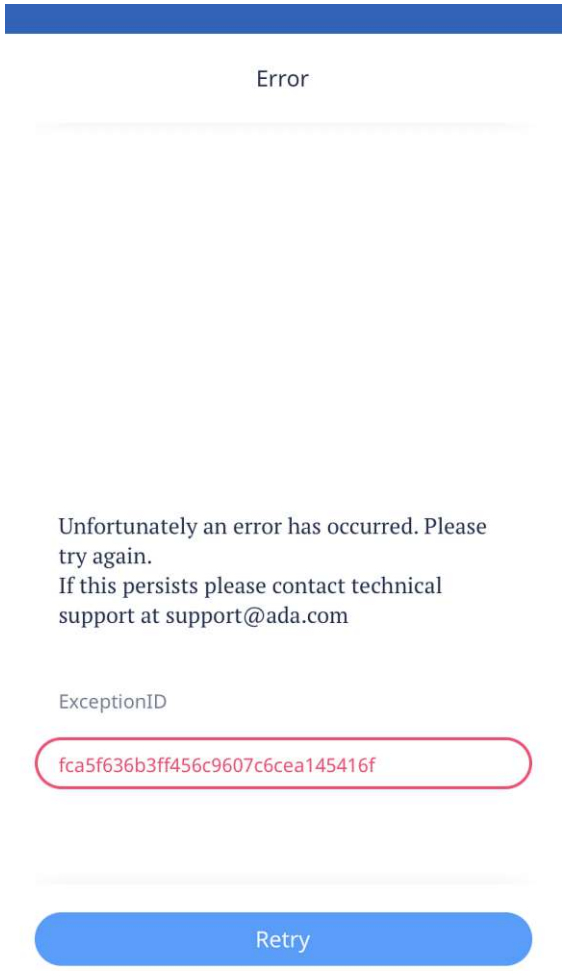

Figure 7.8: The Ada app presented an error when the device was configured to use the Burp proxy.

The following hosts were contacted during the vulnerability assessment of the Ada according to Burp:

- ada-bff.consumer-prod-eu.prod.ada.com
- furiosa-zuul.client-prod-eu.prod.ada.com
- sentry.io
- ada.com
- adahealth.cdn.prismic.io
- prismic-io.s3.amazonaws.com

The following paragraph describes the reason for the NOK state in the Ada app's „Network Communication“ checklist shown in Table 7.14.

V2.3. It was possible to use the app and intercept the traffic after the Burp CA certificate was installed on the iOS device, indicating that the app did not use certificate pinning.

V3 Network Communication		
ID	**Description**	**Status**
V3.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	OK
V3.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	OK
V2.3	The app supports certificate pinning.	NOK

Table 7.14: The Ada app's vulnerability assessment „Network Communication“ checklist.

V3 Authentication Management

The steps from Section 7.4.3 were repeated to enable the analysis of the authentication management.

Then, the login was tested. Therefore, the e-mail and password test data were entered. Afterward, the intercepted request was analyzed in Burp. The e-mail („identity“) and password stored in the request's body were transmitted in cleartext. In addition, the server returned, among others, an RS256 JWT [74] access token.

The Ada app had a password policy that is enforced. The app's terms and conditions recommended using „strong“ passwords consisting of a combination of numbers, both upper and lower case letters and symbols. However, the app only checked if the password length is at least eight chars. Therefore, passwords such as `12345678` were allowed. Moreover, the registration and password change required a password with at least eight chars length, but the login expected at least one char.

The following paragraph describes the reason for the NOK status in the Ada app's „Authentication Management“ checklist shown in Table 7.15.

V3.4. The app had no security mechanisms such as delays in subsequent login attempts or temporary lock after entering the wrong password ten times.

V3 Authentication Management		
ID	**Description**	**Status**
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	OK
V3.2	A password policy exists and is enforced.	OK
V3.3	Users can change their passwords.	OK
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	NOK

Table 7.15: The Ada app's vulnerability assessment „Authentication Management“ checklist.

V4 Data Storage

The following paragraphs describe the reasons for the NOK and N/A states in the Ada app's „Data Storage“ checklist shown in Table 7.16.

V4.2. The command `grep MyPassword1234 -r .` was executed in the Ada app's „Data“ directory. The `Library/Caches/net.medx.Ada.production/Cache.db-wal` file contained the password as plaintext.

V4.3. When the Ada app was moved to the background, its content was still visible, as shown in Figure 7.9. Moreover, the snapshots were locally stored in the *Snapshots* directory.

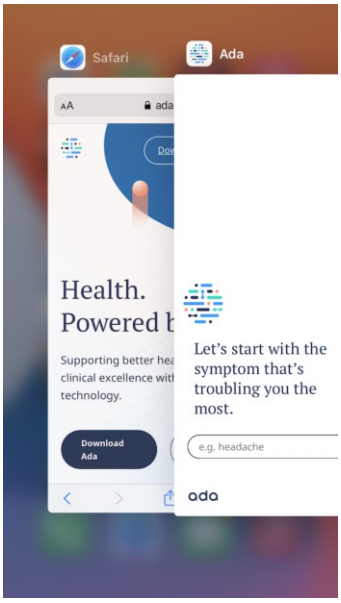

Figure 7.9: When the Ada app was moved to the background, its content was still visible.

V4 Data Storage		
ID	**Description**	**Status**
V4.1	The keychain is used to store sensitive data. They are sufficiently protected.	OK
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	NOK
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	NOK
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	OK

Table 7.16: The Ada app's vulnerability assessment „Data Storage“ checklist.

7.4.4 Final Analysis

This section lists the ten findings of the Ada app's vulnerability assessment. The findings are sorted chronologically.

Vulnerabilities

The three tables 7.17, 7.19 and 7.19 describe the three found vulnerabilities in the Ada app.

ID:	ADA-008
Checklist ID:	V2.3
CVSS Score:	4.0 (Medium)
References:	CWE-295
Title:	Missing certificate pinning
Description:	
The app does not support certificate pinning and therefore, might be vulnerable to man-in-the-middle attacks.	

Table 7.17: This Ada app's vulnerability concerning the missing certificate pinning was found in checklist item V2.3.

ID:	ADA-009
Checklist ID:	V4.2
CVSS Score:	5.5 (Medium)
References:	CWE-312, CWE-522, CWE-524
Title:	Credentials in NSURLRequest cache
Description:	
Login credentials such as the e-mail and password are cached plaintext by NSURLRequest.	

Table 7.18: This Ada app's vulnerability concerning the credentials in the NSURLRequest cache was found in checklist item V4.2.

Miscellaneous

The tables 7.20, 7.21, 7.22 summarize the seven miscellaneous Ada app's vulnerability assessment findings.

ID:	ADA-010
Checklist ID:	V4.3
CVSS Score:	4.0 (Medium)
References:	CWE-200
Title:	Information disclosure in auto-generated snapshots
Description:	
The app's *Snapshots* directory in the device's local storage contains snapshots of its screens that are auto-generated by iOS. These snapshots might expose sensitive and confidential user data shown in the app.	

Table 7.19: This Ada app's vulnerability concerning information disclosure in auto-generated snapshots was found in checklist item V4.3.

ID:	ADA-001
Checklist ID:	T2.1
Title:	ATS NSExceptionRequiresForwardSecrecy is set to NO for localhost
Description:	
The app has NSExceptionRequiresForwardSecrecy set to *NO* for localhost and hence, overrides the requirement that the server must support Perfect Forward Secrecy (PFS) for localhost.	
ID:	ADA-002
Checklist ID:	T2.1
Title:	ATS NSAllowsArbitraryLoads is allowed
Description:	
The app allows ATS NSAllowsArbitraryLoads and therefore, insecure HTTP connections can be established.	
ID:	ADA-003
Checklist ID:	T2.1
Title:	NSExceptionAllowsInsecureHTTPLoads is allowed
Description:	
The app has NSExceptionAllowsInsecureHTTPLoads set to *YES* for localhost and hence, insecure HTTP loads to localhost are allowed.	

Table 7.20: The miscellaneous tool-based scan findings of the Ada app's vulnerability assessment.

ID:	ADA-004
Checklist ID:	V1.5
Title:	Missing safety notes for users
Description:	
The app does not recommend any safety measures that users can follow in order to increase the device's security.	
ID:	ADA-005
Checklist ID:	V1.6
Title:	Missing jailbreak detection
Description:	
The app neither shows a warning nor prevents its usage when it is launched on a jailbroken device.	

Table 7.21: The miscellaneous „Design and Privacy“ findings of the Ada app's vulnerability assessment.

ID:	ADA-006
Checklist ID:	V3.2
Title:	Inconsistent min. password char length in input fields
Description:	
The app's registration and password change masks require a password with eight chars length. However, the login already accepts one char.	
ID:	ADA-007
Checklist ID:	V3.4
Title:	Missing brute-force login protection
Description:	
The app may not have a security mechanism for the login that prevents brute-force attacks.	

Table 7.22: The miscellaneous „Authentication Management“ findings of the Ada app's vulnerability assessment.

7.5 Practical Assessment of the mySugr eHealth Application

The mySugr app was downloaded from the Apple App Store and installed on the non-jailbroken as well as on the jailbroken iOS device.

7.5.1 Information Gathering

The gathered information is documented in the checklist shown in Table 7.23.

MySugr is published and developed the app. The company is based in Austria, according to its website.

ID	Description	
I1 Evaluation of publicly available data		
I1.1	App Name:	mySugr
I1.2	App Store Link:	https://apps.apple.com/de/app/mysugr-diabetes-tagebuch/id516509211
I1.3	Analyzed Version:	3.83.2
I1.4	Application category:	Medical
I1.5	iOS compatibility:	iOS 13.0 or later
I1.6	Purpose of the application:	Diabetes logbook.
I1.7	Collected data linked to the user's identity:	Health & Fitness, Location, Contact Info, User Content, Usage Data, Diagnostics, Identifiers
I2 Information about the application publisher		
I2.1	Publisher name:	mySugr
I2.2	Publisher website:	https://mysugr.com
I2.3	Developer name:	mySugr
I2.4	Developer website:	https://mysugr.com

Table 7.23: The mySugr app's information gathering checklist.

7.5.2 Tool-Based Scan

The results of the tool-based scan with MobSF are summarized in Table 7.24.

The following paragraph describes the reason for the NOK status in the tool-based scan checklist of the mySugr app.

T2.1. MobSF found the following ATS issues:

- **NSExceptionRequiresForwardSecrecy is allowed:** The boolean value is set to *false* for the following domains:
  - s3-eu-west-1.amazonaws.com
  - cloudfront.net
  - cdnjs.cloudflare.com
  - p5.zdassets.com
  - support.mysugr.com
  - mysugr.com
  - assets.mysugr.com
  - mysugr.zendesk.com
  - mysugr-elements.com

ID	Description	Result
T1 Information about the system		
T1.1	Programming languages:	Objective-C
T1.2	App Identifier:	com.mysugr.companion.mySugr
T1.3	App Permissions:	NFC reader, Bluetooth Interface, Camera, read/write Health Data, Location, Photo Library
T2 App Transport Security		
T2.1	ATS is enabled. No insecure connections are configured.	NOK
T3 IPA Binary Analysis		
T3.1	ARC is enabled.	OK
T3.2	The NX bit is set in the binary.	OK
T3.3	The stack canary value is added to the stack of the binary.	OK
T3.4	PIE support is activated.	OK

Table 7.24: The tool-based scan checklist of the mySugr app.

7.5.3 Vulnerability Assessment

After analyzing the tool-based scan results, the vulnerability assessment was performed utilizing the vulnerability assessment checklists as a guideline.

V1 Design and Privacy

The mySugr app asked for an e-mail address when it was is launched. In addition, a link was provided where it was explained why registration is needed.

According to mySugr, registration is needed to ensure data security, guaranteed backups, and privacy.

The mySugr user manual that was accessible after login recommended not to use the app on a jailbroken device and update the app if updates are available.

The following paragraph describes the reason for the NOK status in the mySugr app's „Design and Privacy" checklist shown in Table 7.25.

V1.6. The app neither showed a warning nor prevented its usage when it was launched on a jailbroken device.

V1.10. The app did not describe how security problems can be reported. However, there was a feedback form that could be used for feedback and support.

V2 Network Communication

The Burp tool was utilized to analyze the network communication in the mySugr app.

The app presented an error message as shown in Figure 7.10 when it was launched on a device that was configured to use the Burp proxy. Hence, it was not possible to perform actions that send requests to the server. The installation of the Burp CA certificate did not solve this issue. Burp reported „Remote host terminated the handshake" in the event log, indicating that the app used certificate pinning.

It was possible to bypass the certificate pinning by using the Objection tool on the jailbroken device. As a result, no further error messages occurred.

The mySugr app contacted the following hosts during the vulnerability assessment, according to Burp:

- sdk.fra-01.braze.eu
- countly-api.mysugr.com
- assets-nocache.mysugr.com
- assets.mysugr.com
- eu-prod-api.mysugr.com
- support.mysugr.com
- mysugr.zendesk.com
- p18.zdassets.com

- static.zdassets.com
- theme.zdassets.com
- ajax.googleapis.com
- app.adjust.com
- app.adjust.net.in

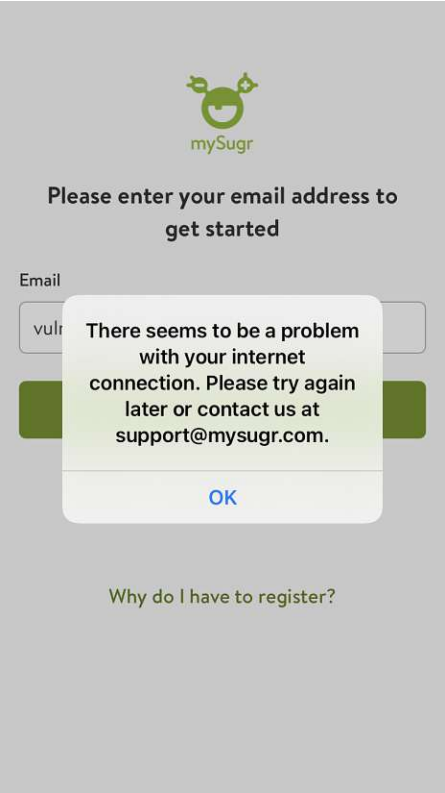

Figure 7.10: The mySugr app presented an error message when it was launched on a device that was configured to use the Burp proxy.

V1 Design and Privacy		
ID	**Description**	**Status**
V1.1	The app discloses its primary purpose and the use of personal data before installation (e.g., in the description of the app store). The user is informed about this, at least when the app is first launched.	OK
V1.2	The app only collects and processes data that serves its primary purpose.	OK
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	OK
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	OK
V1.5	The app educates about security best practices the user should follow in using the app (e.g., using a passcode)	OK
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	NOK
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	OK
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	OK
V1.9	Error messages do not contain sensitive data.	OK
V1.10	The application provides a simple option to report security problems.	NOK
V1.11	No sensitive data (e.g., passwords, pins) is exposed through the user interface.	OK

Table 7.25: The mySugr app's vulnerability assessment „Design and Privacy“ checklist.

The mySugr app's „Network Communication“ checklist is shown in Table 7.26.

V2 Network Communication		
ID	**Description**	**Status**
V2.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	OK
V2.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	OK
V2.3	The app supports certificate pinning.	OK

Table 7.26: The mySugr app's vulnerability assessment „Network Communication“ checklist of the.

V3 Authentication Management

The steps from Section 7.6.3 were repeated to enable the analysis of the authentication management.

Then, the login was tested. Therefore, the e-mail and password test data were entered. Afterward, the intercepted request was analyzed in Burp. The e-mail and password were transmitted using basic access authentication [61]. The request contained a header field *Authorization* where the e-mail and password were Base64 encoded and joined by a `:`. The server returned a JSON containing user data including the ID, e-mail address, given consent, and granted authorities.

The mySugr app had a password policy that was enforced. However, the app only checked if the password length is at least eight chars. Therefore, passwords such as `12345678` were allowed. Moreover, the registration and password change required a password with at least eight chars length, but the login expected at least six chars.

The following paragraph describes the reason for the NOK status in the mySugr app's „Authentication Management“ checklist shown in Table 7.27.

V3.4. The app had no security mechanisms such as delays in subsequent login attempts or temporary lock after entering the wrong password ten times.

V4 Data Storage

The following paragraph describes the reason for the NOK status in the mySugr app's „Data Storage“ checklist shown in Table 7.28.

V4.3. When the mySugr app was moved to the background, its content was still visible, as shown in Figure 7.11. Moreover, the snapshots were locally stored in the *Snapshots* directory.

V3 Authentication Management		
ID	**Description**	**Status**
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	OK
V3.2	A password policy exists and is enforced.	OK
V3.3	Users can change their passwords.	OK
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	NOK

Table 7.27: The mySugr app's vulnerability assessment „Authentication Management" checklist.

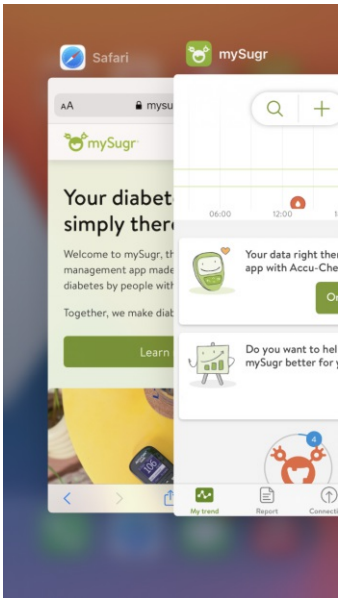

Figure 7.11: When the mySugr app was moved to the background, its content was still visible.

7.5.4 Final Analysis

This section lists the 11 findings of the vulnerability assessment. The findings are sorted chronologically.

Vulnerabilities

Table 7.29 describes the found vulnerability in the mySugr app.

V4 Data Storage		
ID	**Description**	**Status**
V4.1	The keychain is used to store sensitive data. They are sufficiently protected.	OK
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	OK
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	NOK
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	OK

Table 7.28: The mySugr app's vulnerability assessment „Data Storage" checklist.

ID:	MYSUGR-006
Checklist ID:	V2.3
CVSS Score:	4.0 (Medium)
References:	CWE-200
Title:	Information disclosure in auto-generated snapshots
Description:	
The app's *Snapshots* directory in the device's local storage contains snapshots of its screens that are auto-generated by iOS. These snapshots might expose sensitive and confidential user data shown in the app.	

Table 7.29: This vulnerability concerning information disclosure in auto-generated snapshots was found in checklist item V4.3 in the mySugr app.

Miscellaneous

The tables 7.30, 7.31 and 7.32 summarize the six miscellaneous vulnerability assessment findings of the mySugr app.

ID:	MYSUGR-001
Checklist ID:	T2.1
Title:	NSExceptionRequiresForwardSecrecy is set to NO for several domains
Description:	
The app has NSExceptionRequiresForwardSecrecy set to *NO* for several domains and hence, overrides the requirement that the server must support PFS for them.	

Table 7.30: The miscellaneous tool-based scan findings of the mySugr app's vulnerability assessment.

ID:	MYSUGR-002
Checklist ID:	V1.6
Title:	Missing jailbreak detection
Description:	
The app neither shows a warning nor prevents its usage when it is launched on a jailbroken device.	
ID:	MYSUGR-003
Checklist ID:	V1.8
Title:	Lacking information on how to report security problems
Description:	
The app does not explain how users can report security issues.	

Table 7.31: The miscellaneous „Design and Privacy“ findings of the mySugr app's vulnerability assessment.

ID:	MYSUGR-004
Checklist ID:	V3.2
Title:	Inconsistent min. password char length in input fields
Description:	
The registration and password change require a password with eight chars length. However, the login already accepts six chars.	
ID:	MYSUGR-005
Checklist ID:	V3.4
Title:	Missing brute-force login protection
Description:	
The app may not have a security mechanism for the login that prevents brute-force attacks.	

Table 7.32: The miscellaneous „Authentication Management“ findings of the mySugr app's vulnerability assessment.

7.6 Practical Assessment of the Femometer eHealth Application

The Femometer app was downloaded from the Apple App Store and installed on the non-jailbroken as well as on the jailbroken iOS device.

7.6.1 Information Gathering

The gathered information is documented in the checklist shown in Table 7.33.

The name of the app publisher is „Bongmi Global Group“. The Femometer app is advertised on their website. According to the website, the company is based in China.

ID	Description	
I1 Evaluation of publicly available data		
I1.1	App Name:	Femometer
I1.2	App Store Link:	https://apps.apple.com/de/app/femometer-perioden-kalender/id1529565125
I1.3	Analyzed Version:	4.6.1
I1.4	Application category:	Health & Fitness
I1.5	iOS compatibility:	iOS 9.0 or later
I1.6	Purpose of the application:	Smart period tracker and ovulation calendar.
I1.7	Collected data linked to the user's identity:	Health & Fitness, Contact Info, User Content, Usage Data, Diagnostics, Identifiers, Sensitive Info
I2 Information about the application publisher		
I2.1	Publisher name:	Bongmi Global Group
I2.2	Publisher website:	https://www.femometer.com/app/femometer-app.html
I2.3	Developer name:	Hangzhou Bangtang Network Technology Co., Ltd
I2.4	Developer website:	https://www.bongmi.com

Table 7.33: The Femometer app's information gathering checklist.

7.6.2 Tool-Based Scan

The results of the tool-based scan with MobSF are summarized in Table 7.34.

The following paragraph describes the reason for the NOK status in the tool-based scan checklist of the Femometer app.

T2.1. MobSF found the following ATS issue:

- **NSAllowsArbitraryLoads is allowed:** The ATS restrictions are disabled for all network connections. Hence, HTTP connections are allowed and additional security checks such as minimum TLS version are not provided.

ID	Description	Result
T1 Information about the system		
T1.1	Programming languages:	Objective-C
T1.2	App Identifier:	com.btang.femometer.eu
T1.3	App Permissions:	Bluetooth Interface, Camera, Face ID, read/write Health Data, Location, Microphone, Photo Library
T2 App Transport Security		
T2.1	ATS is enabled. No insecure connections are configured.	NOK
T3 IPA Binary Analysis		
T3.1	ARC is enabled.	OK
T3.2	The NX bit is set in the binary.	OK
T3.3	The stack canary value is added to the stack of the binary.	OK
T3.4	PIE support is activated.	OK

Table 7.34: The Femometer app's tool-based scan checklist.

7.6.3 Vulnerability Assessment

After analyzing the tool-based scan results, the vulnerability assessment was performed utilizing the vulnerability assessment checklists as a guideline.

V1 Design and Privacy

When the Femometer app was launched, a slide was presented that described its purpose and functions, and various login methods were provided as shown in Figure 7.12.

After login, information about the current cycle was displayed. In addition, the app utilized user data for predictions and calculations regarding the menstrual cycle.

There was a „Contact Customer Service“ button in the „Support and Help“ section that showed the e-mail of the support team. In the „Support and Help“ section, the app described that logs could be uploaded to support their developers in identifying an issue. Two different sections enabled the upload of logs in the app, namely the „Support and Help“ and the „APP Settings“ section. The log was uploaded immediately

without confirmation when „Upload Logs“ was clicked and showed a message when it was successfully uploaded.

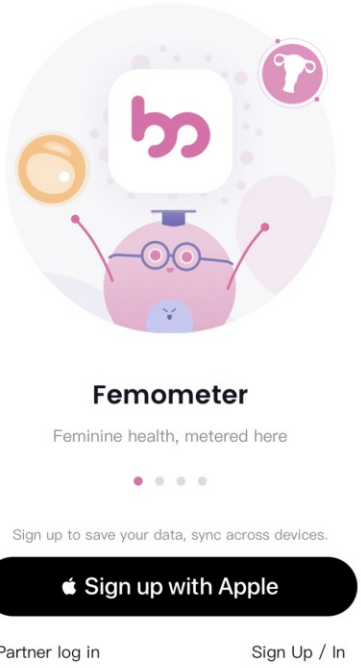

Figure 7.12: The screenshot shows the start screen of the Femometer app.

The following paragraphs describe the reasons for the NOK states in the Femometer app's „Design and Privacy“ checklist shown in Table 7.35.

V1.1. The App Store provided a link to the Femometer's privacy policy where it was stated how the app collects, uses, and shares the user data. However, when the app was launched, a login screen was shown, but no other information or links to, for example, the privacy policy. Furthermore, the signup page did not provide information about the privacy policy or how the data will be used or processed. The privacy policy was accessible after login but the provided privacy policy in the Femometer app differs from the one linked in the App Store.

V1.3. It was not described why access to the microphone, location, and Face ID was needed.

V1.4. The app did not obtain the user's consent for collecting and processing PII.

V1.5. The app did not provide safety notes. However, the privacy policy pointed out that the app has a safety lock feature that enables setting a code to protect the app. According to the privacy policy, this feature automatically logs the user out after several incorrect code input attempts.

V1.6. The app neither showed a warning nor prevented its usage when it was launched on a jailbroken device.

V1.7. The app's privacy policy stated that users could revoke their consent at any time, and can request that Femometer deletes their account. However, the policy did not explain where and which consent users can revoke.

V1 Design and Privacy		
ID	**Description**	**Status**
V1.1	The app discloses its primary purpose and the use of personal data before installation (e.g., in the description of the app store). The user is informed about this, at least when the app is first launched.	NOK
V1.2	The app only collects and processes data that serves its primary purpose.	OK
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	NOK
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	NOK
V1.5	The app educates about security best practices the user should follow in using the app (e.g., using a passcode)	NOK
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	NOK
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	NOK
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	OK
V1.9	Error messages do not contain sensitive data.	OK
V1.10	The application provides a simple option to report security problems.	OK
V1.11	No sensitive data (e.g., passwords, pins) is exposed through the user interface.	OK

Table 7.35: The Femometer app's vulnerability assessment „Design and Privacy“ checklist.

V2 Network Communication

The Burp tool was utilized to analyze the network communication in the Femometer app.

It was impossible to perform actions that send requests to the server when the device was configured to use the Burp proxy. On the login screen, no error messages were shown. When a user tried to login, nothing happened when the „Sign In“ button was clicked. It was possible to use the app and intercept the traffic after the Burp CA certificate was installed on the iOS device.

The Femometer app contacted the following hosts during the vulnerability assessment, according to Burp:

- api.femometer.com
- api-eu.femometer.com
- sa.bongmi.com
- inapps.appsflyer.com
- launches.appsflyer.com
- api.xmpush.xiaomi.com
- stats.jpush.cn
- ali-stats.jpush.cn
- gd-stats.jpush.cn
- ios.bugly.qq.com
- config.jpush.cn
- kegelcourse.vtio.cn
- lollypop-log.s3-us-west-1.amazonaws.com

The following paragraph describes the reason for the NOK state in the Femometer app's „Network Communication“ checklist shown in Table 7.36.

V2.3. It was possible to use the app and intercept the traffic after the Burp CA certificate was installed on the iOS device, indicating that the app did not use certificate pinning.

V3 Authentication Management

The steps from Section 7.6.3 were repeated to enable the analysis of the authentication management.

Then, the login was tested. Therefore, the e-mail and password test data were entered. Afterward, the intercepted request was analyzed in Burp. The e-mail and password were

V2 Network Communication		
ID	**Description**	**Status**
V2.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	OK
V2.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	OK
V2.3	The app supports certificate pinning.	NOK

Table 7.36: The Femometer app's vulnerability assessment „Network Communication" checklist.

transmitted in a JSON. The password was hashed with MD5. The server returned a JSON with user data, including, among others, gender, nickname, and e-mail.

The Femometer app had a password policy that was enforced. The registration and password change required a password with eight chars length as well as a mix of letters, numbers, and symbols. However, the login already accepted one char.

The following paragraph describes the reason for the NOK status in the Femometer app's „Authentication Management" checklist shown in Table 7.37.

V3.4. The app had no security mechanisms regarding the login process such as delays in subsequent login attempts or temporary lock after entering the wrong password ten times.

V3 Authentication Management		
ID	**Description**	**Status**
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	OK
V3.2	A password policy exists and is enforced.	OK
V3.3	Users can change their passwords.	OK
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	NOK

Table 7.37: The Femometer app's vulnerability assessment „Authentication Management" checklist.

V4 Data Storage

The following paragraphs describe the reasons for the NOK states in the Femometer app's „Data Storage" checklist shown in Table 7.38.

V4 Data Storage		
ID	**Description**	**Status**
V4.1	The keychain is used to store sensitive data. They are sufficiently protected.	OK
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	NOK
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	NOK
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	NOK

Table 7.38: The Femometer app's vulnerability assessment „Data Storage“ checklist.

V4.2. The aults of the app contained the user's e-mail address that was used for the login and several tokens, such as for „AppsFlyer“ and „MiPush“. Moreover, the NSUserDefaults in the Objection analysis revealed that the passcode of the Femometer's safety lock feature, which was hashed with MD5, was stored there. The `grep` command with the password as well as the MD5 hashed password from the test data was executed in the „Data“ folder of the app.

The following three files contained the MD5 hashed password:

- `Documents/thermometer.db`
- `Library/caches/com.btang.femometer.eu/Cache.db`
- `Library/Preferences/group.com.btong.femometer.eu.plist`

V4.3. When the Femometer app was moved to the background, its content was still visible, as shown in Figure 7.13. Moreover, the snapshots were locally stored in the *Snapshots* directory.

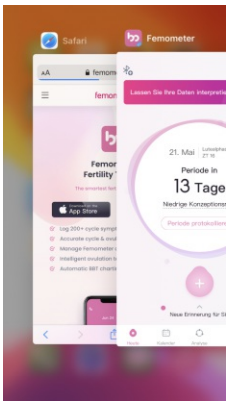

Figure 7.13: When the Femometer app was moved to the background, its content was still visible.

V4.4. The passwords and passcodes found in V4.2 were hashed with MD5.

7.6.4 Final Analysis

This section lists the 13 findings of the vulnerability assessment. The findings are sorted chronologically.

Vulnerabilities

The four tables 7.39, 7.40, 7.41 and 7.42 describe the four vulnerabilities found in the Femometer app.

ID:	FEMOMETER-010
Checklist ID:	V2.3
CVSS Score:	4.0 (Medium)
References:	CWE-295
Title:	Missing certificate pinning
Description:	
The app does not support certificate pinning and therefore, might be vulnerable to man-in-the-middle attacks.	

Table 7.39: This Femometer app's vulnerability concerning the missing certificate pinning was found in checklist item V2.3.

ID:	FEMOMETER-011
Checklist ID:	V4.2
CVSS Score:	5.5 (Medium)
References:	CWE-313, CWE-359
Title:	Storage of credentials in NSUserDefaults
Description:	
The app stores the user's e-mail as well as passwords and passcodes hashed with MD5 in the device's NSUserDefaults local storage.	

Table 7.40: This vulnerability concerning the storage of credentials in NSUserDefaults was found in checklist item V4.2 in the Femometer app.

Miscellaneous

The tables 7.43, 7.44 and 7.45 summarize the nine miscellaneous vulnerability assessment findings of the Femometer app.

ID:	FEMOMETER-012
Checklist ID:	V4.3
CVSS Score:	4.0 (Medium)
References:	CWE-200
Title:	Information disclosure in auto-generated snapshots
Description:	
The app's *Snapshots* directory in the device's local storage contains snapshots of its screens that are auto-generated by iOS. These snapshots might expose sensitive and confidential user data shown in the app.	

Table 7.41: This vulnerability concerning information disclosure in auto-generated snapshots was found in checklist item V4.3 in the Femometer app.

ID:	FEMOMETER-013
Checklist ID:	V4.4
CVSS Score:	4.0 (Medium)
References:	CWE-916
Title:	Hashing passwords and passcodes with MD5
Description:	
The app stores user passwords and passcodes as MD5 hashes in the local storage of the device.	

Table 7.42: This Femometer app's vulnerability concerning hashing passwords and passcodes with MD5 was found in checklist item V4.4.

ID:	FEMOMETER-001
Checklist ID:	T2.1
Title:	ATS NSAllowsArbitraryLoads is allowed
Description:	
The app allows ATS NSAllowsArbitraryLoads and therefore the establishment of insecure HTTP connections.	

Table 7.43: The miscellaneous tool-based scan findings of the Femometer app's vulnerability assessment.

ID:	FEMOMETER-002
Checklist ID:	V1.1
Title:	Lacking information of the PII processing before registration
Description:	
The app does not inform users about its privacy policy before they register.	
ID:	FEMOMETER-003
Checklist ID:	V1.3
Title:	Possibly unnecessary permissions
Description:	
The app's `Info.plist` file does not explain why access to the microphone, location, and Face ID is needed. Access to the microphone and the location do not serve the app's primary purpose.	
ID:	FEMOMETER-004
Checklist ID:	V1.4
Title:	Missing obtainment of declaration of consent
Description:	
The app does not obtain a declaration of consent for processing the data from the users.	
ID:	FEMOMETER-005
Checklist ID:	V1.5
Title:	Missing safety notes for users
Description:	
The app does not recommend any safety measures that users can follow in order to increase the device's security.	
ID:	FEMOMETER-006
Checklist ID:	V1.6
Title:	Missing jailbreak detection
Description:	
The app neither shows a warning nor prevents its usage when it is launched on a jailbroken device.	
ID:	FEMOMETER-007
Checklist ID:	V1.7
Title:	Lacking information of withdrawing the user's consent
Description:	
The app does not explain how users can revoke their consent.	

Table 7.44: The miscellaneous „Design and Privacy“ findings of the Femometer app's vulnerability assessment.

ID:	FEMOMETER-008
Checklist ID:	V3.2
Title:	Inconsistent min. password char length in input fields
Description:	
The registration and password change require a password with eight chars length as well as a mix of letters, numbers, and symbols. However, the login already accepts one char.	
ID:	FEMOMETER-009
Checklist ID:	V3.4
Title:	Missing brute-force login protection
Description:	
The app may not have a security mechanism for the login that prevents brute-force attacks.	

Table 7.45: The miscellaneous „Authentication Management“ findings of the Femometer app's vulnerability assessment.

7.7 Summary of Findings

The elaborated concept was used for a practical vulnerability assessment of the TK, Ada, mySugr, and Femometer apps. It revealed at least one vulnerability in each of the four apps. The found vulnerabilities are summarized in Table 7.46. Almost all apps disclosed information in auto-generated snapshots. Two apps did not support certificate pinning. One app might be vulnerable to stack smashing attacks because no stack canary value was added to the stack. Credentials were found in the NSURLRequest cache during the assessment of one app. Another app stored credentials in the NSUserDefaults and hashed passwords and passcodes with MD5.

In addition, different miscellaneous issues, as shown in Table 7.47 were found in the apps. All apps had a missing jailbreak detection. Therefore, it was possible to launch and use the apps on a jailbroken device. Three apps had misconfigured ATS that might allow insecure connections. Three apps did not recommend any safety measures that users can follow in order to increase the device's security. It was possible to enter a wrong password multiple times in three apps, so they may not have a security mechanism for the login that prevents brute-force attacks. Moreover, two apps did not explain how to report security problems in their apps. Further issues concerned the user interface, permissions, and obtainment of declaration of consent.

App Occurrence	Description	References	CVSS Score
Ada, mySugr, Femometer	Information disclosure in auto-generated snapshots	CWE-200	4.0 (Medium)
Ada, Femometer	Missing certificate pinning	CWE-295	4.0 (Medium)
TK	Missing stack canary value	CWE-121	4.0 (Medium)
Ada	Credentials in NSURLRequest cache	CWE-312, CWE-522, CWE-524	5.5 (Medium)
Femometer	Storage of credentials in NSUserDefaults	CWE-313, CWE-359	5.5 (Medium)
Femometer	Hashing passwords and passcodes with MD5	CWE-916	4.0 (Medium)

Table 7.46: The summarized vulnerabilities of the practical vulnerability assessments.

App Occurrence	Description
TK, Ada, mySugr, Femometer	Missing jailbreak detection
Ada, mySugr, Femometer	Misconfigured ATS
TK, Ada, Femometer	Missing safety notes for users
Ada, mySugr, Femometer	Missing brute-force login protection
mySugr	User interface issues
TK, mySugr	Lacking information on how to report security problems
Femometer	Lacking information of the PII processing before registration
Femometer	Possibly unnecessary permissions
Femometer	Missing obtainment of declaration of consent

Table 7.47: The summarized miscellaneous issues of the practical vulnerability assessments.

CHAPTER 8

Conclusion and Outlook

In summary, this thesis aimed to answer the following questions:

1. What are common vulnerabilities of iOS eHealth applications?
2. How can iOS eHealth application vulnerability testing be conducted?
3. Are vulnerability assessments an effective measure to detect vulnerabilities in iOS eHealth apps?

For the first research question, there are different common vulnerabilities in iOS eHealth applications as outlined in Section 2.2. The OWASP [99] top 10 mobile risks introduces common mobile vulnerabilities, such as improper platform usage, insecure data storage, and insecure communication. Improper platform usage indicates a misused platform feature or a failing platform security control. It includes, for example, platform permission issues and misuse of the iOS keychain as introduced in Section 4.1.4. The iOS keychain is a storage of short bits of data in an encrypted database. It can be used, for example, for storing credentials. Section 2.1 outlines that vulnerabilities arise because of faulty software and system misconfigurations and due to differences in knowledge about secure programming and privacy protection among people who implement and distribute applications. Improper platform usage arises and can lead to vulnerabilities. For example, if a developer does not follow those guidelines or interprets them wrong. The access of the keychain is controlled by an attribute key which is chosen by the developers. If they choose a less restrictive attribute key, it can lead to a vulnerability. As pointed out in Section 2.2.2, mobile devices store and process a lot of sensitive data. The insecure data storage implies that an attacker has physical access to the mobile device or that the attacker's malware is executed on the device. Thus, attackers can access, for example, the local storage of the eHealth applications where they might find

sensitive user data. Therefore, sensitive user data processed by an eHealth app needs to be protected. The insecure data storage arises and can lead to vulnerabilities, for example, when wrong assumptions about access to a device file system are made. As pointed out in Section 2.2.2, attackers can intercept the sent data. Therefore, the data sent and received by an eHealth app needs to be protected. The network communication of the app is considered insecure, for example, if the app has implemented transport security incorrectly.

Vulnerabilities can be found by utilizing security testing techniques as introduced in Chapter 5 which answers the second research question. Security testing techniques include vulnerability scans, vulnerability assessments and penetration testing. Vulnerability scans are automatic tools utilized to detect vulnerabilities. They can recognize deprecated software versions and misconfigurations that can lead to vulnerabilities. Vulnerability assessments help to find security issues in applications using different techniques such as static and dynamic analysis. In contrast to vulnerability assessments, a penetration test aims to prove a vulnerability exists and analyze what information and accesses attackers can retrieve.

Some standards and guidelines support the security testing process of mobile apps, such as the OSSTMM [68], the OWASP MSTG [97] and the OWASP Mobile Top 10 [99]. However, most of the guidelines describe the general reviewing process. They do not consider specific GDPR and eHealth requirements, for example, as summarized by the BSI [72]. Moreover, the architecture of the different mobile operating systems and their applications differ. Hence, a general reviewing process might lead to miss vulnerable parts of the application.

Therefore, this thesis aimed to elaborate a vulnerability assessment concept for iOS eHealth applications. Knowledge of security, mobile devices, applications, eHealth, and iOS fundamentals was obtained to find the most common vulnerabilities of iOS eHealth apps. Moreover, general aspects of security testing were researched. Standards and guidelines such as the OSSTMM [68], the NIST [119], the OWASP [95], the PCI-DSS [114] and the BSI [73] [72] formed the basis of the concept. Based on the literature research, the concept was elaborated. It consists of three phases, namely preassessment, information gathering and vulnerability assessment, and final analysis. The preassessment phase covers the organizational and planning steps of the assessment, such as scope, goals, and testing setup preparation. The concept is designed for binary code and black-box testing because this setup is the closest to an actual attack scenario. The information gathering and vulnerability assessment phase is divided into three parts: information gathering, tool-based scan, and vulnerability assessment. The latter has four parts, namely design and privacy, network communication, authentication management, and data storage. Checklists inspired by the OSSTMM, the BSI, and the OWASP were added to ensure that no common vulnerabilities are missed in the assessment. In the last phase, the final analysis, the findings of the previous phase are summarized and categorized into vulnerabilities and miscellaneous issues.

The elaborated vulnerability assessment concept was used for practical assessments

of chosen iOS eHealth applications to validate if it is an effective measure to detect vulnerabilities and hence, to answer the third research question. eHealth apps that were recently mentioned in the media or advertisements were selected for the practical assessment. In the end, four eHealth apps, namely, Ada, mySugr, Femometer, including the TK ePA app were chosen. A proper black-box vulnerability assessment of the ePA app was not possible since the registration required being a member of the insurance. Hence, many checklist items, such as those from the „Data Storage“, could not be checked. The reviewed eHealth iOS applications implemented different security mechanisms to protect the user data. For example, the Ada app prevented its usage on a device with an insecure network connection, such as when the Burp proxy was used. The mySugr and TK apps used certificate pinning, where the app verifies the server's authenticity and only accepts specific, trusted certificates. As a result, intercepting requests with Burp was not possible. However, the vulnerability assessment of the apps showed that it was possible to disable most of their security mechanisms. As a result, requests between app and server could be analyzed, and user data stored in the „Data“ directories could be examined. The vulnerability assessment has found that some apps store or cache sensitive user data in the local device storage, which is considered insecure. Moreover, some apps had a misconfigured ATS that might allow insecure connections. Furthermore, one app might be vulnerable to stack buffer overflows. Another app used MD5 for hashing passwords and passcodes, which is considered insecure.

Since only iOS applications were examined, no statements can be made for the corresponding Android applications. A vulnerability assessment of eHealth Android applications could be the subject of future works. Furthermore, the covered areas in the vulnerability assessment's scope can be extended in future works using other testing methods such as reverse engineering. Moreover, the iOS vulnerability assessment scope did not include the apps' backend servers and active attacks. A pentest requires an agreement between the tester and the app publisher. Therefore, future works could cooperate with the reviewed applications in this thesis to conduct a penetration test to verify the found vulnerabilities in the assessments. For example, the Femometer app used MD5 for hashing passwords and passcodes. The passwords were transmitted in this form to the server. Therefore, the app might store the MD5 hashed passwords directly in their database. Moreover, the intercepted requests in Burp were not manipulated, for example, in order to test if the server exposes confidential user data. Furthermore, it could not be verified if the TK app's stack buffer overflow vulnerability is exploitable. The analysis of common vulnerabilities such as insecure data storage often follows the same steps and is often simple to verify. Therefore, future works could automate the testing processes, and existing tool-based scanner tools like MobSF could be extended.

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 10, 17, 19, 25

Apps Applications. 1, 5, 8

ARC Automatic Reference Counting. 52, 66, 76, 85, 96, 105

ASLR Address Space Layout Randomisation. 19, 20, 52

ATS App Transport Security. 21, 51, 52, 66, 76, 84, 85, 93, 96, 105, 112, 115, 118, 119

BSI The German Federal Office for Information Security. 2, 14, 15, 43, 46–50, 119, 121

bundle ID bundle identifier. 27, 30, 31

CIA confidentiality, integrity and availability. 5, 9, 12

CVE Common Vulnerabilities and Exposures. 7, 59

CVSS Common Vulnerability Scoring System. 7, 8, 59

CWE Common Weakness Enumeration. 7, 36

DoS Denial of Service. 9, 13

DTLS Datagram Transport Layer Security. 21

eHealth electronic health. 1–5, 11–15, 41, 42, 53–56, 62, 65, 117–119

EHR electronic health record. 1, 5, 11, 13, 14

ePA German electronic health records. 1, 3, 13, 14, 61, 65, 66, 75, 117

EU European Union. 11, 12

FdV Insurant Frontend. 14

FIRST Forum of Incident Response and Security Teams. 8

GDPR General Data Protection Regulation. 11, 12, 53, 66, 76

gematik Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH. 1, 13, 14, 61

HTTP Hypertext Transfer Protocol. 12, 84, 85, 93, 105, 112

HTTPS Hypertext Transfer Protocol Secure. 12, 85

ID identifier. 29, 32, 50, 51, 53, 54, 59, 100

IEC International Electro-technical Commission. 43

IPA iOS App Store Package. 29, 30, 50, 51, 63, 64, 119

IPC Inter-Process Communication. 19

ISECOM Institute for Security and Open Methodologies. 43

ISO International Organization for Standardization. 43, 62

MASVS Mobile Application Security Verification Standard. 44–46, 119

mHealth mobile health. 11, 46

MitM Man-in-the-Middle. 9

MobSF Mobile Security Framework. 51, 64

MSTG Mobile Security Testing Guide. 44, 45, 117, 119

N/A not available. 53, 68, 72, 76, 81, 86, 87, 91

NFC Near Field Communication. 8

NIST National Institute of Standards and Technology. 43–45, 119

NOK not ok. 51, 53, 58, 72, 75, 76, 84, 86, 90, 91, 95, 97, 100, 104, 106, 109

NVD National Vulnerability Database. 7, 8, 44

NX No eXecute. 52, 66, 76, 85, 96, 105

. 43

OSSTMM Open Source Security Testing Methodology Manual. 43, 45, 50, 117

OWASP Open Web Application Security Project. 7, 9, 10, 36, 41, 43–46, 49, 50, 53, 57, 59, 117, 119

PCI-DSS Payment Card Industry Data Security Standards. 43, 45

PFS Perfect Forward Secrecy. 93, 102

PIE Position-Independent Executable. 52, 66, 76, 85, 96, 105

PII Personally Identifiable Information. 10, 11, 54, 55, 70, 79, 88, 99, 106, 107, 113, 116

plist Property List. 27, 32

ROM Read-Only Memory. 18

SDK Software Development Kit. 26

SSL Secure Sockets Layer. 11, 64

STAR Security Test Audit Report. 43

TK Techniker Krankenkasse. 61, 74–83, 117, 118, 120–122

TLS Transport Layer Security. 11, 15, 21, 52, 55, 56, 84, 105

XN Execute Never. 20

Bibliography

[1] Sara Alwahedi et al. „Security in Mobile Computing: Attack Vectors, Solutions, and Challenges“. In: *Mobile Networks and Management*. Ed. by Ramón Agüero et al. Cham: Springer International Publishing, 2017, pp. 177–191.

[47] Yan Bai et al. „Issues and Challenges in Securing eHealth Systems“. In: *International Journal of E-Health and Medical Communications* 5 (July 2014), pp. 1–19. DOI: 10.4018/ijehmc.2014010101.

[48] Rafay Baloch. *Ethical hacking and penetration testing guide*. Boca Raton: CRC Press, 2015. ISBN: 9781482231625.

[49] Lisa Ann Baumann et al. „The impact of electronic health record systems on clinical documentation times: A systematic review“. In: *Health Policy* 122.8 (2018), pp. 827–836. DOI: 10.1016/j.healthpol.2018.05.014.

[50] Paul E. Black et al. „Dramatically Reducing Software Vulnerabilities: Report to the White House Office of Science and Technology Policy“. In: *NIST Interagency/Internal Report (NISTIR)*. Gaithersburg, MD: National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.IR.8151.

[51] Sriramulu Bojjagani and V. N. Sastry. „VAPTAi: A Threat Model for Vulnerability Assessment and Penetration Testing of Android and iOS Mobile Banking Apps“. In: *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*. San Jose, CA, USA: IEEE, 2017, pp. 77–86. DOI: 10.1109/CIC.2017.00022.

[52] Sumedha Chauhan and Mahadeo Jaiswal. „A meta-analysis of e-health applications acceptance: Moderating impact of user types and e-health application types“. In: *Journal of Enterprise Information Management* 30 (Mar. 2017). DOI: 10.1108/JEIM-08-2015-0078.

[53] Dominic Chell et al. *The Mobile Application Hacker's Handbook*. Indianapolis: John Wiley & Sons, 2015, pp. 1–15. ISBN: 978-1-118-95850-6.

[56] Tobias Dehling and Ali Sunyaev. „Information Security and Privacy of Patient-Centered Health IT Services: What Needs to Be Done?“ In: *2014 47th Hawaii International Conference on System Sciences*. Waikoloa, HI, USA: IEEE, 2014, pp. 2984–2993. DOI: 10.1109/HICSS.2014.371.

[57] Frank Eickmeier. *E-Health: Datenschutz und Datensicherheit: Herausforderungen und Lösungen im IoT-Zeitalter*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. DOI: 10.1007/978-3-658-15091-4_4.

[58] Dmitry Evtyushkin et al. „Jump over ASLR: Attacking branch predictors to bypass ASLR“. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Taipei, Taiwan: IEEE, 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783743.

[59] Muhammad Farooq et al. „A Critical Analysis on the Security Concerns of Internet of Things (IoT)“. In: *International Journal of Computer Applications* 111 (Feb. 2015), pp. 1–6. DOI: 10.5120/19547-1280.

[60] José Luis Fernández-Alemán et al. „Security and privacy in electronic health records: A systematic literature review“. In: *Journal of Biomedical Informatics* 46.3 (2013), pp. 541–562. DOI: 10.1016/j.jbi.2012.12.003.

[62] Shivi Garg and Niyati Baliyan. „Comparative analysis of Android and iOS from security viewpoint“. In: *Computer Science Review* 40 (2021), p. 100372. DOI: 10.1016/j.cosrev.2021.100372.

[67] Jai Narayan Goel and B.M. Mehtre. „Vulnerability Assessment & Penetration Testing as a Cyber Defence Technology“. In: *Procedia Computer Science* 57 (2015). 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015), pp. 710–715. DOI: 10.1016/j.procs.2015.07.458.

[69] Alexander Hoerbst et al. „Attitudes and behaviors related to the introduction of electronic health records among Austrian and German citizens“. In: *International Journal of Medical Informatics* 79.2 (2010), pp. 81–89. ISSN: 1386-5056. DOI: 10.1016/j.ijmedinf.2009.11.002.

[75] Samuel Tusubira Kalyango and Gilbert Maiga. „A Technique for Strengthening Weak Passwords in Electronic Medical Record Systems“. In: *Foundations of Health Informatics Engineering and Systems*. Ed. by Zhiming Liu and Alan Wassyng. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 207–224. DOI: 10.1007/978-3-642-32355-3_13.

[76] Ansgar Kellner et al. „False Sense of Security: A Study on the Effectivity of Jailbreak Detection in Banking Apps“. In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. Stockholm, Sweden: IEEE, 2019, pp. 1–14. DOI: 10.1109/EuroSP.2019.00011. (Visited on 10/11/2021).

[77] Yugansh Khera et al. „Analysis and Impact of Vulnerability Assessment and Penetration Testing“. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. Faridabad, India: IEEE, 2019, pp. 525–530. DOI: 10.1109/COMITCon.2019.8862224.

[80] Kevin Languedoc. *Build iOS Database Apps with Swift and SQLite*. Berkeley, CA: Apress, 2016. DOI: 10.1007/978-1-4842-2232-4_1.

[81] Feng Liu et al. „Research on the Technology of iOS Jailbreak“. In: *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*. Harbin, China: IEEE, 2016, pp. 644–647. DOI: 10.1109/IMCCC.2016.178. (Visited on 10/11/2021).

[82] Kaiping Liu et al. „Binary Code Analysis“. In: *Computer* 46.8 (2013), pp. 60–68. DOI: 10.1109/MC.2013.268.

[83] Isabel Maria Lopes and Pedro Oliveira. „Implementation of the general data protection regulation: A survey in health clinics“. In: *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. Caceres, Spain: IEEE, 2018, pp. 1–6. DOI: 10.23919/CISTI.2018.8399156.

[84] Mădălina Angelica Marin et al. „Proactive Secure Coding for iOS Applications“. In: *2019 18th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. Galati, Romania: IEEE, 2019, pp. 1–5. DOI: 10.1109/ROEDUNET.2019.8909672.

[85] Nir Menachemi and Taleah Collum. „Benefits and drawbacks of electronic health record systems“. In: *Risk management and healthcare policy* 4 (May 2011), pp. 47–55. DOI: 10.2147/RMHP.S12985.

[87] Ovidiu Constantin Novac et al. „Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems“. In: *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*. Oradea, Romania: IEEE, 2017, pp. 154–159. DOI: 10.1109/EMES.2017.7980403.

[88] Michael Ogata et al. *Vetting the security of mobile applications*. Tech. rep. National Institute of Standards and Technology, 2019. DOI: 10.6028/NIST.SP.800-163r1.

[107] Ahd Radwan and Samer Zein. „Model-Based Approach for Supporting Quick Caching at iOS Platform“. In: *International Journal of Advanced Trends in Computer Science and Engineering* 9 (Sept. 2020), pp. 4285–4294. DOI: 10.30534/ijatcse/2020/17942020.

[108] Kunal Relan. *iOS Penetration Testing: A Definitive Guide to iOS Security*. 1st. USA: Apress, 2016. DOI: 10.1007/978-1-4842-2355-0.

[109] Jung Hyun Ryu et al. „Analysis of a third-party application for mobile forensic investigation.“ In: *JIPS* 14.3 (2018), pp. 680–693.

[110] Suhasini Sabnis and Doug Charles. „Opportunities and challenges: Security in ehealth“. In: *Bell Labs Technical Journal* 17.3 (2012), pp. 105–111. DOI: 10.1002/bltj.21561.

[111] Tony Sahama et al. „Security and Privacy in eHealth: Is it possible?“ In: *2013 IEEE 15th International Conference on e-Health Networking, Applications and Services (Healthcom 2013)*. Lisbon, Portugal: IEEE, 2013, pp. 249–253. DOI: 10.1109/HealthCom.2013.6720676.

[112] Aqeel Sahi et al. „A review of the state of the arts in privacy and security in the eHealth cloud“. In: *IEEE Access* (2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3098708.

[113] Brinda Hansraj Sampat and Bala Prabhakar. „Privacy risks and security threats in mHealth apps“. In: *Journal of International Technology and Information Management* 26.4 (2017), pp. 126–153.

[114] Sugandh Shah and Babu Mehtre. „A modern approach to cyber security analysis using vulnerability assessment and penetration testing“. In: *Int J Electron Commun Comput Eng* 4.6 (2013), pp. 47–52.

[115] Sugandh Shah and Babu Mehtre. „An overview of vulnerability assessment and penetration testing techniques“. In: *Journal of Computer Virology and Hacking Techniques* 11 (Feb. 2014), pp. 27–49. DOI: 10.1007/s11416-014-0231-x.

[116] Hessa Mohammed Zaher Al Shebli and Babak D. Beheshti. „A study on penetration testing process and tools“. In: *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. Farmingdale, NY, USA: IEEE, 2018, pp. 1–7. DOI: 10.1109/LISAT.2018.8378035.

[117] Prashant S. Shinde and Shrikant B. Ardhapurkar. „Cyber security analysis using vulnerability assessment and penetration testing“. In: *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*. Coimbatore, India: IEEE, 2016, pp. 1–5. DOI: 10.1109/STARTUP.2016.7583912.

[118] Kumar Pal Singh et al. „Classification of Data to Enhance Data Security in Cloud Computing“. In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. Bhimtal, India: IEEE, 2018, pp. 1–5. DOI: 10.1109/IoT-SIU.2018.8519934.

[120] Rohit Tamma et al. *Practical Mobile Forensics*. 4rd. UK: Packt Publishing, 2020. ISBN: 9781838647520.

[121] David Thiel. *IOS Application Security: The Definitive Guide for Hackers and Developers*. San Francisco: No Starch Press, 2016.

[123] Sachin Umrao et al. „Vulnerability Assessment and Penetration Testing“. In: *International Journal of Computer & Communication Technology (IJCCT) ISSN (ONLINE): 2231 - 0371* 3 (Jan. 2012), pp. 71–74. DOI: 10.47893/IJCCT.2016.1367.

[125] Vijay Kumar Velu. *Mobile Application Penetration Testing*. Birmingham: Packt Publishing Ltd, 2016. ISBN: 978-1-78588-337-8.

[126] Tielei Wang et al. „Jekyll on iOS: When Benign Apps Become Evil“. In: *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 559–572. ISBN: 978-1-931971-03-4.

[127] Irfan Yaqoob et al. „Penetration testing and vulnerability assessment“. In: *Journal of Network Communications and Emerging Technologies (JNCET) www. jncet. org* 7.8 (2017).

Online Resources

[2] Apple. *Apple App Store*. 2021. URL: https://www.apple.com/in/app-store/ (visited on 10/11/2021).

[3] Apple. *Apple Developer Enterprise Program*. 2021. URL: https://developer.apple.com/programs/enterprise/ (visited on 10/11/2021).

[4] Apple. *Apple Developer Program*. 2021. URL: https://developer.apple.com/programs/ (visited on 10/11/2021).

[5] Apple. *Boot process for iOS and iPadOS devices*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/boot-process-for-ios-and-ipados-devices-secb3000f149/web (visited on 10/11/2021).

[6] Apple. *Bundle ID Capabilities*. 2021. URL: https://developer.apple.com/documentation/appstoreconnectapi/bundle_id_capabilities (visited on 10/11/2021).

[7] Apple. *Bundle IDs*. 2021. URL: https://developer.apple.com/documentation/appstoreconnectapi/bundle_ids (visited on 10/11/2021).

[8] Apple. *Core Data*. 2021. URL: https://developer.apple.com/documentation/coredata (visited on 10/11/2021).

[9] Apple. *Data Protection classes*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/data-protection-classes-secb010e978a/1/web/1 (visited on 10/11/2021).

[10] Apple. *Data Protection overview*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/data-protection-overview-secf6276da8a/web (visited on 10/11/2021).

[11] Apple. *Data Vault*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/aside/sec7d851b00c/1/web/1 (visited on 10/11/2021).

[12] Apple. *Defining a Custom URL Scheme for Your App*. 2021. URL: https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app (visited on 10/11/2021).

[13] Apple. *Encrypting Your App's Files*. Feb. 18, 2021. URL: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files (visited on 10/11/2021).

[14] Apple. *Encryption and Data Protection overview*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/encryption-and-data-protection-overview-sece3bee0835/1/web/1 (visited on 10/11/2021).

[15] Apple. *Entitlements*. Feb. 18, 2021. URL: https://developer.apple.com/documentation/bundleresources/entitlements (visited on 10/11/2021).

[16] Apple. *Further protections*. Feb. 18, 2021. URL: https://support.apple.com/en-gb/guide/security/sec7a12328b3/web (visited on 10/11/2021).

[17] Apple. *How developers sign their apps*. 2021. URL: https://support.apple.com/en-gb/guide/security/sec7af2d5f20/web (visited on 10/11/2021).

[18] Apple. *Identity Pinning: How to configure server certificates for your app*. 2021. URL: https://developer.apple.com/news/?id=g9ejcf8y (visited on 10/11/2021).

[19] Apple. *Information Property List*. 2021. URL: https://developer.apple.com/documentation/bundleresources/information_property_list (visited on 10/11/2021).

[20] Apple. *Introduction to Apple platform security*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/intro-to-apple-platform-security-seccd5016d31/1/web/1 (visited on 10/11/2021).

[21] Apple. *iOS 14*. 2021. URL: https://support.apple.com/en-us/HT211808 (visited on 10/11/2021).

[22] Apple. *Item Attribute Keys and Values*. 2021. URL: https://developer.apple.com/documentation/security/keychain_services/keychain_items/item_attribute_keys_and_values (visited on 10/11/2021).

[23] Apple. *Keychain data protection*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/keychain-data-protection-secb0694df1a/1/web/1 (visited on 10/11/2021).

[24] Apple. *Keychain Services*. 2021. URL: https://developer.apple.com/documentation/security/keychain_services (visited on 10/11/2021).

[25] Apple. *Managing Your App's Information Property List*. 2021. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/managing_your_app_s_information_property_list (visited on 10/11/2021).

[26] Apple. *Mandatory code signing*. Feb. 18, 2021. URL: https://support.apple.com/en-gb/guide/security/sec7c917bf14/web (visited on 10/11/2021).

[27] Apple. *NSExceptionAllowsInsecureHTTPLoads*. 2020. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/nsexceptionallowsinsecurehttploads (visited on 10/11/2021).

[28] Apple. *NSURLCache*. 2021. URL: https://developer.apple.com/documentation/foundation/nsurlcache (visited on 10/11/2021).

[29] Apple. *NSURLRequest*. 2021. URL: https://developer.apple.com/documentation/foundation/nsurlrequest (visited on 10/11/2021).

[30] Apple. *NSUserDefaults*. 2021. URL: https://developer.apple.com/documentation/foundation/nsuserdefaults (visited on 10/11/2021).

[31] Apple. *Preserving Your App's UI Across Launches*. 2021. URL: https://developer.apple.com/documentation/uikit/view_controllers/preserving_your_app_s_ui_across_launches (visited on 10/11/2021).

[32] Apple. *Preventing Insecure Network Connections*. 2021. URL: https://developer.apple.com/documentation/security/preventing_insecure_network_connections (visited on 10/11/2021).

[33] Apple. *Profiles*. 2021. URL: https://developer.apple.com/documentation/appstoreconnectapi/profiles (visited on 10/11/2021).

[34] Apple. *Protected Resources*. 2021. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/protected_resources (visited on 10/11/2021).

[35] Apple. *Protecting app access to user data*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/protecting-app-access-to-user-data-secc01781f46/1/web/1 (visited on 10/11/2021).

[36] Apple. *Protecting the User's Privacy*. 2021. URL: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy (visited on 10/11/2021).

[37] Apple. *Requesting Access to Protected Resources*. 2021. URL: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/requesting_access_to_protected_resources (visited on 10/11/2021).

[38] Apple. *Secure software updates*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/secure-software-updates-secf683e0b36/1/web/1 (visited on 10/11/2021).

[39] Apple. *Security of runtime process in iOS and iPadOS*. 2021. URL: https://support.apple.com/en-gb/guide/security/sec15bfe098e/web (visited on 10/11/2021).

[40] Apple. *Swift*. 2021. URL: https://developer.apple.com/swift/ (visited on 10/11/2021).

[41] Apple. *System security overview*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/system-security-overview-sec114e4db04/1/web/1 (visited on 10/11/2021).

[42] Apple. *TLS security*. Feb. 18, 2021. URL: https://support.apple.com/guide/security/tls-security-sec100a75d12/1/web/1 (visited on 10/11/2021).

[43] Apple. *Use of entitlements*. Feb. 18, 2021. URL: https://support.apple.com/en-gb/guide/security/sec6a57885b6/web (visited on 10/11/2021).

[44] Apple. *WebKit*. 2021. URL: https://developer.apple.com/documentation/webkit (visited on 10/11/2021).

[45] Apple. *What's New in the iOS SDK*. 2021. URL: https://developer.apple.com/ios/whats-new/ (visited on 10/11/2021).

[46] Apple. *Xcode 13*. 2021. URL: https://developer.apple.com/xcode/ (visited on 10/11/2021).

[54] European Commission. *What information must be given to individuals whose data is collected?* URL: https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/principles-gdpr/what-information-must-be-given-individuals-whose-data-collected_en (visited on 10/11/2021).

[55] Penetration Test Guidance Special Interest Group PCI Security Standards Council. *Penetration Testing Guidance*. 2015. URL: https://www.pcisecuritystandards.org/documents/Penetration-Testing-Guidance-v1_1.pdf (visited on 10/11/2021).

[61] Professor John Franks et al. *HTTP Authentication: Basic and Digest Access Authentication*. June 1999. DOI: 10.17487/RFC2617. URL: https://rfc-editor.org/rfc/rfc2617.txt.

[63] gematik. *Die elektronische Patientenakte (ePA)*. 2021. URL: https://www.gematik.de/anwendungen/e-patientenakte/ (visited on 10/11/2021).

[64] gematik. *Einheitliche elektronische Patientenakte für das deutsche Gesundheitssystem*. Dec. 19, 2018. URL: https://www.gematik.de/news/news/einheitliche-elektronische-patientenakte-fuer-das-deutsche-gesundheitssystem/ (visited on 10/11/2021).

[65] gematik. *Patienten*. 2021. URL: https://www.gematik.de/anwendungen/e-patientenakte/patienten/ (visited on 10/11/2021).

[66] Hartmut Gieselmann et al. *Massive Datenschutzmängel in der Gesundheits-App Ada*. Oct. 11, 2019. URL: https://heise.de/-4549354 (visited on 10/11/2021).

[68] Pete Herzog. *The Open Source Security Testing Methodology Manual*. 2010. URL: https://www.isecom.org/OSSTMM.3.pdf (visited on 10/11/2021).

[70] Forum of Incident Response and Security Teams. *Common Vulnerability Scoring System Version 3.1 Calculator*. 2021. URL: https://www.first.org/cvss/calculator/3.1 (visited on 10/11/2021).

[71] Bundesamt für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlusselllängen*. Mar. 24, 2021. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile (visited on 10/11/2021).

[72] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheitsanforderungen an digitale Gesundheitsanwendungen*. Apr. 15, 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03161/BSI-TR-03161.pdf (visited on 10/11/2021).

[73] Bundesamt für Sicherheit in der Informationstechnik. *Studie Penetrationstests*. Nov. 22, 2003. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest.pdf (visited on 10/11/2021).

[74] JWT. *JWT*. 2021. URL: https://jwt.io (visited on 10/11/2021).

[78] Krankenkassen. *Die größten Krankenkassen: Versicherte 2021*. 2021. URL: https://www.krankenkassen.de/krankenkassen-vergleich/statistik/versicherte/aktuell/ (visited on 10/11/2021).

[79] KrankenkassenInfo. *Krankenversichertennummer*. 2021. URL: https://www.krankenkasseninfo.de/zahlen-fakten/lexikon/krankenversichertennummer (visited on 10/11/2021).

[86] EU eHealth Network. *Mobile applications to support contact tracing in the EU's fight against COVID-19: Common EU Toolbox for Member States*. Apr. 15, 2020. URL: https://ec.europa.eu/health/sites/default/files/ehealth/docs/covid-19_apps_en.pdf (visited on 10/11/2021).

[89] OWASP. *Attacks*. 2021. URL: https://owasp.org/www-community/attacks/ (visited on 10/11/2021).

[90] OWASP. *Blocking Brute Force Attacks*. 2021. URL: https://owasp.org/www-community/Threat_Modeling (visited on 10/11/2021).

[91] OWASP. *Blocking Brute Force Attacks*. 2021. URL: https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks (visited on 10/11/2021).

[92] OWASP. *Code Quality and Build Settings for iOS Apps*. Aug. 11, 2019. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06i-testing-code-quality-and-build-settings (visited on 10/11/2021).

[93] OWASP. *Data Storage on iOS*. 2019. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06d-testing-data-storage (visited on 10/11/2021).

[94] OWASP. *M9: Reverse Engineering*. 2021. URL: https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering (visited on 10/11/2021).

[95] OWASP. *Mobile App Security Testing*. 2019. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/overview/0x04b-mobile-app-security-testing (visited on 10/11/2021).

[96] OWASP. *Mobile Application Security Verification Standard*. Mar. 17, 2020. URL: https://github.com/OWASP/owasp-masvs/releases/tag/v1.3 (visited on 10/11/2021).

[97] OWASP. *Mobile Security Testing Guide*. Aug. 11, 2019. URL: https://github.com/OWASP/owasp-mstg/releases/tag/v1.2 (visited on 10/11/2021).

[98] OWASP. *OWASP Mobile Top 10*. 2016. URL: https://owasp.org/www-project-mobile-top-10 (visited on 10/11/2021).

[99] OWASP. *Top 10 Mobile Risks - Final List 2016*. 2016. URL: https://owasp.org/www-project-mobile-top-10/2016-risks/ (visited on 10/11/2021).

[100] Julia Palmai. *Alles im Griff: Blutzuckerkontrolle im digitalen Zeitalter*. Nov. 24, 2020. URL: https://www.derstandard.de/story/2000121836566/alles-im-griff-blutzuckerkontrolle-im-digitalen-zeitalter (visited on 10/11/2021).

[101] PortSwigger. *Burp Suite*. 2021. URL: https://portswigger.net/burp (visited on 10/11/2021).

[102] Cambridge University Press. *app Meaning in the Cambridge English Dictionary*. 2021. URL: https://dictionary.cambridge.org/us/dictionary/english/app (visited on 10/11/2021).

[103] Cambridge University Press. *attack Meaning in the Cambridge English Dictionary*. 2021. URL: https://dictionary.cambridge.org/us/dictionary/english/attack (visited on 10/11/2021).

[104] Cambridge University Press. *exploit Meaning in the Cambridge English Dictionary*. 2021. URL: https://dictionary.cambridge.org/de/worterbuch/englisch/exploit (visited on 10/11/2021).

[105] Cambridge University Press. *mobile device Meaning in the Cambridge English Dictionary*. 2021. URL: https://dictionary.cambridge.org/us/dictionary/english/mobile-device (visited on 10/11/2021).

[106] Cambridge University Press. *threat Meaning in the Cambridge English Dictionary*. 2021. URL: https://dictionary.cambridge.org/us/dictionary/english/threat (visited on 10/11/2021).

[119] Murugiah Souppaya and Karen Scarfone. *Technical Guide to Information Security Testing and Assessment*. en. Sept. 30, 2008. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=152164 (visited on 10/11/2021).

[122] Martin Tschirsich and Thorsten Schröder. *Schwachstellen in Gesundheits-App Vivy*. Jan. 14, 2018. URL: https://www.modzero.com/static/vivy-app-security-final.pdf (visited on 10/11/2021).

[124] SGB V. *§ 291a SGB V Elektronische Gesundheitskarte als Versicherungsnachweis und Mittel zur Abrechnung*. Jan. 18, 2021. URL: https://www.sozialgesetzbuch-sgb.de/sgbv/291a.html (visited on 10/11/2021).

Appendix

iOS Vulnerability Assessment Report Template

The following document shows the iOS vulnerability assessment report template introduced in Chapter 6.

iOS Vulnerability Assessment Report

Date:	
Assessment Approach:	
Test Method:	

Test Classification	
Criteria	**Value**
Information Basis:	
Aggressiveness:	
Scope:	
Approach:	
Technique:	
Starting Point:	

Information Gathering

ID	Description	
I1	**Evaluation of publicly available data**	
I1.1	App Name:	
I1.2	AppStore Link:	
I1.3	Version:	
I1.4	Application category:	
I1.5	IOS compatibility:	
I1.6	Purpose of the application:	
I1.7	Collected data linked to the users identity:	
I2	**Information about the application publisher**	
I2.1	Publisher name:	
I2.2	Publisher website:	
I2.3	Developer name:	
I2.4	Developer website:	

Tool-Based Scan

ID	Description		Status
T3	**Information about the system**		
T1.1	Programming languages:		
T1.2	App Identifier:		
T1.3	App Permissions:		
T2	**App Transport Security**		
T2.1	ATS is enabled. No insecure connections are configured.		
T3	**IPA Binary Analysis**		
T3.1	ARC is enabled.		
T3.4	The NX bit is set in the binary.		
T3.5	The stack canary value is added to the stack of the binary.		
T3.6	PIE support is activated.		

Vulnerability Assessment

ID	Description	Status
V1	**Design and Privacy**	
V1.1	The app discloses its primary purpose and the use of personal data before installation (e. g., in the description of the app store). The user is informed about this, at least when the app is first launched.	
V1.2	The app only collects and processes data that serves its primary purpose.	
V1.3	The app only requests the minimum set of permissions necessary for its primary purpose.	
V1.4	The app obtains a declaration of consent from the user before collecting or processing any PII.	
V1.5	The app educates about security best practices the user should follow in using the app (e. g., using a passcode)	
V1.6	The app detects a jailbroken device and warns the user by listing the risks for user data.	
V1.7	The app allows the withdrawl of the user's consent. Information is provided on how this changes the behavior of the app.	
V1.8	The user can enable push notifications that might contain sensitive data. This option is deactivated by default.	
V1.9	The application provides a low-barrier option to report security problems.	
V1.10	Error messages and notifications do not contain sensitive data.	
V1.11	No sensitive data (e. g., passwords, pins) is exposed through the user interface.	
V2	**Network Communication**	
V2.1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	
V2.2	The app verifies the certificate of the remote endpoint when the secure channel is established.	
V2.3	The app supports certificate pinning.	

ID	Description	Status
V3	**Authentication Management**	
V3.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	
V3.2	A password policy exists and is enforced.	
V3.3	Users can change their passwords.	
V3.4	Several incorrect login attempts lead to security measures like delays in subsequent login attempts.	
V4	**Data Storage**	
V4.1	System credential storage facilities need to be used to store sensitive data. They are sufficiently protected.	
V4.2	Insecure storages do not include sensitive user data and information like passwords or tokens.	
V4.3	The app removes sensitive data from views when moved to the background, and the app's cache does not expose sensitive data.	
V4.4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	

Vulnerabilities

ID:	
Checklist ID:	
CVSS Score:	
References:	
Title:	
Description:	

Miscellaneous

ID:	
Checklist ID:	
Title:	
Description:	